Till Mossakowski
Ugo Montanari
Magne Haveraaen (Eds.)

LNCS 4624

Algebra and Coalgebra in Computer Science

Second International Conference, CALCO 2007
Bergen, Norway, August 2007
Proceedings

Lecture Notes in Computer Science 4624

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

Editorial Board

David Hutchison
Lancaster University, UK
Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA
Josef Kittler
University of Surrey, Guildford, UK
Jon M. Kleinberg
Cornell University, Ithaca, NY, USA
Friedemann Mattern
ETH Zurich, Switzerland
John C. Mitchell
Stanford University, CA, USA
Moni Naor
Weizmann Institute of Science, Rehovot, Israel
Oscar Nierstrasz
University of Bern, Switzerland
C. Pandu Rangan
Indian Institute of Technology, Madras, India
Bernhard Steffen
University of Dortmund, Germany
Madhu Sudan
Massachusetts Institute of Technology, MA, USA
Demetri Terzopoulos
University of California, Los Angeles, CA, USA
Doug Tygar
University of California, Berkeley, CA, USA
Moshe Y. Vardi
Rice University, Houston, TX, USA
Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Till Mossakowski Ugo Montanari
Magne Haveraaen (Eds.)

Algebra and Coalgebra in Computer Science

Second International Conference, CALCO 2007
Bergen, Norway, August 20-24, 2007
Proceedings

Springer

Volume Editors

Till Mossakowski
Deutsches Forschungszentrum für künstliche Intelligenz (DFKI)
Safe & Secure Cognitive Systems
28359 Bremen, Germany
E-mail: Till.Mossakowski@dfki.de

Ugo Montanari
Università di Pisa
Dipartimento di Informatica
56127 Pisa, Italy
E-mail: ugo@di.unipi.it

Magne Haveraaen
Universitetet i Bergen
Institutt for Informatikk
Postboks 7800, 5020 Bergen, Norway
E-mail: Magne.Haveraaen@ii.uib.no

Library of Congress Control Number: 2007931881

CR Subject Classification (1998): F.3.1, F.4, D.2.1, I.1

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

ISSN 0302-9743
ISBN-10 3-540-73857-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-73857-2 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2007
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12094387 06/3180 5 4 3 2 1 0

Preface

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

The first, and very successful, CALCO conference took place in 2005 in Swansea, Wales. The second CALCO took place in 2007 in Bergen, Norway, and was organized by Magne Haveraaen (Chair), Yngve Lamo, Michal Walicki, and Uwe Wolter.

The CALCO Steering Committee consists of Jiří Adámek, Michel Bidoit, Corina Cirstea, José Fiadeiro (Co-chair), H.Peter Gumm, Magne Haveraaen, Bart Jacobs, Hans-Jörg Kreowski, Alexander Kurz, Ugo Montanari, Larry Moss, Till Mossakowski, Peter Mosses, Fernando Orejas, Francesco Parisi-Presicce, John Power, Horst Reichel, Markus Roggenbach, Jan Rutten (Co-chair), and Andrzej Tarlecki.

CALCO 2007 received 57 submissions (including four tool papers), out of which 26 (including two tool papers) were selected for presentation at the conference. Each submission received three or four reviews of high quality. We want to thank the Program Committee and the additional reviewers, who brought in their competence and expertise. They are listed at the end of this preface. The discussion and decision-making took place in March 2007. As for CALCO 2005, all submissions by PC members were accepted only in the case of unanimous agreement, and the decisions were reached without the PC members involved being aware that their papers were under discussion.

The revised papers can be found in this volume, which also includes the papers contributed by the invited speakers: Stephen L. Bloom, Luís Caires, Barbara König, and Glynn Winskel. We wish to express our warmest thanks to all of them.

The technical program of CALCO 2007 was preceded by the CALCO Young Researchers Workshop, CALCO-jnr, dedicated to presentations by PhD students and by those who completed their doctoral studies within the past few years. CALCO-jnr was organized by Magne Haveraaen, John Power, and Monika Seisenberger.

The successful application of algebraic and coalgebraic techniques in practice depends on the availability of good tools. The CALCO-tools workshop, organized by Narciso Marti-Oliet and Grigore Roşu, provided presentations of such tools.

During the presentations, extra time for demonstrations of the running systems was allotted.

The organizers would like to thank Rolf Rosé Jensen for designing the poster, Karl Trygve Kalleberg for designing and creating the CALCO Web pages, and Adrian Rutle for creating and administrating the registration system. Support from IFIP WG1.3 on Foundations of System Specification, Research Council of Norway, Bergen University College, Norway and Department of Informatics, University of Bergen, Norway is gratefully acknowledged.

At Springer, Alfred Hofmann and his team supported the publishing process. The activity of the PC was supported by the Conference Online Service from Dortmund University; and Martin Karusseit patiently answered our numerous questions and problems. Our deepest thanks go to all of them and, last but not least, to all the authors for providing the high-quality contributions that made CALCO 2007 such a successful event.

June 2007

Till Mossakowski
Ugo Montanari
Magne Haveraaen

Organization

Program Committee

Jiří Adámek, University of Braunschweig, Germany
José Fiadeiro, University of Leicester, UK
H.Peter Gumm, Philipps University, Marburg, Germany
Bartek Klin, University of Warsaw, Poland
Bart Jacobs, University of Nijmegen, The Netherlands
Marina Lenisa, University of Udine, Italy
Ugo Montanari, University of Pisa, Italy (Co-chair)
Larry Moss, Indiana University, Bloomington, USA
Till Mossakowski, DFKI Lab Bremen, Germany (Co-chair)
Peter Mosses, University of Wales Swansea, UK
Fernando Orejas, Polytechnical University Catalonia, Barcelona, Spain
Prakash Panangaden, McGill University, Canada
Dirk Pattinson, University of Leicester, UK
Dusko Pavlovic, Kestrel Institute, USA
Jean-Eric Pin, CNRS-LIAFA Paris, France
John Power, University of Edinburgh, UK
Horst Reichel, Technical University of Dresden, Germany
Grigore Roşu, University of Illinois, Urbana, USA
Jan Rutten, CWI and Free University, Amsterdam, The Netherlands
Davide Sangiorgi, University of Bologna, Italy
Andrzej Tarlecki, Warsaw University, Poland
Martin Wirsing, Ludwig Maximilian University, Munich, Germany
Uwe Wolter, University of Bergen, Norway

Additional Reviewers

Fabio Alessi
Alexandru Baltag
Marek A. Bednarczyk
Mikolaj Bojanczyk
Marcello Bonsangue
Artur Boronat
Andrzej Borzyszkowski
Tomasz Borzyszkowski
Maria Grazia Buscemi
Luís Caires
Maura Cerioli
Bob Coecke
Giovanna D'Agostino
Rocco De Nicola
Fer-Jan de Vries
Pietro Di Gianantonio
Gilles Dowek
Francisco Durán
Zoltán Ésik
Fabio Gadducci
Marie-Claude Gaudel
Giorgio Ghelli
Yuri Gurevich
Ichiro Hasuo
Daniel Hausmann
Chris Heunen
Mark Hills
Adis Hodzic
Jiho Kim
Jürgen Koslowski
Clemens Kupke
Alexander Kurz
José Labra
Alberto Lluch Lafuente
Christoph Lüth
Bas Luttik
Paulo Mateus
Marino Miculan
Michael Mislove

Faron Moller
Peter Padawitz
Ricardo Peña
Carla Piazza
Andrei Popescu
Ulrike Prange
M. A. Reniers
Mehrnoosh Sadrzadeh
Ivan Scagnetto
Alan Schmitt
Lutz Schröder
Traian-Florin Serbanuta
Olha Shkaravska
Doug Smith
Paweł Sobociński
Ana Sokolova
Sam Staton
Hendrik Tews
Alwen Tiu
Emilio Tuosto
Tarmo Uustalu
Birna van Riemsdijk
Walter Vogler
Dennis Walter
Herbert Wiklicky
Marek Zawadowski

Table of Contents

Invited Talks

Regular and Algebraic Words and Ordinals 1
Stephen L. Bloom and Zoltán Ésik

Logical Semantics of Types for Concurrency 16
Luís Caires

Deriving Bisimulation Congruences with Borrowed Contexts 36
Barbara König

Symmetry and Concurrency 40
Glynn Winskel

Contributed Papers

Ready to Preorder: Get Your BCCSP Axiomatization for Free! 65
Luca Aceto, Wan Fokkink, and Anna Ingólfsdóttir

Impossibility Results for the Equational Theory of Timed CCS 80
Luca Aceto, Anna Ingólfsdóttir, and MohammadReza Mousavi

Conceptual Data Modeling with Constraints in Maude 96
Scott Alexander

Datatypes in Memory 111
David Aspinall and Piotr Ho man

Bisimilarity and Behaviour-Preserving Reconfigurations of Open Petri Nets 126
Paolo Baldan, Andrea Corradini, Hartmut Ehrig, Reiko Heckel, and Barbara König

Free Modal Algebras: A Coalgebraic Perspective 143
Nick Bezhanishvili and Alexander Kurz

Coalgebraic Epistemic Update Without Change of Model 158
Corina Cîrstea and Mehrnoosh Sadrzadeh

The Maude Formal Tool Environment 173
Manuel Clavel, Francisco Durán, Joe Hendrix, Salvador Lucas, José Meseguer, and Peter Ölveczky

Bifinite Chu Spaces 179
Manfred Droste and Guo-Qiang Zhang

Structured Co-spans: An Algebra of Interaction Protocols 194
José Luiz Fiadeiro and Vincent Schmitt

Graphical Encoding of a Spatial Logic for the π-Calculus.............. 209
Fabio Gadducci and Alberto Lluch Lafuente

Higher Dimensional Trees, Algebraically 226
Neil Ghani and Alexander Kurz

A Semantic Characterization of Unbounded-Nondeterministic Abstract State Machines 242
Andreas Glausch and Wolfgang Reisig

Parametric (Co)Iteration vs. Primitive Direcursion 257
Johan Glimming

Bisimulation for Neighbourhood Structures........................... 279
Helle Hvid Hansen, Clemens Kupke, and Eric Pacuit

Algebraic Models of Simultaneous Multithreaded and Multi-core Processors 294
Neal A. Harman

Quasitoposes, Quasiadhesive Categories and Artin Glueing 312
Peter T. Johnstone, Stephen Lack, and Paweł Sobociński

Applications of Metric Coinduction................................. 327
Dexter Kozen and Nicholas Ruozzi

The Goldblatt-Thomason Theorem for Coalgebras 342
Alexander Kurz and Jiří Rosický

Specification-Based Testing for CoCasl's Modal Specifications 356
Delphine Longuet and Marc Aiguier

CIRC: A Circular Coinductive Prover 372
Dorel Lucanu and Grigore Roşu

Observing Distributed Computation. A Dynamic-Epistemic Approach 379
Radu Mardare

Nabla Algebras and Chu Spaces 394
Alessandra Palmigiano and Yde Venema

An Institutional Version of Gödel's Completeness Theorem 409
Marius Petria

Coalgebraic Foundations of Linear Systems........................... 425
J.J.M.M. Rutten

Bootstrapping Types and Cotypes in HasCasl 447
Lutz Schröder

Author Index 463

Regular and Algebraic Words and Ordinals

S.L. Bloom[1] and Z. Ésik[2,⋆]

[1] Dept. of Computer Science
Stevens Institute of Technology
Hoboken, NJ. USA
[2] Dept. of Computer Science
University of Szeged
Szeged, Hungary
GRLMC
Rovira i Virgili University
Tarragona, Spain

Abstract. We solve fixed point equations over finite and infinite words by initiality. By considering equations without and with parameters, two families of words arise, the regular and the algebraic words. Regular words were introduced by Courcelle in the late 1970's. We provide a summary of results on regular words, some of which have been obtained very recently, and include some new results for algebraic words.

Some notation. For an integer n, we denote the set $\{1, \ldots, n\}$ by $[n]$. In any category, we will write composition of morphisms $f : a \to b$ and $g : b \to c$ as $g \circ f$. Horizontal composition of natural transformations is also denoted by $\circ$. The identity natural transformation of a functor f is denoted also by f.

1 Introduction

There are two common approaches in computer science to solve fixed point equations in algebraic or categorical structures. One uses structures enriched with a complete partial order or a complete metric, or some related structure, that ensures that fixed point equations possess an extremal solution, cf. e.g. [Scott76, Ni75, ADJ77]. The other approach is to impose axioms on the existence and properties of fixed points. Elgot considered what we call here regular recursion schemes, or systems of fixed point equations, and required that the nontrivial ones possess a unique solution. In the framework of Lawvere algebraic theories, this led to the notion of *iterative theories*, cf. [Elg75]. In contrast, the definition of iteration theories [BEW1, BEW2, Es80] imposes equational axioms on the fixed point operation. It has been shown that the theories of Elgot as well as the ordered, metric and categorical models are all examples of iteration theories, cf. [BE93, BE97, EsLab98].

⋆ Supported in part by the AUTOMATHA ESF project.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 1–15, 2007.
© Springer-Verlag Berlin Heidelberg 2007

In this paper, we will be concerned with solving fixed point equations in certain algebras enriched with a *categorical* structure. The continuous categorical algebras defined below extend the notion of continuous (ordered) algebras [ADJ77, Bl76].

In this paper, two classes of these algebras play a central role: trees and words.

2 Continuous Categorical Algebras Defined

Suppose that Σ is a ranked set. A **categorical Σ-algebra**, or *cΣa*, is a *small* category A equipped with a functor $\sigma^A : A^n \to A$ for each $\sigma \in \Sigma_n$, $n \geq 0$. A **morphism** between cΣa's A and B is a functor $h : A \to B$ such that for each $\sigma \in \Sigma_n$ the functors $h \circ \sigma^A$ and $\sigma^B \circ h^n$ are naturally isomorphic. Here, h^n denotes the functor $A^n \to B^n$ sending each object and morphism $(x_1, \ldots, x_n)$ of A^n to $(h(x_1), \ldots, h(x_n))$. A morphism h is *strict* if the functors $h \circ \sigma^A$ and $\sigma^B \circ h^n$ are the same, for all $\sigma \in \Sigma$.

Let A be a cΣa. We call A *continuous*, or a *ccΣa*, if A has an initial object (denoted $\perp^A$) and colimits of all ω-diagrams $(f_k : a_k \to a_{k+1})_{k \geq 0}$. Moreover, each functor σ^A is continuous, i.e., preserves colimits of ω-diagrams. A **morphism** between ccΣa's is a cΣa morphism which preserves the initial object and colimits of all ω-diagrams.

The concept of a Σ*-term* over a set $X = \{x_1, x_2, \ldots\}$ is defined as usual, (see [Gr79], for example). For each n, we let $T_\Sigma(X_n)$ denote the set of all Σ-terms in the variables $X_n = \{x_1, \ldots, x_n\}$. When A is a cΣa and $t \in T_\Sigma(X_n)$, t induces a functor $t^A : A^n \to A$. The definition is by induction on the structure of t. When $t = x_i$, for some variable $x_i \in X_n$, then t^A is the ith projection functor $A^n \to A$. Suppose that $t = \sigma(t_1, \ldots, t_m)$, where $\sigma \in \Sigma_m$ and $t_1, \ldots, t_m \in T_\Sigma(X_n)$. Then $t^A = \sigma^A \circ \langle t_1^A, \ldots, t_m^A \rangle$, where $\langle t_1^A, \ldots, t_m^A \rangle : A^n \to A^m$ is the *target tupling* of the functors t_i^A, $i \in [m]$. When A is continuous, so is t^A. For $n \geq 0$, we let

$$[A^n \to A]$$

denote the category whose objects are all continuous functors $A^n \to A$; morphisms are natural transformations.

2.1 Some Facts

If A, B are ccΣa's, so is $A \times B$, where the Σ-functors are defined pointwise, and so is $[A \to B]$, the collection of continuous functors $A \to B$, where the Σ-functors are defined in the expected way. Say for example that $\sigma \in \Sigma_2$, and $f, g : A \to B$ are continuous functors: then so is

$$\sigma^{[A \to B]}(f, g) : A \to B,$$

defined on objects and morphisms x in A by:

$$\sigma^{[A \to B]}(f, g)(x) := \sigma_B(f(x), g(x)),$$

i.e., $\sigma^{[A \to B]} = \sigma^A \circ \langle f, g \rangle$. Thus, in particular, when A is a ccΣa, so is $[A^m \to A]$, and any finite product of these categories.

If A is a ccΣa, and $F : A \to A$ is a continuous endofunctor, then it is well-known that F has an initial fixed point $F^\dagger$, i.e., there is an isomorphism

$$\iota : F(F^\dagger) \to F^\dagger,$$

and if $\alpha : F(a) \to a$ is any morphism in A, then there is a unique morphism $h : F^\dagger \to a$ such that

$$F(F^\dagger) \stackrel{F(h)}{\to} F(a) \stackrel{\alpha}{\to} a = F(F^\dagger) \stackrel{\iota}{\to} F^\dagger \stackrel{h}{\to} a.$$

$F^\dagger$ is "the" colimit of the usual diagram

$$\perp^A \stackrel{\alpha_0}{\to} F(\perp^A) \stackrel{\alpha_1}{\to} \dots$$

where α_0 the unique map, and, for $n \geq 0$, $\alpha_{n+1} = F(\alpha_n)$.

2.2 Examples

We give two examples of ccΣa's. In both examples, we fix a finite alphabet A and let $\Sigma = \Sigma(A)$ be the ranked alphabet with $\Sigma_2 = \{\cdot\}$ and $\Sigma_0 = A$. When $n \neq 0, 2$, $\Sigma_n = \emptyset$.

Example I: Words

A word over A is a *countable* linear order $w = (W, <_w)$ equipped with a labeling function $\lambda_w : W \to A$. (In order to have only a small set of words we require that the underlying set W of a word w is a subset of a fixed set.) A morphism between words $v = (V, <_v, \lambda_v)$ and $w = (W, <_w, \lambda_w)$ is a function $h : V \to W$ which preserves the order (and is thus injective) and the labeling. A word w is finite if the underlying set is finite. The category W_A of words over A has as initial object the *empty word* ϵ, where $L_\epsilon = \emptyset$. Moreover, W_A has colimits of all ω-diagrams. We turn W_A into a ccΣa by interpreting the binary symbol $\cdot$ as the concatenation functor $W_A^2 \to W_A$, and each letter a as a singleton word labeled a.

When A is an alphabet equipped with a linear order $<_A$, there are three important orderings on the finite words on A. The **prefix order** is denoted $<_p$; $u <_p v$ holds when there is a nonempty word w with $v = uw$; the *strict order*, denoted $u < v$, holds when

$$u = u_1 a u_2$$
$$v = u_1 b v_2$$

and $a <_A b$ in the alphabet A. Last, the **lexicographic order** is defined as follows. If $u \neq v$,

$$u <_\ell v \iff u <_p v \text{ or } u < v.$$

While both $<_p$ and $<$ are partial orderings on A^*, $<_\ell$ is a linear order on all words. Proposition 2.1 below recalls a universal property of $<_\ell$ on $\{0,1\}^*$.

Suppose that A and B are alphabets such that B is linearly ordered. Suppose that for each $a \in A$ we are given a language $L_a \subseteq B^*$ such that $L_a \cap L_{a'} = \emptyset$, for all $a, a' \in A$ with $a \neq a'$. Then the word $w(L_a : a \in A)$ over A is defined as the word

$$w(L_a : a \in A) = (\bigcup_{a \in A} L_a, <_\ell, \lambda)$$

where $\lambda(u) = a$ if $u \in L_a$, $a \in A$.

Given a collection of languages $\mathcal{L}$, such as the regular languages, we say that a word w **is determined by the languages in** $\mathcal{L}$ if there exists an alphabet B and (pairwise disjoint) languages $L_a \subseteq B^*$ in $\mathcal{L}$, $a \in A$ such that w is isomorphic to $w(L_a : a \in A)$. We let $W_{\mathcal{L}}$ denote the class of all words determined by the languages in $\mathcal{L}$. We will use this definition for the cases that $\mathcal{L} = \mathrm{Reg}$, the regular languages, $\mathcal{L} = \mathrm{DCFL}$, the deterministic context-free languages, $\mathcal{L} = \mathrm{CFL}$, the context-free languages $\mathcal{L} = \mathrm{Rec}$, the recursive languages.

We call a word $w \in W_A$ *recursive* if it is finite or there is a recursive linear order $<_r$ on $\mathbb{N}$, the set of nonnegative integers such that w is isomorphic to a word $(\mathbb{N}, <_r, \lambda)$ such that each set $\lambda^{-1}(a)$ is recursive, i.e., λ is a computable function $\mathbb{N} \to A$.

Proposition 2.1. *Each word over an alphabet A is isomorphic to a word $w(L_a : a \in A)$ where each L_a is a language over the alphabet $\{0,1\}$. Moreover, a word over A is recursive i   it is determined by a family of recursive languages, i.e., when it is isomorphic to a word $w(L_a : a \in A)$ where each $L_a \subseteq \{0,1\}^*$ is recursive. Thus W_{Rec} is the set of recursive words.*

In fact, we have

Proposition 2.2. *Any word $w(L_a : a \in A)$, where each L_a is a language over $\{0,1\}$ is isomorphic to a word $w(K_a : a \in A)$ where $K = \bigcup_{a \in A} K_a$ is a **prefix code** over $\{0,1\}$, i.e., if $u, uv \in K_a$, then v is the empty word. When each L_a is regular or (deterministic) context-free, or recursive, then so is each K_a.*

Example II: Trees

The second class of ccΣa algebras involves trees.

The ordered algebra

$$T_\Sigma^\omega$$

consists of all (finite and infinite) $\Sigma(A)$-trees. A tree t in T_Σ^ω is a partial function $t : \{0,1\}^* \to \Sigma(A)$ whose domain is prefix closed and such that if $t(w) \in A$, then u is a maximal element of the domain of t with respect to the prefix order, i.e., u is a leaf. A tree t is **finite** if its domain is finite and **complete** if whenever $t(u) = \cdot$ then the words $u0, u1$ are in $\mathrm{dom}(t)$.

Trees are equipped with the following partial order $\sqsubset$: Given $t, t' \in T_\Sigma^\omega$ such that $t \neq t'$, we define $t \sqsubset t'$ iff for all words u, if $t(u)$ is defined, then $t(u) = t'(u)$. We consider T_Σ^ω as a category in the usual way: there is a morphism $t \to t'$ when $t \sqsubseteq t'$. Since T_Σ^ω has as least element the totally undefined tree $\perp$, the category T_Σ^ω has an initial object, and sups of all ω-chains, i.e., colimits of all ω-diagrams.

We turn T_Σ^ω into a cc$\Sigma(A)$a. The value of the functor $\cdot^{T_\Sigma^\omega}(t_0, t_1)$ on trees t_0, t_1 is the the tree t with

$$\begin{aligned} t(\epsilon) &= \cdot \\ t(iu) &= t_i(u), \end{aligned}$$

for $i = 0, 1$ and $u \in \{0,1\}^*$. Since the operation $\cdot^{T_\Sigma^\omega}$ is monotonic, it determines a continuous functor $T_\Sigma^\omega \times T_\Sigma^\omega \to T_\Sigma^\omega$. We interpret each letter $a \in A$ as the trivial tree that maps the empty word ϵ to a and is undefined on nonempty words.

The **yield of a tree** $t \in T_\Sigma^\omega$ is defined as the word

$$\text{yield}(t) = (W, <, \lambda_t),$$

where W is the set $\{u \in \{0,1\}^* : t(u) \in A\}$, linearly ordered by the strict order $<$ on finite words. The labeling is defined by $\lambda_t(u) = t(u)$ for all $u \in W$. In particular, yield$(\perp) = \epsilon$. When $t \sqsubset t'$ in the partial order of trees, then we define yield$(t \sqsubset t')$ as the embedding of $W = \text{yield}(t)$ into $W' = \text{yield}(t')$. (Note that $W \subseteq W'$.)

Proposition 2.3. *The functor* yield $: T_\Sigma^\omega \to W_A$ *is a ccΣa-morphism.*

Remark 2.4. *It is known that T_Σ^ω is the initial in the category of ccΣa's and continuous morphisms. Thus* yield *is, up to isomorphism, the unique ccΣa morphism $T_\Sigma^\omega \to W_A$.*

3 Regularity and Algebraicity

In this section we will consider regular and algebraic objects (or elements) in a ccΣa A as initial solutions of finite systems of fixed point equations. The main fact established in this section is a Mezei-Wright theorem: *Any morphic image of a regular or algebraic element is also regular, or algebraic.*

For the origins of the term "Mezei-Wright theorem" see [MeWr67] and [ADJ77].

Let $F = \{F_1, \ldots, F_n\}$ be a ranked alphabet disjoint from Σ, and suppose that the rank of F_i is k_i, for $i \in [n]$. Define

$$A^{\rho(F)} = ([A^{k_1} \to A] \times \cdots \times [A^{k_n} \to A])$$

Then $A^{\rho(F)}$ is a ccΣa, as noted in Section 2.1 above.

Given any ccΣa A, each term $t \in T_{\Sigma \cup F}(X_m)$ induces a continuous functor

$$t^A : A^{\rho(F)} \to [A^m \to A],$$

as follows. When $f_i : A^{k_i} \to A$, $i \in [n]$ is continuous, we may define a cc($\Sigma \cup F$)a structure on A by interpreting each F_i as the functor f_i. We define $t^A(f_1, \ldots, f_n)$ as the functor induced by t in this cc($\Sigma \cup F$)a. If $g_i : A^{k_i} \to A$ is also continuous, and

$$\alpha_i : f_i \to g_i$$

is a natural transformation, for each $i \in [n]$, then we define

$$t^A(\alpha_1, \ldots, \alpha_n)$$

to be the natural transformation

$$t^A(f_1, \ldots, f_n) \to t^A(g_1, \ldots, g_n)$$

defined inductively as follows.

- When t is a variable x_j, where $j \in [m]$, then $t^A(\alpha_1, \ldots, \alpha_n)$ is the identity natural transformation from the jth projection function $A^m \to A$ to itself.
- If t is of the form $\sigma(t_1, \ldots, t_p)$, then $t^A(\alpha_1, \ldots, \alpha_n)$ is $\sigma^A \circ \langle u_1, \ldots, u_p \rangle$, where $u_j = t_j^A(\alpha_1, \ldots, \alpha_n)$ for each j.
- Finally, when $t = F_i(t_1, \ldots, t_{k_i})$, $i \in [n]$, then $t^A(\alpha_1, \ldots, \alpha_n)$ is $\alpha_i \circ \langle u_1, \ldots, u_m \rangle$, where each u_j is $t_j^A(\alpha_1, \ldots, \alpha_n)$.

Now we define recursion schemes.

A **recursion scheme** over Σ is a sequence E of equations

$$\begin{aligned} F_1(x_1, \ldots, x_{k_i}) &= t_1 \\ &\vdots \\ F_n(x_1, \ldots, x_{k_n}) &= t_n \end{aligned} \tag{1}$$

where t_i is a term in $T_{\Sigma \cup F}(X_{k_i})$, for $i \in [n]$.

The recursion scheme E in (1) determines a continuous functor

$$E_A : A^{\rho(F)} \to A^{\rho(F)},$$

which therefore has an initial fixed point. When $k_1 = 0$, the first component of this initial fixed point is an object in A.

Definition 3.1. *Let A be a ccΣa. An object a in A is* ***algebraic*** *if there is a recursion scheme E as above with $k_1 = 0$ such that a is isomorphic to the first component of the initial fixed point of E_A.*

A recursion scheme E is **regular** if each F_i has rank 0, $1 \leq i \leq n$.

Definition 3.2. *An object a in A is* ***regular*** *if there is a regular recursion scheme E such that a is isomorphic to the first component of the initial fixed point of E_A.*

Thus, any regular object is algebraic. In particular, when A is the algebra W_A, a regular (algebraic, resp.) object is called a regular (algebraic, resp.) word. And when A is T^ω_Σ, a regular object is called a regular tree (see e.g. [BE93]), and an algebraic object an algebraic tree (cf. [Cour78b]).

Theorem 3.1 ("Mezei-Wright"). *Suppose that E is a recursion scheme as above such that k_1 is 0. Let A and B be ccΣa's, and $h : A \to B$ a ccΣa-morphism. Then the first component of the initial fixed point of E_B is isomorphic to the image under h of the first component of the initial fixed point of E_A.*

Corollary 3.3. *Let A and B be ccΣa's and $h : A \to B$ a ccΣa-morphism. Then an object $b \in B$ is regular or algebraic i there is some $a \in A$ which is regular or algebraic such that $h(a)$ is isomorphic to b.*

4 Application to Words

In this section, we apply the Mezei-Wright Theorem to obtain a characterization of algebraic and regular words. For regular words, the following fact was proved by Courcelle [Cour78b].

Corollary 4.1. *A word w in W_A is regular (resp. algebraic) i there is a regular (resp. algebraic) tree in T^ω_Σ such that w is isomorphic to the yield of t.*

Proof. This follows from the Mezei-Wright Theorem and the fact that yield is a ccΣa-morphism. □

Following Courcelle [Cour78b], for each $a \in A$ we define the *branch language* $L_a(t)$ of a tree $t \in T_\Sigma$ as the set

$$L_a(t) = \{u : t(u) = a\}.$$

We let $K(t) = \bigcup_{a \in A} L_a$. Note that the language $K(t)$ is a prefix code.

If Δ is any ranked alphabet, we call a tree t in T^ω_Δ *locally finite* if for each $u \in \mathrm{dom}(t)$ there is some v with $t(uv) \in \Delta_0$.

The following fact is a special case of results proved in [Cour78b].

Theorem 4.1. *Let L_a, $a \in A$, be a family of pairwise disjoint languages over $\{0, 1\}$. There is a locally finite complete algebraic tree $t \in T^\omega_{\Sigma(A)}$ with $L_a(t) = L_a$ for all $a \in A$ i each L_a is a dcfl and $L = \bigcup_{a \in A} L_a$ is a complete prefix code.*

Corollary 4.2. *A complete locally finite tree $t \in T^\omega_{\Sigma(A)}$ is algebraic i each $L_a(t)$ is a dcfl.*

We may use the above result to prove:

Theorem 4.2. *Let L_a, $a \in A$ be a family of pairwise disjoint languages over $\{0, 1\}$. There is an algebraic tree $t \in T^\omega_{\Sigma(A)}$ with $L_a(t) = L_a$ for all $a \in A$ i each L_a is a dcfl and $L = \bigcup_{a \in A} L_a$ is a prefix code.*

Further results follow.

Corollary 4.3. *A tree* $t \in T^{\omega}_{\Sigma(A)}$ *is algebraic i each* $L_a(t)$ *is a dcfl.*

From this we have:

Corollary 4.4. *A word in* W_A *is algebraic i it is in* W_{DCFL}*.*

Corollary 4.5 ([BE03]). *A word in* W_A *is regular i it is in* W_{Reg}*.*

5 More About Regular Words

Several results on regular words have been obtained by Courcelle [Cour78a, Cour78b], Heilbrunner [Heil80], Thomas [Th86], and the authors [BE03, BE03a, BE05]. Heilbrunner showed that all nonempty regular words on the set A can be generated from single letters by means of the "regular operations", namely concatenation, omega and omega-op power, together with (infinitely many) "shuffle" operations. Terms formed from letters in A and these operations are called "regular terms on A". Heilbrunner gave an algorithm which, given a finite system of fixed point equations of the form (1), in which the rank of F_i is zero, for each $i \in [n]$, produces a regular term denoting the first component of the initial solution (if the first component is not the empty word). Thomas gave an algorithm to determine when two terms denote isomorphic words. His algorithm is based on Rabin's theorem on automata for infinite trees. There is an extensive literature on automata (and logics) on words, here we mention only [Bu65] and the more recent [BeCa06, BruyCar].

Heilbrunner discussed several identities involving the terms with both Courcelle's operations, as well as the shuffle operations, but did not obtain a completeness result. In [BE05], we gave a set of axioms for these operations and proved them complete. This result implies that

- for any alphabet A, the algebra of regular words on an alphabet A is freely generated by A in the variety defined by these equations;
- the equational theory of this variety is decidable in polynomial time, and is not finitely based.

We give some more details below. For a fixed finite alphabet A, the regular operations are concatenation, the omega, omega-op operations, and shuffle operations, written

$$u \cdot v,\ u^{\omega},\ u^{\omega^{op}},\ [\![u_1, \ldots, u_n]\!]^{\eta}.$$

We will define each of these operations using generalized sums.

Definition 5.1 (generalized sum). *Suppose that* $(L, <)$ *is a linear order, and for each* $x \in L$*, let* $(K_x, <_x)$ *be a linear order. The ordering*

$$\sum_{x \in L} K_x$$

obtained by substitution of K_x for $x \in L$, is defined as follows: the underlying set of $\sum_{x\in L} K_x$ is the set of pairs (k, x) with $x \in L$ and $k \in K_x$ ordered by:

$$(k, x) < (k', x') \iff x < x' \text{ or } (x = x' \text{ and } k <_x k').$$

For any word u, $\mathbf{alph}(u)$ is the set of letters occurring in u.

Definition 5.2. *Let $u = (L_u, <_u, \lambda_u)$ be a word with $\mathbf{alph}(u) \subseteq \{a_1, \ldots, a_n\}$, and let $v_{a_i} = (L_{v_{a_i}}, <_{v_{a_i}}, \lambda_{v_{a_i}})$ be a word on the set B, for each $i \in [n]$. The sets A, B need not be the same. We define $w = u(a_1/v_{a_1}, \ldots, a_n/v_{a_n})$, the word obtained* ***by substituting*** *v_{a_i}* ***for each occurrence of*** *a_i* ***in*** *u, as follows. The underlying order of w is the linear order $\sum_{x\in L_u} L_{\lambda_u(x)}$, defined just above, labeled as follows:*

$$\lambda_w(k, x) := \lambda_{v_{\lambda_u(x)}}(k), \quad x \in L_u,\ k \in L_{v_{\lambda_u(x)}}.$$

We note that the collection of regular words is closed under substitution.

Proposition 5.3. *Suppose that $u \in W_A$ is regular, and for each letter $a_i \in A$, v_i is a regular word. Then $w = u(a_1/v_1, \ldots, a_n/v_n)$ is also regular.*

Indeed, we obtain a regular recursion scheme for w by adjoining to the scheme E for u, the schemes for each v_i, and replacing the letters a_i in E by the initial variable in the scheme for v_i. □

Define the following words on the countable set $a_1, a_2, \ldots,$.

- $c := a_1a_2$,
  the word $([2], \leq, u)$ with $u(i) = a_i$.
- $p_\omega := a_1a_1 \ldots$,
  the word whose underlying linear order is ω, each point of which is labeled a_1.
- $r_{\omega^{op}} := \ldots a_1a_1$,
  the word whose underlying linear order is ω^{op}, each point of which is labeled a_1.
- For $1 \leq n < \omega$, ρ_n is the word whose underlying linear order is $\mathbb{Q}$, the rational numbers, every point labeled by some a_i, $i \in [n]$, and between any two points $q < q'$ in $\mathbb{Q}$, for each $j \in [n]$ there is a point labeled a_j. There is a unique such word, up to isomorphism (see [Ro82], pp 116.)

Definition 5.4 (regular operations). *For any words $u, v, u_1, \ldots, u_n$ on A:*

$$\begin{aligned} u \cdot v &:= c(a_1/u, a_2/v) \\ u^\omega &:= p_\omega(a_1/u) \\ u^{\omega^{op}} &:= r_{\omega^{op}}(a_1/u) \\ [u_1, \ldots, u_n]^\eta &:= \rho_n(a_1/u_1, \ldots, a_n/u_n). \end{aligned}$$

Note that there is one shuffle operation for each positive integer n.

Theorem 5.1 ([Heil80]). *A nonempty word in W_A is regular iff it is in the least collection of words containing the single letter words which is closed under the regular operations.*

Recall that a countable linear order $(L, \leq)$ is called **scattered** if there is no injective order-preserving function $\mathbb{Q} \to L$, where $\mathbb{Q}$ is ordered as usual. A word $w = (W, <_w, \lambda_w)$ is scattered if $(W, <_w)$ is scattered.

Corollary 5.5. *A nonempty regular word u has a scattered underlying linear order iff u is in the least class of words containing the single letter words which is closed under concatenation, the omega and omega-op operations.*

Indeed, any word of the form $[\![u_1, \ldots, u_n]\!]^\eta$ is not scattered.

Recall that a regular language is **monotone** (or $\mathcal{R}$-trivial, cf. [Pi86]) if it is determined by a DFA with the property that its states can be linearly ordered so that for any state q and letter a, if $\delta(q, a) = q'$, then $q \leq q'$.

The following facts about scattered regular words are from [BE03].

Theorem 5.2. *For a word $w = (L_w, \leq_w, \lambda_w)$ on the alphabet A, the following are equivalent.*

1. *w belongs to the least class of words containing the single letters $a \in A$, closed under product and both ω-operations.*
2. *w is regular and L_w is a scattered (regular) linear order.*
3. *w is isomorphic to a regular word u, where L_u is a monotone, (complete) prefix code.*
4. *w is isomorphic to a regular word u, where L_u is scattered and a regular (complete) prefix code.*
5. *w is isomorphic to a word $w(L_a : a \in A))$, where the sets L_a are regular, pairwise disjoint, and $\bigcup_{a \in A} L_a$ is a monotone (complete) prefix code.*

□

We list our axioms for the regular operations.

Definition 5.6 (scattered axioms)

$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \tag{2}$$
$$(x \cdot y)^\omega = x \cdot (y \cdot x)^\omega \tag{3}$$
$$(x \cdot y)^{\omega^{op}} = (y \cdot x)^{\omega^{op}} \cdot y \tag{4}$$
$$(x^n)^\omega = x^\omega, \quad n \geq 2 \tag{5}$$
$$(x^n)^{\omega^{op}} = x^{\omega^{op}}, \quad n \geq 2. \tag{6}$$

The scattered axioms were proved complete for the operations of concatenation, omega and omega-op powers in [BE03a].

Definition 5.7 (logical axioms)

$$[\![x_{f(1)}, \ldots, x_{f(n)}]\!]^\eta = [\![x_1, \ldots, x_p]\!]^\eta,$$

where $f : [n] \to [p]$ is any set-theoretic surjection.

The logical axioms say that the shuffle operation $[\![u_1, \ldots, u_n]\!]^\eta$ is a function whose value is determined by the set $\{u_1, \ldots, u_n\}$, not the sequence $(u_1, \ldots, u_n)$; for example, using these axioms one may derive the facts that $[\![a, a, b]\!]^\eta = [\![b, b, a]\!]^\eta = [\![a, b]\!]^\eta = [\![b, a]\!]^\eta$.

Definition 5.8 (concatenation/shuffle axioms)

$$[\![x_1, \ldots, x_p]\!]^\eta \cdot [\![x_1, \ldots, x_p]\!]^\eta = [\![x_1, \ldots, x_p]\!]^\eta \tag{7}$$

$$[\![x_1, \ldots, x_p]\!]^\eta \cdot x_i \cdot [\![x_1, \ldots, x_p]\!]^\eta = [\![x_1, \ldots, x_p]\!]^\eta, \quad i \in [p]. \tag{8}$$

Definition 5.9 (omega/shuffle axioms)

$$([\![x_1, \ldots, x_p]\!]^\eta)^\omega = [\![x_1, \ldots, x_p]\!]^\eta \tag{9}$$

$$([\![x_1, \ldots, x_p]\!]^\eta \cdot x_i)^\omega = [\![x_1, \ldots, x_p]\!]^\eta, \quad i \in [p]. \tag{10}$$

Definition 5.10 (omega-op/shuffle axioms)

$$([\![x_1, \ldots, x_p]\!]^\eta)^{\omega^{op}} = [\![x_1, \ldots, x_p]\!]^\eta \tag{11}$$

$$(x_i \cdot [\![x_1, \ldots, x_p]\!]^\eta)^{\omega^{op}} = [\![x_1, \ldots, x_p]\!]^\eta, \quad i \in [p]. \tag{12}$$

Definition 5.11 (shuffle/shuffle axioms)

$$[\![u_1, \ldots, u_k, v_1, \ldots, v_s]\!]^\eta = [\![x_1, \ldots, x_p]\!]^\eta, \quad k \geq 0, \ s > 0, \tag{13}$$

where in (13), the terms u_i are letters in the set $\{x_1, \ldots, x_p\}$, and each term v_j is one of the following:

$$[\![x_1, \ldots, x_p]\!]^\eta, \ x_i[\![x_1, \ldots, x_p]\!]^\eta, \ [\![x_1, \ldots, x_p]\!]^\eta x_j, \ \text{or} \ \ x_i[\![x_1, \ldots, x_p]\!]^\eta x_j.$$

Note that a special case of the shuffle/shuffle axioms is the identity

$$([\![x_1, \ldots, x_p]\!]^\eta)^\eta = [\![x_1, \ldots, x_p]\!]^\eta.$$

Theorem 5.3 ([BE05]). *Two terms s, t denote isomorphic regular words i the identity $s = t$ is provable from the above axioms. No finite set of the axioms is complete. There is is a polynomial time algorithm to decide when $s = t$ is provable.*

In fact, it was proved in [BE05] that there is an $O(n^5)$ algorithm to decide if two terms denote isomorphic words, where n is the total length of the terms s, t.

6 Ordinal Words

Let $w = (L_w, <_\ell, \lambda_w)$ be a word over A such that L_w is a subset of $\{0,1\}^*$. We call w an *ordinal word* if L_w is well-ordered by $<_\ell$. A regular, context-free, or deterministic context-free ordinal word is an ordinal word which is a regular,

context-free, or deterministic context-free word, respectively. In particular, when A is a singleton, we may forget the labeling and call a regular (context-free, deterministic context-free) ordinal word a regular (context-free, deterministic context-free) ordinal.

An ordinal α is regular, (resp., context-free) if there is some regular (resp. context-free) ordinal word w such that $\alpha = \mathbf{o}(L_w, <_\ell)$, the order-type of the linear order $(L_w, <_\ell)$.

It is clear that each regular ordinal is deterministic context-free and each deterministic context-free ordinal is a context-free ordinal.

What are the context-free ordinals? Or the deterministic context-free ordinals? What is an order-theoretic characterization of the (deterministic) context-free ordinals? In this section we give a partial answer to these questions.

Proposition 6.1 ([BC01]). *An ordinal α is regular i $\alpha < \omega^\omega$.*

Conjecture. An ordinal α is context-free iff $\alpha < \omega^{\omega^\omega}$.

While we do not have a proof of the conjecture, we seem to be close.

One direction is known.

Theorem 6.1. *Any ordinal $< \omega^{\omega^\omega}$ is deterministic context-free.*

Proof. Below, for each $n \geq 0$, we will specify a deterministic, context-free grammar G_n such that

$$\mathbf{o}(\mathcal{L}(G_n), <_\ell) = \omega^{\omega^n}.$$

This will prove the theorem. Indeed, if α is an ordinal less than ω^{ω^ω}, then $\alpha < \omega^{\omega^n}$, for some n. But, for any word $u \in \{0,1\}^*$, the set of words

$$pr(u) = \{v \in \{0,1\}^* : v <_\ell u\}$$

is regular, it follows that $\alpha < \omega^{\omega^n}$ is deterministic, context-free, since $\alpha = \mathbf{o}(\mathcal{L}(G_n) \cap pr(u))$, for some word $u \in \mathcal{L}(G_n)$. □

The proof of that the grammars are correct makes use of several lemmas concerning prefix codes $X \subseteq \{0,1\}^*$ such that $\mathbf{o}(X, <_\ell)$ is an ordinal.

Recall that if $(L_i, \leq_i)$ are well ordered sets, with order types α_i, $i = 1,2$, then the ordinal $\alpha_1 \times \alpha_2$ is the order type of $L_1 \times L_2$, ordered by "last difference"

$$(x_1, x_2) < (y_1, y_2) \iff x_2 < y_2, \text{ or } (x_2 = y_2 \;\&\; x_1 < y_1).$$

Lemma 6.2. *Suppose that $X_i \subseteq \{0,1\}^*$ is a prefix code, for $i = 1,2$. Then $X_1 \cdot X_2 = \{uv : u \in X_1, v \in X_2\}$ is also a prefix code and*

$$\mathbf{o}(X_1 \cdot X_2, <_\ell) = \mathbf{o}(X_2) \times \mathbf{o}(X_1). \qquad \square$$

Corollary 6.3. *If α, β are (deterministic) context-free ordinals, so is $\alpha \times \beta$.*

Lemma 6.4. *Suppose that* $A_n \subseteq \{0,1\}^*$ *and* $\alpha_n = \mathbf{o}(A_n, <_\ell)$*, for all* $n \geq 1$*. Then*

$$\sum_{n<\omega} \alpha_n = \mathbf{o}(\bigcup_n 1^n 0 A_n, <_\ell). \qquad \square$$

Corollary 6.5. *Suppose that* $A \subseteq \{0,1\}^*$ *is a prefix code and* $\alpha = \mathbf{o}(A, <_\ell)$*. Then*

$$\alpha^\omega = \mathbf{o}(\bigcup_n 1^n 0 A^n, <_\ell),$$

so that if α *is (deterministic) context-free, so is* α^ω*.*

Now, by induction on n, we define a context-free grammar G_n.

1. The grammar G_0 has the following productions.

$$S_0 \to 0,\ 1S_0$$

   $\mathcal{L}(S_0) = 1^*0$, so that $\mathbf{o}(\mathcal{L}(G_0), <_\ell) = \omega$.
2. The grammar G_1 has start symbol S_1, the rules of G_0 and the rules

$$S_1 \to 0,\ 1S_1S_0. \tag{14}$$

   $\mathcal{L}(G_1) = \bigcup_n 1^n 0 \mathcal{L}(G_0)^n$, so that $\mathbf{o}(G_1) = \omega^\omega$, by Corollary 6.5.
3. The grammar G_{n+1} has all rules of G_n, the start symbol S_{n+1}, and in addition, the rules

$$S_{n+1} \to 0,\ 1S_{n+1}S_n.$$

   Then $\mathcal{L}(G_{n+1}) = \bigcup_k 1^k 0 \mathcal{L}(G_n)^k$, so that, by induction and Corollary 6.5,

$$\begin{aligned} \mathbf{o}(L(G_{n+1}) &= (\omega^{\omega^n})^\omega \\ &= \omega^{\omega^{n+1}}. \end{aligned}$$

Each of the grammars G_n above is an example of a "prefix grammar". In addition, of course, $\mathcal{L}(G_n)$ is well-ordered by $<_\ell$.

Definition 6.6. *A context-free grammar* G *is a* ***prefix grammar*** *if, for each nonterminal* X*, the set of terminal words derivable from* X *is a prefix code. An* ***ordinal grammar*** *is a prefix grammar* G *such that* $\mathcal{L}(G)$ *is well-ordered by* $<_\ell$*.*

We have part of a converse to Theorem 6.1.

Theorem 6.2. *If* G *is an ordinal grammar, then*

$$\mathbf{o}(\mathcal{L}(G), <_\ell) < \omega^{\omega^\omega}.$$

Thus, if either of the following statements is true, the conjecture will follow.

- If G is a context-free grammar such that $\mathcal{L}(G)$ is a prefix language, then there is a prefix grammar G' such that $\mathbf{o}(\mathcal{L}(G), <_\ell) = \mathbf{o}(\mathcal{L}(G'), <_\ell)$.
- If G is a context-free grammar such that $\mathcal{L}(G)$ is well-ordered by $<_\ell$, then there is an ordinal grammar G' such that $\mathbf{o}(\mathcal{L}(G), <_\ell) = \mathbf{o}(\mathcal{L}(G'), <_\ell)$.

7 Summary

We have presented a new Mezei-Wright theorem on ccΣa's, and applied it to the case of trees and words. We have discussed some known and new results on regular words, and introduced the problem of characterizing the algebraic ordinals. Last, we stated a characterization of those ordinals having an ordinal grammar. Detailed proofs will appear elsewhere.

References

[ADJ77] Goguen, J., Thatcher, J., Wagner, E., Wright, J.: Initial algebra semantics and continuous algebras. J. ACM 24, 68–95 (1977)

[BeCa06] Bès, A., Carton, O.: A Kleene theorem for languages of words indexed by linear orderings. Int. J. Foundations Computer Science 17, 519–542 (2006)

[Bl76] Bloom, S.L.: Varieties of ordered algebras. J. Computers and Sys. Sci. 45, 200–212 (1976)

[BC01] Bloom, S.L., Choffrut, C.: Long words: the theory of concatenation and ω-power. Theoretical Computer Science 259, 533–548 (2001)

[BEW1] Bloom, S.L., Elgot, C.C., Wright, J.B.: Solutions of the iteration equation and extensions of the scalar iteration operation. Siam J. Computing 9, 26–45 (1980)

[BEW2] Bloom, S.L., Elgot, C.C., Wright, J.B.: Vector iteration in pointed iterative theories. Siam J. Computing 9, 525–540 (1980)

[BE93] Bloom, S.L., Ésik, Z.: Iteration Theories. Springer, Heidelberg (1993)

[BE97] Bloom, S.L., Ésik, Z.: The equational logic of fixed points. Theoretical Computer Science 179, 1–2 (1997)

[BE03] Bloom, S.L., Ésik, Z.: Deciding whether the frontier of a regular tree is scattered. Fundamenta Informatica 55, 1–21 (2003)

[BE03a] Bloom, S.L., Ésik, Z.: Axiomatizing omega and omega-op powers on words. Theoretical Informatics and Applications 38, 3–17 (2004)

[BE05] Bloom, S.L., Ésik, Z.: The equational theory of regular words. Information and Computation 197(1-2), 55–89 (2005)

[BruyCar] Bruyère, V., Carton, O.: Automata on linear orderings. J. Computer System Sciences 73, 1–24 (2007)

[Bu65] Bčhi, J.R.: Transfinite automata recursions and weak second order theory of ordinals. In: Logic, Methodology and Philosophy of Science Proc. 1964 Internatational Congress North-Holland, Amsterdam, pp. 3–23 (1965)

[Cour78a] Courcelle, B.: A representation of trees by languages, 1 and II, Theoretical Computer Science, 6, 155-279, 7, 25–55 (1978)

[Cour78b] Courcelle, B.: Frontiers of infinite trees. RAIRO Informatique théorique/Theoretical Computer Science 12, 319–337 (1978)

[Elg75] Elgot, C.: Monadic computation and iterative algebraic theories. In: Shepherdson, J.C. (ed.) Logic Colloquium 1973, Studies in Logic, vol. 80, North Holland, Amsterdam (1975)

[Es80] Ésik, Z.: Identities in Iterative and rational theories. Computational Linguistics and Computer Languages 14, 183–207 (1980)

[EsLab98] Ésik, Z., Labella, A.: Equational properties of iteration in algebraically complete categories. In: Selected papers from the 21st symposium on Mathematical foundations of computer science MFCS '96. Theoretical Computer Science, vol. 195, pp. 61–89 (1998)

[Gr79] Grätzer, G.: Universal Algebra, 2nd edn. Springer, Heidelberg (1979)
[Heil80] Heilbrunner, S.: An algorithm for the solution of fixed-point equations for infinite words. Theoretical Informatics and Applications 14, 131–141 (1980)
[MeWr67] Mezei, J., Wright, J.B.: Algebraic automata and context-free sets. Information and Control 11, 3–29 (1967)
[Ni75] Nivat, M.: On the interpretation of recursive polyadic program schemes. Symposia Mathematica 15, 255–281 (1975)
[Pi86] Pin, J.-E.: Varieties of Formal Languages. Plenum Publishing Corp. New York (1986)
[Ro82] Rosenstein, J.B.: Linear Orderings. Academic Press, New York (1982)
[Scott76] Scott, D.: Data types as lattices. SIAM J. Computing 5, 522–587 (1976)
[Th86] Thomas, W.: On frontiers of regular trees. Theoretical Informatics and Applications 20, 371–381 (1986)

Logical Semantics of Types for Concurrency

Luís Caires

CITI / Departamento de Informática, Universidade Nova de Lisboa,
Portugal

Abstract. We motivate and present a logical semantic approach to types for concurrency and to soundness of related systems. The approach is illustrated by the development of a generic type system for the π-calculus, which may be instantiated for specific notions of typing by extension with adequate subtyping principles. Soundness of our type system is established using a logical predicate technique, based on a compositional spatial logic interpretation of types.

1 Introduction

The aim of this paper is to present a semantic approach to types for concurrency and soundness of related systems, based on spatial logic interpretations. Types are definitely one of the most successful applications of logical methods in concrete programming languages and tools. A type system for a programming language or programming calculus should really be seen as a specialized logic, usually decidable, and presented by a syntax-directed proof system. A classical example is the familiar type system for assigning simple functional types to the λ-calculus. In this case, the properties of interest are absence of errors due to undefined function applications, and (last but not the least) strong normalization. In this case, termination is obtained as a consequence of the soundness of the simple type system with respect to a logical predicate interpretation [22].

As programming languages and calculi evolved, so to include increasingly sophisticated features such as state, exceptions, polymorphism, and concurrency, it has become clearer that classical semantic approaches to prove soundness of type systems did not scale or generalize very well, due to the independent difficulty of finding suitable semantic domains. Fortunately, if one is essentially interested in properties of programs such as absence of certain types of runtime errors, and not really in higher (logical) complexity properties such as termination, more convenient, purely syntactic, proof techniques may frequently be used. As Curry and Feys have put in [9] if one makes sure that “subject reduction preserves the predicate”, and the “predicate” implies absence of immediate errors, then any “subject” program that satisfies the predicate is safe. Motivated by this remark, the (now standard) technique of “subject-reduction” (SR) was first proposed by Felleisen and Wright [23], by letting the “predicate” be identified with formal provability of typing judgments in a system of typing rules.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 16–35, 2007.
© Springer-Verlag Berlin Heidelberg 2007

The SR soundness proof method has certainly been very successful, and revealed to be applicable to various kinds of languages and calculi, in particular, to types for concurrency. In fact, most modern type theoretic analyses of concurrent, distributed, and mobile calculi have been developed in such a framework.

Nevertheless, the purely syntactic SR method is not without its weaknesses, and sometimes appears to have contributed to widespread a too syntactic understanding of types and typing, far from the original semantic view of types as explicit properties or predicates. Usually, SR soundness proofs are quite monolithic, and each intermediate result proceeds by tedious inductions on type derivations. Adding a new construct to the programming language or a new typing rule to the system forces a cross-cutting modification on several auxiliary proofs. This lack of modularity is also caused by the usual absence of any independently defined compositional (algebraic, co-algebraic, or logical) semantics for the type structure. Although one may be careful enough to define such a semantics, unfortunately the SR method does not require such a semantics to be formally defined. Thus, usually we just find some useful but informal intuitions about what the typing rules or the types are intended to mean. It also seems that the SR methods does not by itself improve the degree of reuse foreseen in [23], given the particularities of each operational model.

On the other hand, a semantic proof of soundness builds on an explicitly defined compositional interpretation of types, that potentially provides deeper intuitions, and focuses the proof developments on behavioral aspects of the computational domain, rather than on details of the syntactic presentation of a calculus or of their types as syntactic annotations. In principle, the semantic technique is also more powerful, inducing in general some form of compositionality of typing, and being potentially applicable to properties that are not provable by the SR method (such as termination).

In the original spirit of semantic soundness proofs, we develop in this paper a feasible approach to types for concurrency that combines the advantages of the semantic approach with the technical simplicity of syntactic approaches, such as the SR method. More precisely, we show how the semantics of a general type structure for processes modeled in the π-calculus may be compositionally defined by resorting to a logical interpretation, reminiscent of the logical predicate (or relations) method, and considering as underlying semantic model the standard labeled transition system and associated operational techniques. As in purely semantic approaches, we proceed by defining a compositional semantics of the type language, by induction on the type structure. A formal type system then assigns types to processes by induction on the structure of processes. We illustrate the approach by developing a generic type system $\mathbf{T}$ for the π-calculus, and prove its soundness by showing that typing preserves the validity of typing assertions with respect to an interpretation of types as process predicates.

The generic type system $\mathbf{T}$ may be instantiated to check for various specific properties, just by extending it with appropriate (sound) subtyping principles. In fact, a remarkable advantage of the approach is due to the way the several properties of interest may be factored out. For example, subtyping may be dealt

with as a completely orthogonal aspect, so that our soundness proof does not depend on the syntactic presentation of subtyping, but only on its semantic properties. So, we can pick for subtyping any sound axiomatization of semantic entailment in the underlying logic; soundness of each instance of **T** is then immediately granted as a consequence of this modular approach.

Typically, most interesting process properties of the kinds considered by type systems (*e.g.*, channel arity mismatch) are not invariant under standard behavioral equivalences of processes, for instance, bisimilarity. Therefore, to characterize such kind of properties, the traditional behavioral logics (cf. Hennessy-Milner logics [11]) are not adequate. It turns out that spatial logics for concurrency offer the appropriate expressiveness, as already argued elsewhere [4,5,1,7].

Spatial logics have been proposed with the aim of specifying distributed behavior and other essential aspects of distributed computing systems. An important feature of spatial logics, shared by some other sub-structural logics such as separation logics [20,18], is that its operators are able to separate and count resources; this sometimes seems to add an "intensional" character to these logics (although not always [6]). It is precisely such intensional character that seems necessary for the logical characterization of many type-like properties [4,1]. So, our type language combines behavioral operators, that observe process actions, with spatial logic operators, namely the composition $A \mid B$, and its adjunct $A \triangleright B$. Then, a judgment in the type system **T**, of the form $P :: A \vdash B$, expresses a rely guarantee property and is interpreted by the $A \triangleright B$ operation.

The structure of the paper is as follows. In Section 2, we present an overview of the syntax and semantics of the fragment of the π-calculus we will base our study on. The main intent of Section 3 is to motivate the semantic approach to typing, by providing an alternative proof of soundness for a standard simple type system for the π-calculus. In Section 4, we develop and present the generic type system **T**, and prove its soundness with respect to a logical predicate semantics. We will also consider several incremental extensions to **T**. In Section 5, we will show how **T** may be instantiated so to capture some familiar notions of typing, namely the simple types, I/O types, and some kind of session types. We will close the paper with some conclusions and remarks.

2 The Process Model

In this section, we briefly introduce the syntax and semantics of our intended process model, a fragment of the monadic π-calculus.

Definition 2.1 (Processes). *Given infinite sets Λ of* names (m, n, p)*, and χ of* process variables $(\mathcal{X}, \mathcal{Y})$ *the set $\mathcal{P}$ of* processes (P, Q, R) *is given by*

$$P, Q ::= \mathbf{0} \;\mid\; m(n).P \;\mid\; m\langle n\rangle.P \;\mid\; P \mid Q \;\mid\; (\boldsymbol{\nu} n)P \;\mid\; \mathcal{X} \;\mid\; \mathtt{rec}\,\mathcal{X}.P$$

In restriction $(\boldsymbol{\nu} n)P$ and input $m(n).P$ the distinguished occurrence of name n is binding, with scope the process P. We denote by $\equiv_\alpha$ the relation of α-equivalence on processes: we will implicitly consider processes up to α-equivalence, with care.

For any process P, we assume defined as usual the set $fn(P)$ of *free names* of P. By $\{^m/_n\}$ (resp. $\{^{\mathcal{X}}/_Q\}$) we denote the safe substitution of m by n (resp. of $\mathcal{X}$ by Q), and by $\{m{\leftrightarrow}n\}$ the safe transposition of m and n. Structural congruence expresses basic identities on the spatial structure of processes:

Definition 2.2 (Structural congruence). *Structural congruence* $\equiv$ *is the least congruence relation on processes such that*

$$
\begin{array}{ll}
P \mid \mathbf{0} \equiv P & (\textit{Struct Par Void}) \\
P \mid Q \equiv Q \mid P & (\textit{Struct Par Comm}) \\
P \mid (Q \mid R) \equiv (P \mid Q) \mid R & (\textit{Struct Par Assoc}) \\
n \notin fn(P) \Rightarrow P \mid (\boldsymbol{\nu} n)Q \equiv (\boldsymbol{\nu} n)(P \mid Q) & (\textit{Struct Res Par}) \\
(\boldsymbol{\nu} n)\mathbf{0} \equiv \mathbf{0} & (\textit{Struct Res Void}) \\
(\boldsymbol{\nu} n)(\boldsymbol{\nu} m)P \equiv (\boldsymbol{\nu} m)(\boldsymbol{\nu} n)P & (\textit{Struct Res Comm}) \\
\mathtt{rec}\ \mathcal{X}.P \equiv P\{^{\mathcal{X}}/_{\mathtt{rec}\ \mathcal{X}.P}\} & (\textit{Struct Unfold})
\end{array}
$$

The behavior of processes is defined by a relation of reduction that captures the computations that a process may perform by itself.

Definition 2.3 (Reduction). Reduction *$(P \rightarrow Q)$ is defined as follows:*

$$
\begin{array}{ll}
m\langle n\rangle.Q \mid m(p).P \rightarrow Q \mid P\{^p/_n\} & (\textit{Red React}) \\
Q \rightarrow Q' \Rightarrow P \mid Q \rightarrow P \mid Q' & (\textit{Red Par}) \\
P \rightarrow Q \Rightarrow (\boldsymbol{\nu} n)P \rightarrow (\boldsymbol{\nu} n)Q & (\textit{Red Res}) \\
P \equiv P',\ P' \rightarrow Q',\ Q' \equiv Q \Rightarrow P \rightarrow Q & (\textit{Red Struct})
\end{array}
$$

We denote by $\Rightarrow$ the reflexive-transitive closure of $\rightarrow$. We say that $P \mapsto Q$ if $P \rightarrow Q$ results from a communication on a restricted channel name of P. By $\Mapsto$ we denote the reflexive-transitive closure of $\mapsto$. To observe the interaction between a process and its environment one introduces a labeled transition semantics, in this case, the standard (late) labeled transition system [21]. For that we introduce

Definition 2.4 (Labels). *Labels $\mathcal{L}$ (α, β) are define by*

$$L ::= (\nu n)\alpha \mid m\langle n\rangle \mid m(n) \mid \tau$$

Name restriction on labels is used to express bound output [21]. We assume defined the standard $fn(\alpha)$ (free names) and $bn(\alpha)$ (bound names) of label α.

Definition 2.5 (Labeled Transition System). *The relation of* labeled transition *$(P \stackrel{\alpha}{\rightarrow} Q)$ is defined by the rules:*

$$a(n).P \stackrel{a(n)}{\rightarrow} P\ (\textit{In}) \qquad m\langle n\rangle.P \stackrel{m\langle n\rangle}{\rightarrow} P\ (\textit{Out})$$

$$\frac{P \stackrel{\alpha}{\rightarrow} Q}{P \mid R \stackrel{\alpha}{\rightarrow} Q \mid R}\ (\textit{Par}) \qquad \frac{P \stackrel{\alpha}{\rightarrow} Q \quad n \notin fn(\alpha)}{(\boldsymbol{\nu} n)P \stackrel{\alpha}{\rightarrow} (\boldsymbol{\nu} n)Q}\ (\textit{Res}) \qquad \frac{P\{^{\mathcal{X}}/_{\mathtt{rec}\ \mathcal{X}.P}\} \stackrel{\alpha}{\rightarrow} Q}{\mathtt{rec}\ \mathcal{X}.P \stackrel{\alpha}{\rightarrow} Q}\ (\textit{Rec})$$

$$\frac{P \stackrel{(\nu \overline{s})n\langle m\rangle}{\rightarrow} P' \quad Q \stackrel{n(p)}{\rightarrow} Q'}{P \mid Q \stackrel{\tau}{\rightarrow} (\boldsymbol{\nu} \overline{s})(P' \mid Q'\{^p/_m\})}\ (\textit{Com}) \qquad \frac{P \stackrel{m\langle n\rangle}{\rightarrow} Q}{(\boldsymbol{\nu} n)P \stackrel{(\nu n)m\langle n\rangle}{\rightarrow} Q}\ (\textit{Open})$$

The following provisos apply: rule (Par) subject to fn(Q)#bn(α), rule (Com) subject to $\overline{s}\#Q$, rule (Open) subject to $p \neq m$.

Reduction $\rightarrow$ coincides with silent transition $\xrightarrow{\tau}$, and does not increase the free names of processes. We write $P \xrightarrow{n} Q$ when $P \xrightarrow{\alpha} Q$ and either $\alpha = (\nu\overline{s})n\langle m\rangle$ or $\alpha = n(m)$, and abbreviate a label $(\nu\overline{s})\alpha$ by α when the identity of $\overline{s}$ is not important. Strong late bisimilarity over the labeled transition system defined above is taken as our reference behavioral semantic equivalence of processes.

Definition 2.6. *A* strong late bisimulation $\mathcal{R}$ *is a symmetric binary relation over processes such that for all* P, Q

- *If* $P \mathrel{\mathcal{R}} Q$ *and* $P \xrightarrow{\alpha} P'$ *for some* P' *then exists* Q' *st.* $Q \xrightarrow{\alpha} Q'$ *and* $P' \mathrel{\mathcal{R}} Q'$.
- *If* $P \mathrel{\mathcal{R}} Q$ *and* $P \xrightarrow{n(p)} P'$ *for some* P' *then exists* Q' *st.* $Q \xrightarrow{n(p)} Q'$ *and for all* m $P'\{p/m\} \mathrel{\mathcal{R}} Q'\{p/m\}$.

Strong late bisimilarity $\sim$ *is the greatest strong late bisimulation.*

There are well known characterizations of bisimilarities using modal logics. These are mostly variants of Hennessy-Milner logics [11]. For the strong late bisimilarity case, we may consider the logic $\mathcal{LM}$ [16]: essentially Hennessy-Milner logic augmented with a modality $\langle x(y)\rangle^E A$, such that

$$P \models \langle x(y)\rangle^E A \text{ iff All } Q.P \Rightarrow\xrightarrow{x(y)} Q \text{ implies All } m.\ Q\{y/m\} \models A$$

Two processes P and Q are defined to be *logically equivalent* ($P =_{\mathcal{L}} Q$) for a logic $\mathcal{L}$ if they satisfy exactly the same formulas of $\mathcal{L}$. For $\mathcal{LM}$, one have that $P =_{\mathcal{LM}} Q$ if and only if $P \sim Q$ [16]. In general, purely behavioral logics such as $\mathcal{LM}$ do not distinguish between bisimilar processes. As we shall see in the next section, the kind of properties captured by the simplest type systems for concurrency are not invariant under bisimilarity, and therefore cannot be expressed by logics that just rely on observing process actions.

3 Simple Types

The simplest type systems for concurrent systems modeled in the π-calculus, originating in Milner's system of sorts [15] for the polyadic π-calculus, are intended to enforce communication safety. If a process tries to communicate on a shared channel, but the sender issues a tuple of length different from the one expected by the receiver, an error occurs (undefined synchronization). In our simpler monadic setting, we introduce a slightly different, but in some sense equivalent, notion of error. We partition the set of names Λ into disjoint subsets of channel names Λ_c (x, a, b) and basic names Λ_v (v, u). Then, while channel names may be used to send and receive values, a basic name (cf., an integer value) cannot. More precisely, a process is *wrong*, when it attempts writing to or reading from something that does not refer to a communication channel.

$$\text{Wrong}(P) \triangleq (P \equiv (\boldsymbol{\nu}\overline{m})(a(n).Q \mid R) \text{ or } P \equiv (\boldsymbol{\nu}\overline{m})(a\langle n\rangle.Q \mid R)) \text{ and } a \in \Lambda_v$$

N.B. This may be seen as a special case of arity mismatch: names in Λ_v cannot be used at any arity, and names in Λ_c may be used at all arities (since there is a single arity). We say a process is *safe* if not wrong: $\mathrm{Safe}(P) \triangleq \neg\mathrm{Wrong}(P)$.

Notice that being wrong is not a purely behavioral property, because a wrong process may be bisimilar to a safe process. A type system for arity matching is usually based on formal judgments of the form $\Gamma \vdash P$, where P is the process to be typed, and Γ a typing environment, more precisely, an assignment of a type T to *every* free name of P. We consider channel types (T) and a base type `nil`.

Definition 3.1. *The set of $\mathcal{ST}$ of simple types is given by* $T, U, V ::= \mathtt{nil} \mid (T)$.

Given a finite set of names N, a typing environment of domain N is a mapping Γ assigning each name $n \in D$ a type $\Gamma(n) \in T$ such that $n \in \Lambda_v$ implies $T = \mathtt{nil}$. We denote by $\mathcal{C}$ the set of all typing environments. We denote by $\mathfrak{D}(\Gamma)$ the domain N of Γ. As usual, the typing environment Γ of domain $\{n_1, \ldots, n_m\}$ that maps n_i to T_i may be written $n_1 : T_1, n_2 : T_2, \ldots, n_m : T_m$. Usually, types such as the simple types given above are seen as formal annotations, and type safety for the type system proven by resorting to a subject reduction result. In order to motivate our approach, we will instead develop a semantic proof of soundness. For that purpose, we need to define a compositional interpretation of typing environments as properties (sets of) of processes. We say that a mapping $J[\![-]\!] : \mathcal{C} \to \wp(\mathcal{P})$ is *conjunctive* if $J[\![\Gamma, \Delta]\!] = J[\![\Gamma]\!] \cap J[\![\Delta]\!]$.

Definition 3.2. *A* typing interpretation $J[\![-]\!] : \mathcal{C} \to \wp(\mathcal{P})$ *is a conjunctive mapping assigning to each typing environment a set of processes such that:*

$$
\begin{array}{l}
\textit{If } P \in J[\![n : T]\!] \textit{ then } \mathrm{Safe}(P) \\
\textit{If } P \in J[\![n : T]\!] \textit{ and } P \xrightarrow{\alpha} Q \textit{ then } Q \in J[\![n : T]\!] \\
\textit{If } P \in J[\![n : (U)]\!] \textit{ and } P \xrightarrow{(\boldsymbol{\nu}\overline{s})n\langle m\rangle} Q \textit{ then } Q \in J[\![m : U]\!] \\
\textit{If } P \in J[\![n : (U)]\!] \textit{ and } P \xrightarrow{n(m)} Q \textit{ then } Q \in J[\![m : U]\!] \\
\textit{If } P \in J[\![n : \mathtt{nil}]\!] \textit{ and } P \xrightarrow{n} Q \textit{ then } False
\end{array}
$$

Notice that $J[\![\Gamma]\!]_{\Gamma \in \mathcal{C}}$ is a (typing environment)-indexed family of sets of processes; inductively defined on types, co-inductively defined on transitions. This definition is parametric on the safety predicate $\mathrm{Safe}(-)$, and on standard behavioral observations on processes, expressed by transitions on a labeled transition system. Indeed, if $\mathrm{Safe}(-)$ were closed under bisimilarity (*e.g.*, if $P \sim Q$ and $\mathrm{Safe}(P)$ implies $\mathrm{Safe}(Q)$), then we might check that the corresponding typing interpretation would also be closed under bisimilarity, in the sense that if $P \in J[\![\Gamma]\!]$ and $P \sim Q$ then also $Q \in J[\![\Gamma]\!]$. However, as remarked above, the safety properties of interest captured by type systems are seldom purely behavioral, so that usually any correct (sound) logical interpretation of types is bound to be "intensional" (finer than usual extensional behavioral types). We can check that typing interpretations are closed under arbitrary unions.

Lemma 3.3. *Let $\mathcal{J}$ be a family of typing interpretations. Then $\bigcup_{J \in \mathcal{J}} J$ (defined pointwise as $\Gamma \mapsto \bigcup_{J \in \mathcal{J}} J[\![\Gamma]\!]$) is also a typing interpretation.*

We may then define our interpretation of typing environments.

Definition 3.4. *We define typing, noted $\mathcal{T}[\![-]\!]$, by letting, for all $\Gamma \in \mathcal{C}$,*

$$\mathcal{T}[\![\Gamma]\!] \quad \triangleq \quad \bigcup\{J[\![\Gamma]\!] : \ J \textit{ is a typing interpretation}\}$$

By definition, $\mathcal{T}[\![-]\!]$ is the largest (with relation to the inclusion partial ordering) typing interpretation. It is immediate that if $P \in \mathcal{T}[\![\Gamma]\!]$ and $P \in \mathcal{T}[\![\Delta]\!]$ then $P \in \mathcal{T}[\![\Gamma, \Delta]\!]$ and conversely. We can already verify the key properties of our typing interpretation $\mathcal{T}[\![-]\!]$: these properties hold whenever the type covers all free names of the process. It is typical of predicates of terms defined via realizability or logical relations techniques to characterize the intended properties just when all free variables/names of the subject are covered. We then define

$$P \models_{\mathsf{s}} \Gamma \quad \triangleq \quad P \in \mathcal{T}[\![\Gamma]\!] \textit{ and } \mathit{fn}(P) \in \mathfrak{D}(\Gamma)$$

Lemma 3.5. *The following closure properties of $\mathcal{T}[\![-]\!]$ hold:*

1. $\mathbf{0} \models_s \Gamma$.
2. *If* $P \models_s \Gamma \wedge n : T \wedge m : T$ *then* $P\{^n/_m\} \models_s \Gamma \wedge m : T$.
3. *If* $P \models \Gamma$ *and* $n \notin \mathfrak{D}(\Gamma)$ *then* $P \models_s \Gamma \wedge n : T$.
4. *If* $P \models_s \Gamma$ *and* $Q \models_s \Gamma$ *then* $P \mid Q \models_s \Gamma$.
5. *If* $P \models_s \Gamma \wedge n : T$ *and* $n \notin \mathfrak{D}(\Gamma)$ *then* $(\boldsymbol{\nu} n)P \models_s \Gamma$.
6. *If* $P \models_s \Gamma$ *and* $\Gamma(n) = (U)$ *and* $\Gamma(m) = U$ *then* $n\langle m\rangle.P \models_s \Gamma$.
7. *If* $P \models_s \Gamma \wedge x : U$ *and* $\Gamma(n) = (U)$ *then* $n(x).P \models_s \Gamma$.

Proof. The proof of most cases is by coinduction, given the definition of $\mathcal{T}[\![-]\!]$. It is instructive to look at a few cases (full proofs of this and other results in [2]).

1. We have $\mathbf{0} \in \mathcal{T}[\![n : T]\!]$ for any n and T, since $\mathbf{0} \not\xrightarrow{\alpha}$. Hence $\mathbf{0} \models_{\mathsf{s}} \Gamma$ for any Γ.
2. We show that $S(\Gamma \wedge m : T) \triangleq \{P\{^n/_m\} \mid P \models_{\mathsf{s}} \Gamma \wedge n : T \wedge m : T\}$ is a typing interpretation. Pick $R \in S(\Gamma \wedge m : T)$. Then $R = P\{^n/_m\}$ where $P \models_{\mathsf{s}} \Gamma \wedge n : T \wedge m : T$. Let $R \xrightarrow{\alpha} R'$.
   (a) If $\alpha = \tau$ then $R' \models_{\mathsf{s}} \Gamma \wedge n : T \wedge m : T$, and so $R'\{^n/_m\} \in S(\Gamma \wedge m : T)$.
   (b) if $\alpha = a(v)$ then $P \xrightarrow{b(v)} Q$ where $a = b\{^n/_m\}$. We have $Q \models_{\mathsf{s}} \Gamma \wedge n : T \wedge m : T \wedge v : V$ and $\Gamma_{m,n}(b) = (V)$. If $b \neq n$ then $R' \in S(\Gamma \wedge m : T \wedge v : V)$. If $b = n$ then $\alpha = m(v)$ and $T = (V)$. Then $R' \in S(\Gamma \wedge m : T \wedge v : V)$.
   (c) if $\alpha = (\boldsymbol{\nu}\overline{s})a\langle v\rangle$ then $P \xrightarrow{(\boldsymbol{\nu}\overline{s})b\langle q\rangle} Q$ where $a = b\{^n/_m\}$ and $v = q\{^n/_m\}$. We have $P \xrightarrow{(\boldsymbol{\nu}\overline{s})b\langle q\rangle} Q$, where $Q \models_{\mathsf{s}} \Gamma \wedge n : T \wedge m : T \wedge q : V$ and $\Gamma_{m,n}(b) = (V)$. If $b \neq n$ then $R' \in S(\Gamma \wedge m : T \wedge q : V)$. If $b = n$ then $\alpha = (\boldsymbol{\nu}\overline{s})m\langle v\rangle$ and $T = (\overline{V})$. Then $R' \in S(\Gamma \wedge m : T \wedge q : V)$.

   We conclude that S is a typing interpretation. Then $P \models_{\mathsf{s}} \Gamma \wedge n : T \wedge m : T$ implies $P\{^n/_m\} \in S(\Gamma \wedge m : T) \subseteq \mathcal{T}[\![\Gamma \wedge m : T]\!]$. So $P\{^n/_m\} \models_{\mathsf{s}} \Gamma \wedge m : T$. ∎

$$\frac{}{\Gamma \vdash \mathbf{0}}\ (\textit{ST-Void}) \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}\ (\textit{ST-Par}) \qquad \frac{\Gamma \wedge m : U \vdash P}{\Gamma \vdash (\boldsymbol{\nu} n)P}\ (\textit{ST-Res})$$

$$\frac{\Gamma \wedge m : U \vdash P \quad \Gamma(n) = (U)}{\Gamma \vdash n(m).P}\ (\textit{ST-Inp}) \qquad \frac{\Gamma \vdash P \quad \Gamma(n) = (U) \quad \Gamma(m) = U}{\Gamma \vdash n\langle m \rangle.P}\ (\textit{ST-Out})$$

Fig. 1. Simple type system

Given the previous Lemma 3.5, it is immediate that the (standard) proof system *ST* depicted in Figure 1 is sound for simple types, in the following sense.

Proposition 3.6. *If $\Gamma \vdash P$ and $fn(P) \in \mathfrak{D}(\Gamma)$ then $P \models_s \Gamma$.*

It is interesting to compare the structure of our semantic proof of consistency with a subject reduction style proof. Obviously, both proofs build on the same operational model, and need to go through the verification of essentially the same properties of the processes. The main advantages of the semantic approach result from a fairly different underlying proof structure. The semantic proof is modular (on the structure of the calculus operations), while the subject reduction proof is not. The subject reduction proof proceeds by induction on reduction and typing derivations, while the semantic proof deals with each inference principle in isolation. Thus, to check the soundness of an extended system, one just needs to check the added rules, while a subject reduction proof would have to be mostly redone (or at least, add a new induction case to all existing auxiliary results). Other potential advantages, in particular the smooth incorporation of subtyping principles, were already discussed in the Introduction. Type safety properties are obtained "for free", as an internal consequence of the meaning of each property denoted by a type. For example, we directly conclude the semantic counterparts of the familiar type safety and subject reduction statements:

Proposition 37 (Type Safety)

1. *If $P \models_s \Gamma$ then* Safe(P)*, and if $P \models_s \Gamma$ and $P \to Q$ then $Q \models_s \Gamma$.*
2. *If $\Gamma \vdash P$ and $P \Rightarrow Q$ then* Safe(Q)*.*

Proof. (1) By definition of $\mathcal{T}[\![-]\!]$. (2) By (1) and Proposition 3.6. ■

In the case of simple types just discussed, a standard SR proof would have been perhaps more concise. However, the advantages of the semantic approach start to become clearer when more complex type systems are considered. In the next section, we develop a general type system **T** for the π-calculus, based on a behavioral-spatial logic, and prove its soundness using the semantic approach illustrated above. As we shall show later in the paper, the logical primitives of this type system provide a suitable "meta-language" in which the (type-like, intensional) properties captured by many different type systems may be expressed as idioms. Although it would be straightforward to extend **T** with new inference rules, we will show that many interesting instances may be obtained by merely adding new subtyping axioms and rules. The soundness of such subtyping

principles may be proven modularly and incrementally, resorting to the semantic approach. The soundness of each (conservative) extension of the type system **T** considered will then be obtained in fairly automatic way.

4 The Generic Type System T

In this section, we present a general type system for the π-calculus, motivated by a logical semantic of types as properties (sets of processes), and prove soundness of typing (Theorem 4.5) and subtyping (Theorem 4.6). Our presentation is close to a presentation of a logic. This is not unexpected, any type system should be seen as a compositional, decidable, and (usually incomplete) proof system for a specialized logic. We start by defining the syntax of types.

Definition 4.1 (Types). *Types of **T** are given by the following abstract syntax:*

$$\begin{array}{lcl} \alpha & ::= & x.!(T) \triangleright \mid x.?(T) \triangleright \mid x.!(T)\,; \mid x.?(T)\,; \\ A, B, C & ::= & \emptyset \mid \mathsf{F} \mid A \wedge B \mid A|B \mid A \triangleright B \mid \mathsf{H}n.A \mid \alpha A \mid [\alpha]A \mid \Box A \\ & \mid & \texttt{rec}\ X.A \mid X \end{array}$$

N.B.: We write $T(n)$ (also $n : T$) for a type A with a single name n occurring. Then, we refer by T the name-abstracted type $T(-)$.

Name abstracted types, such as T above, appear as arguments to behavioral operators α, to type channels parameters. *E.g.*, if $T(-) = -.!()\,;\, -.?()\,;\emptyset$, then $T(n) = n : T = n.!()\,;n.?()\,;\emptyset$, replacing the hole $-$ with n in T; likewise $n : \emptyset = \emptyset$.

For each type A we define the set $fn(A)$ of free names as usual, considering n bound in $\mathsf{H}n.A$. In any type $A \triangleright B$ we require $fn(A)\#fn(B)$.

At least superficially, the type language of **T** is not far from the spatial logics for concurrency, as presented in [4,5,1]. In fact, we may construct an embedding of **T** in the spatial logic of [4,5]. However, we are here interested in a more refined and specific semantics, reflecting the intended safety properties of types. The semantics of types as logical predicates on processes is given by the relation of satisfaction $P \models A$ defined between processes P and formulas A.

Definition 4.2 (Satisfaction). *Semantics of types is inductively defined as shown in Figure 2. N.B. We define $P \models_n A$ as $P \models A$ and $fn(P) \subseteq fn(A)$.*

Spatial composition $A \mid B$ is interpreted in the standard way, while enforcing free name containment. In the hidden name quantifier $\mathsf{H}n.A$ the name n is bound; this construct is a generic mechanism allowing us to define types with bound names, even if most of such names will be elided away by subtyping. Behavioral modalities are classified along two dimensions: input/output, and spatial/sharing (depending on whether the argument is handled via the spatial (|) or sharing ($\wedge$) conjunction). $\Box$ is useful to express invariants. We omit a full treatment of recursive types, interpreted as greatest fixed point, that will not bring unexpected difficulties. We define the abbreviations:

$$\begin{array}{ll} P{\Downarrow_{safe}} & \triangleq \textit{All } R.\ P \Rrightarrow R \textit{ implies } \mathrm{Safe}(R) \\ P{\Rrightarrow_{safe}}Q & \triangleq P{\Downarrow_{safe}} \textit{ and } P \Rrightarrow R \end{array}$$

We can now state some fundamental properties of the satisfaction relation.

$$
\begin{array}{ll}
P \models \mathsf{F} & \textit{iff False} \\
P \models \emptyset & \textit{iff } P \Downarrow_{safe} \\
P \models A \mid B & \textit{iff } P \equiv R \mid Q \textit{ and } R \models_n A \textit{ and } Q \models_n B \\
P \models A \vdash B & \textit{iff All } R.\ R \models_n A \textit{ implies } (\boldsymbol{\nu} fn(A))(R \mid P) \overset{safe}{\Mapsto} \models B \\
P \models A \wedge B & \textit{iff } P \models A \textit{ and } P \models B \\
P \models \mathsf{H} n.A & \textit{iff } P \models A \textit{ and } n \# fn(P, \mathsf{H} n.A) \\
P \models \Box A & \textit{iff } \mathrm{Safe}(P) \textit{ and All } \alpha.\ \textit{if } P \xrightarrow{\alpha} Q \textit{ then } Q \models A \\
P \models x.!(T) \triangleright A & \textit{iff } P \Downarrow_{safe} \textit{ and if } P \Mapsto R \xrightarrow{\alpha} Q \textit{ then } \alpha = (\nu) x\langle n \rangle,\ Q \models A \mid n : T \textit{ and } n \# A \\
P \models x.?(T) \triangleright A & \textit{iff } P \Downarrow_{safe} \textit{ and if } P \Mapsto R \xrightarrow{\alpha} Q \textit{ then } \alpha = x(n),\ Q \models n : T \vdash A \\
P \models x.!(T); A & \textit{iff } P \Downarrow_{safe} \textit{ and if } P \Mapsto R \xrightarrow{\alpha} Q \textit{ then } \alpha = (\nu) x\langle n \rangle,\ Q \models A \wedge n : T \textit{ and } n \# A \\
P \models x.?(T); A & \textit{iff } P \Downarrow_{safe} \textit{ and if } P \Mapsto R \xrightarrow{\alpha} Q \textit{ then } \alpha = x(n),\ Q \models n : T \wedge A \\
P \models [x.!(T) \triangleright] A & \textit{iff } P \Downarrow_{safe} \textit{ and All } n.\ \textit{if } P \Mapsto R \xrightarrow{(\nu) x\langle n \rangle} Q \textit{ then } Q \models A \mid n : T \textit{ and } n \# A \\
P \models [x.?(T) \triangleright] A & \textit{iff } P \Downarrow_{safe} \textit{ and All } n.\ \textit{if } P \Mapsto R \xrightarrow{x(n)} Q \textit{ then } Q \models n : T \vdash A \\
P \models [x.!(T);] A & \textit{iff } P \Downarrow_{safe} \textit{ and All } n.\ \textit{if } P \Mapsto R \xrightarrow{(\nu) x\langle n \rangle} Q \textit{ then } Q \models A \wedge n : T \textit{ and } n \# A \\
P \models [x.?(T);] A & \textit{iff } P \Downarrow_{safe} \textit{ and All } n.\ \textit{if } P \Mapsto R \xrightarrow{x(n)} Q \textit{ then } Q \models A \wedge n : T
\end{array}
$$

Fig. 2. Logical semantics of types

Lemma 4.3. *Properties of satisfaction.*

1. *Let* $P \models A$. *If* $P \equiv Q$ *then* $Q \models A$.
2. *Let* $P \models A$. *If* $P \Mapsto Q$, *then* $Q \models A$.
3. *Let* $P \models A$. *Then* $P \Downarrow_{safe}$.
4. *Let* $P \models A$. *Then* $P\{m \leftrightarrow n\} \models A\{m \leftrightarrow n\}$.
5. *Let* $P \models A$. *If* $n \notin A$, *then* $(\boldsymbol{\nu} n)P \models A$.

Proof. Induction on the structure of type A. ∎

4.1 Type System

The typing rules of our generic type system **T** is based on formal judgments of two forms: typing judgments and subtyping judgments.

$A <: B$ (*Subtyping Judgment*) $\qquad$ $P :: A \vdash B$ (*Typing Judgement*)

Some formation rules apply. Intuitively, a typing judgment expresses a rely guarantee property, interpreted by the composition adjunct operator of the underlying logic. Thus, we require in any such judgment that $fn(A) \# fn(B)$. Moreover, we require the antecedent to be separated, in the sense that for all composition types $C \mid D$ occurring in the A, we must have $fn(C) \# fn(D)$. On the other hand, the right-hand side B is not subject to any special proviso. These constraints will be preserved by all inference axioms and rules, via adequate provisos.

$$\frac{(\ Void\ )}{\mathbf{0} :: \emptyset \triangleright \emptyset}$$

$$\frac{\begin{array}{c}(\ Out\text{-}Left\ )\\ (y\ nfc.)\\ P :: A \mid C \mid y : T \vdash B\end{array}}{x(y).P :: x :!(T) \triangleright A \mid C \vdash B} \qquad \frac{\begin{array}{c}(\ In\text{-}Right\ )\\ (y\ nfc.)\\ P :: A \mid y : T \vdash B\end{array}}{x(y).P :: A \vdash x :?(T) \triangleright B}$$

$$\frac{\begin{array}{c}(\ In\text{-}Left\ )\\ P :: A \mid C \vdash B\end{array}}{x\langle n\rangle.P :: x :?(T) \triangleright A \mid n : T \mid C \vdash B} \qquad \frac{\begin{array}{c}(\ Out\text{-}Right\ )\\ P :: A \vdash B\end{array}}{x\langle n\rangle.P :: A \mid n : T \vdash x :!(T) \triangleright B}$$

$$\frac{\begin{array}{c}(\ Par\ )\\ P :: A \vdash B \quad Q :: C \vdash D\end{array}}{(P \mid Q) :: A \mid C \vdash B \mid D} \qquad \frac{\begin{array}{c}(\ Rec\ )\\ (P :: A \vdash \alpha) \quad P :: A \vdash B\end{array}}{P :: A \vdash \mathtt{rec}\ \alpha.B}$$

$$\frac{\begin{array}{c}(\ Sub\ )\\ A <: A' \quad P :: A' \vdash B' \quad B' <: B\end{array}}{P :: A \vdash B}$$

$$\frac{\begin{array}{c}(\ Seq\ )\\ (fn(B)\ nfc.)\\ P :: A \vdash A' \mid B \quad Q :: B \mid B' \vdash C\end{array}}{(\boldsymbol{\nu} B)(P \mid Q) :: A \mid B' \vdash A' \mid C} \qquad \frac{\begin{array}{c}(\ Res\ )\\ (n\ nfc.)\\ P :: A \vdash B\end{array}}{(\boldsymbol{\nu} n)P :: A \vdash \mathsf{H}n.B}$$

Fig. 3. The Generic Type System **T**

Judgments express certain assertions about types and processes. The meaning of such assertions is given by the notion of validity.

Definition 4.4 (Validity). *Validity of judgments is defined as follows.*

$$\begin{array}{ll} valid(P :: A \vdash B) & \triangleq P \models_n A \triangleright B \\ valid(A <: B) & \triangleq All\ P.\ if\ P \models_n A\ then\ P \models_n B \end{array}$$

A proof system for subtyping is sound if whenever it derives $A <: B$, then $valid(A <: B)$. Likewise, a proof system for typing is sound if whenever it derives $P :: A \triangleright B$ then $valid(P :: A \triangleright B)$. An immediate consequence of soundness of typability is that if $P :: \emptyset \triangleright \emptyset$ is derivable, then, by Lemma 4.3(2,3), we conclude that for all Q such that $P \Rightarrow Q$ we have Safe(Q).

In Figure 3, we present the rules of the generic type system **T**. A proviso of all rules is that only well-formed judgments may be concluded, and $x \in \Lambda_c$. Notice that typing depends on subtyping just in the (*Sub*) rule. As in any type system, the rules are directed by the syntax of processes (even if we may have more than one rule for each construct). A main result of this paper is then:

Theorem 4.5 (Soundness of Type System *T*). *Let $A <: B$ be any sound subtyping relation. If $P :: A \triangleright B$ is derivable in T, then $valid(P :: A \triangleright B)$.*

Proof. We show that each rule preserves validity. We start by showing the following fact (induction on A): if A is separated, and $R \models A$ then $R \Rightarrow Q$ implies $R \mapsto Q$. We consider each rule in turn; it is interesting to look at a few cases.

- (Case of (*Void*)) Pick $P \models_{\mathtt{n}} \emptyset$. Then $P \mid \mathbf{0} \equiv P$, by closure of satisfaction under structural congruence, and we conclude $P \mid \mathbf{0} \models \emptyset$. Thus $\mathbf{0} \models_{\mathtt{n}} \emptyset \triangleright \emptyset$.
- (Case of (*Par*)) Pick R such that $R \models_{\mathtt{n}} A \mid C$. Then $R \equiv R_1 \mid R_2$ where $R_1 \models_{\mathtt{n}} A$ and $R_2 \models_{\mathtt{n}} B$. By the premises, $(\boldsymbol{\nu} A)(R_1 \mid P)\Rrightarrow_{safe} \models_{\mathtt{n}} B$ and $(\boldsymbol{\nu} C)(R_2 \mid Q)\Rrightarrow_{safe} \models_{\mathtt{n}} D$. We have $B\#C$ and $A\#D$ and $A\#C$. Hence $(\boldsymbol{\nu} AC)(R \mid P \mid Q)\Rrightarrow_{safe} \models_{\mathtt{n}} B \mid D$, and so $(P \mid Q) \models_{\mathtt{n}} A \mid C \triangleright B \mid D$.
- (Case of (*Seq*)) Pick any R such that $R \models_{\mathtt{n}} A \mid B'$. So $R \equiv R_1 \mid R_2$ where $R_1 \models_{\mathtt{n}} A$ and $R_2 \models_{\mathtt{n}} B'$. We know that $A\#C$ and $A'\#B'$.
  By left premise, $(\boldsymbol{\nu} A)(R_1 \mid P)\Rrightarrow_{safe} T \models_{\mathtt{n}} A' \mid B$. Then $T = T_1 \mid T_2$ where $T_1 \models_{\mathtt{n}} A'$ and $T_2 \models_{\mathtt{n}} B$. Thus $T_2 \mid R_2 \models_{\mathtt{n}} B \mid B'$. By right premise, we get $(\boldsymbol{\nu} BB')(T_2 \mid R_2 \mid Q)\Rrightarrow_{safe} \models_{\mathtt{n}} C$, and so $T_1 \mid (\boldsymbol{\nu} BB')(T_2 \mid R_2 \mid Q)\Rrightarrow_{safe} \models_{\mathtt{n}} A' \mid C$. Then

$$(\boldsymbol{\nu} BB')(T_1 \mid T_2 \mid R_2 \mid Q)\Rrightarrow_{safe} \models_{\mathtt{n}} A' \mid C$$
$$(\boldsymbol{\nu} BB')(T \mid R_2 \mid Q)\Rrightarrow_{safe} \models_{\mathtt{n}} A' \mid C$$
$$(\boldsymbol{\nu} BB')(\boldsymbol{\nu} A)(R \mid P \mid Q)\Rrightarrow_{safe} \models_{\mathtt{n}} A' \mid C$$

  by Lemma 4.3(1). Since R is arbitrary, $(\boldsymbol{\nu} B)(P \mid Q) \models_{\mathtt{n}} A \mid B' \triangleright A' \mid C$.
- (Case of (*In-Right*)) From $P :: A \mid y : T \vdash B$ we get $x(y).P :: A \vdash x.?(T) \triangleright B$. Pick R such that $R \models_{\mathtt{n}} A$. Let $S \triangleq (\boldsymbol{\nu} A)(R \mid x(y).P)$. Since $x \notin A$, we have $x \notin fn(R)$. If $S \Rrightarrow\xrightarrow{\alpha} S'$ with a visible ($\neq \tau$) action α, then $S' \equiv (\boldsymbol{\nu} A)(R' \mid P)$ where $R \Rrightarrow_{safe} R'$ and $\alpha = x(y)$ for some $y\#A, R$. By validity of the premise, we have $P \models_{\mathtt{n}} (A \mid y : T) \triangleright B$. By Lemma 4.3(2), $R' \models_{\mathtt{n}} A$. Pick any $Q \models_{\mathtt{n}} y : T$. Then $(\boldsymbol{\nu} y)(Q \mid S')\Rrightarrow_{safe} \models_{\mathtt{n}} B$. So $S' \models_{\mathtt{n}} y : T \triangleright B$. Hence $(\boldsymbol{\nu} A)(R \mid x(y).P) \models_{\mathtt{n}} x.?(T) \triangleright B$. We conclude $x(y).P \models_{\mathtt{n}} A \triangleright x.?(T) \triangleright B$. ∎

4.2 Subtyping

We have deliberately left open the definition of any concrete subtyping relation, in order to given a general soundness result for the core system **T**, independently of any such subtyping relation. However, one expects any interesting subtyping relation to contain at least the deductive closure of the proof system in Figure 4. These principles essentially state the commutative monoidal structure of spatial composition $- \mid -$ with unit $\emptyset$, congruence principles, and (logical) scope extrusion rules for the hidden name quantifier (see [4]). We may then show

Theorem 4.6 (Soundness of Subtyping). *Let A <: B be derivable in $\boldsymbol{T}_{<:}$. Then valid(A <: B).*

Proof. We show that each rule preserves validity of <: judgments. The proof is straightforward for most axioms, using Lemma 4.3(2). For (*HidWeak*), we show (induction on B) that $P \models_{\mathtt{n}} B$ and Safe(Q) implies $P \mid Q \models B$. ∎

4.3 Sharing

The typing rules of the core type system **T** presented above do not make special use of conjunctive types. In fact, only "linear" usages of channel names seem

A <:> $A \mid \emptyset$ (*ParVoid*)	$A \mid B$ <:> $B \mid A$ (*ParCom*)
$A \mid (B \mid C)$ <:> $(A \mid B) \mid C$ (*ParAssoc*)	A <: $B \Rightarrow A \mid C$ <: $B \mid C$ (*ParCong*)
$\mathsf{H}\lvert A \rvert.(A \mid B)$ <: B (*HidWeak*)	$\mathsf{H}\lvert A \rvert.\emptyset$ <: $\emptyset$ (*HidVoid*)
$A \mid \mathsf{H}n.B$ <: $\mathsf{H}n.(A \mid B)$ (*HidExt*)	A <: $B \Rightarrow \mathsf{H}n.A$ <: $\mathsf{H}n.B$ (*HidCong*)
A <:> $A \wedge A$ (*ConjAdd*)	A <:> $A \wedge \emptyset$ (*ConjVoid*)
$A \wedge B$ <:> $B \wedge A$ (*ConjCom*)	$(A \wedge B) \wedge C$ <:> $A \wedge (B \wedge C)$ (*ConjAssoc*)
A <: $B \Rightarrow A \wedge C$ <: $B \wedge C$ (*ConjCong*)	A <: $B \Rightarrow \alpha A$ <: αB (*ActCong*)
F <: A (*Bot*)	A <: $B \Rightarrow [\alpha]A$ <: $[\alpha]B$ (*ActCong*)
$\emptyset$ <: αA (*ActVoid*)	$\emptyset$ <: $[\alpha]A$ (*ActVoid*)

Fig. 4. Basic Subtyping Axioms and Rules $\mathbf{T}_{<:}$

to be allowed. We will now show how conjunctive types may be used to type general forms of sharing, and express common properties of type systems, as the ones described in Section 3. We start by defining

Definition 4.7. *A family $\mathcal{F}$ of types is* sharing *if its is closed under conjunction, and satisfies the following contraction conditions relative to the spatial and sharing conjunctions, for any types A, $m : T$, and $n : T$ in $\mathcal{F}$:*

1. *$A \mid A \models_n A$.*
2. *If $P \models_n A \wedge n : T \wedge m : T$ then $P\{^n/_m\} \models_n A \wedge m : T$.*

For example, the simple types of Section 3 are sharing, in face of Lemma 3.5(2,4).

Notice that as far as behavioral type constructors are concerned, the αA types express fairly strong safety properties, while $[\alpha]A$ types are close to the Hennessy-Milner logic operators. For that reason, we will call *classical* those types with no occurrences of spatial ($A \mid B$, $A \triangleright B$) or αA operators. We also call *invariant* any type A such that $A \models \Box A$.

We then prove the following (perhaps surprising) result, that shows that spatial and shared properties may be composed for the important class of classical (purely behavioral) types.

Lemma 4.8 (Spatial/Sharing Cut). *Let C by a classical invariant and $R \models A \wedge C$ and $P \models A \triangleright B$, with $fn(A) \# fn(C)$. Then $(\boldsymbol{\nu} A)(P \mid R) \models B \wedge C$.*

Proof. Since $R \models A$, we have $(\boldsymbol{\nu} A)(P \mid R) \models B$. We have $R \models C$. By induction on the type C, we show that $(\boldsymbol{\nu} A)(P \mid R) \models C$. ∎

Conjunctive typing rules are depicted in Figure 5, all (*In-S-*) and (*Out-S-*) rules are subject to the proviso that the types in the right-hand-side of the premises (e.g, $B \wedge y : T$), belong to an invariant sharing type family. Essentially, we define a left and right rule for input and output processes, and the "sharing cut" (cf. the (*Seq*) typing rule) motivated by Lemma 4.8. As before, we can state:

Proposition 4.9. *The sharing typing rules* **S** *are sound.*

Proof. We show that each rule preserves validity. We show here a few cases.

1. (Case of (*Sharing-Cut*)) By Lemma 4.8(1).

$$\frac{P :: A \vdash B \wedge y : T}{x(y).P :: A \vdash x :?(T) ; B}\ (\textit{y nfc.})\ \ (\textit{In-S-Right}) \qquad \frac{P :: A \vdash B \wedge y : T}{x(y).P :: x :!(T) ; A \vdash B}\ (\textit{y nfc.})\ \ (\textit{In-S-Left})$$

$$\frac{P :: A \vdash B \wedge n : T \quad (n\#B)}{x\langle n\rangle.P :: x :?(T) ; A \vdash B \wedge n : T}\ (\textit{Out-S-Left}) \qquad \frac{P :: A \vdash B \wedge n : T \quad (n\#B)}{x\langle n\rangle.P :: A \vdash x :!(T) ; B \wedge n : T}\ (\textit{Out-S-Right})$$

$$\frac{B\#C \quad C \textit{ classical invariant} \quad P :: A \vdash B \wedge C \quad Q :: B \vdash D}{(\boldsymbol{\nu} B)(P \mid Q) :: A \vdash D \wedge C}\ (\textit{Sharing-Cut})$$

Fig. 5. Sharing typing rules **S**

$$\frac{P :: A \vdash B \mid n : T}{(\boldsymbol{\nu} n)x\langle n\rangle.P :: A \vdash x :!(T) \triangleright B}\ (\textit{Out-P-Right}) \qquad \frac{P :: A \vdash B \mid n : T}{(\boldsymbol{\nu} n)x\langle n\rangle.P :: x :?(T) \triangleright \emptyset \mid A \vdash B}\ (\textit{Out-P-Left})$$

Fig. 6. Bound output typing rules **P**

2. (Case of (*In-S-Left*)) We have $x(y).P :: x.!(T) ; A \vdash B$ derived from $P :: A \vdash y : T \wedge B$, where $x\#A$. Pick R such that $R \models x.!(T) ; A$. Let $S \triangleq (\boldsymbol{\nu} Ax)(R \mid x(y).P)$. Consider any reduction from S: it has the form $S \Rightarrow S'$ with $S' \equiv (\boldsymbol{\nu} Ax\overline{s})(R' \mid P\{^{y}/_{n}\})$ where $R {\Rrightarrow}_{safe} \xrightarrow{\alpha} R'$ and $\alpha = (\boldsymbol{\nu}\overline{s})x\langle n\rangle$ and $R' \models A \wedge n : T$ with $n\#A$. By Lemma 4.8(1), $(\boldsymbol{\nu} A)(R' \mid P) \models n : T \wedge B \wedge y : T$. Since $B \wedge y : T$ is sharing, $(\boldsymbol{\nu} A)(R' \mid P\{^{y}/_{n}\}) \models n : T \wedge B$. We conclude $x(y).P :: x.!(T) ; A \vdash B$.
3. (Case of (*Out-S-Right*)) We have $x\langle n\rangle.P :: A \vdash x.!(T) ; B \wedge n : T$ concluded from $P :: A \vdash B \wedge n : T$. Pick $R \models A$, and let $S \triangleq (\boldsymbol{\nu} A)(R \mid x\langle n\rangle.P)$.
Since $n, x\#A$, if $S \Rrightarrow\xrightarrow{\alpha} S'$ with a visible action α, then $S' \equiv (\boldsymbol{\nu} A)(R' \mid P)$, $R {\Rrightarrow}_{safe} R'$, and $\alpha = x\langle n\rangle$. Then $R' \models_{\mathtt{n}} A$. By the premise, we have $P \models_{\mathtt{n}} A \triangleright B \wedge n : T$, and thus $S' \models B \wedge n : T$ with $n\#B$. Since R is arbitrary, we have $x\langle n\rangle.P \models A \triangleright x :!(T) ; B$. Then $x\langle n\rangle.P \models_{\mathtt{n}} A \triangleright x.!(T) ; B \wedge n : T$. ■

4.4 Additional Typing Rules

We may still consider additional typing and subtyping rules. For example, reasoning by symmetry, it would seem sensible to consider a variation of the output rules (*Out-Right*) and (*Out-Left*) of **T**, where the source of the output is obtained from the continuation of the output process, rather than from the parallel spatial context. It is interesting to notice that this pattern of resource hand-over seems associated to bound output. We illustrate this development by introducing the post output **P** rules depicted in Figure 6. These rules do not seem derivable from the ones already presented. In any case, we are not concerned here with finding a minimal (in some well defined sense) set of rules, but rather to illustrate the modularity and flexibility of the approach. We have

Proposition 4.10. *The bound output typing rules* **P** *are sound.*

5 Some Instances of Typing and Subtyping

In previous sections, we have motivated and developed the generic type system T, and prove its soundness using a semantic technique based on a logical interpretation of types. In this section, we discuss the expressiveness of our framework, by showing how some type systems of well-known kind, namely, simple types, I/O types, and a form of session types, may be embedded in a fairly direct way in the type system **T**, just by choosing suitable (sound) additional subtyping axioms.

5.1 Simple Types

It is instructive to elaborate a representation of the simple type system of Section 3 in the general type system **T**. Essentially, we need to express typing interpretations (Definition 3.2) using our type primitives. We set

$$\begin{array}{ll} \lceil \mathtt{nil} \rceil(n) & \triangleq \mathtt{rec}\ X.\emptyset \wedge [n.!(\emptyset);]\mathsf{F} \wedge [n.?(\emptyset);]\mathsf{F} \wedge \Box X \\ \lceil (T) \rceil(n) & \triangleq \mathtt{rec}\ X.\emptyset \wedge [n.!(\lceil T \rceil);]X \wedge [n.?(\lceil T \rceil);]X \wedge \Box X \\ \lceil\ \rceil & \triangleq \emptyset \\ \lceil n : T, \Gamma \rceil & \triangleq \lceil T \rceil(n) \wedge \lceil \Gamma \rceil \end{array}$$

Notice that the translation of a typing context Γ essentially just spells out, fairly directly, the coinductive definition of $[\![\Gamma]\!]$ of Definition 3.2. It is then not difficult to check that $P \models_{\mathtt{n}} \lceil \Gamma \rceil$ if and only if $P \models_{\mathtt{s}} \Gamma$. We will then deliberately mix (the syntax of) simple types (U, V, T) with general types, with the assumption that the former are seen as the abbreviations defined above. We may then show:

Proposition 5.1. *The subtyping judgments* $\mathbf{ST}_{\texttt{<:}}$ *, listed below, are valid:*

$$\begin{array}{ll} \emptyset \texttt{ <: } \Gamma & (\mathit{Weaken}) \\ n.?(T);(\Gamma \wedge n : (T)) \texttt{ <: } \Gamma \wedge n : (T) & (\mathit{ContrInp}) \\ n.!(T);(\Gamma \wedge n : (T)) \texttt{ <: } \Gamma \wedge n : (T) & (\mathit{ContrOut}) \\ \Gamma \mid \Gamma \texttt{ <: } \Gamma & (\mathit{ContrPar}) \\ \mathsf{H}n.(\Gamma \wedge n : T) \texttt{ <: } \Gamma & (\mathit{ContrRes}) \end{array}$$

Proof. Verification is direct for (Weaken). We show $(\mathit{ContrOut})$ this in detail, for $(\mathit{ContrInp})$ is similar. Let $P \models_{\mathtt{n}} n.!(T);(\Gamma \wedge n : (T))$, and $P \xrightarrow{\alpha} Q$. By the assumption, either $\alpha = \tau$ and $Q \models n.!(T);(\Gamma \wedge n : (T))$, or $\alpha = (\boldsymbol{\nu}\overline{s})n\langle m \rangle$ and $R \models \Gamma \wedge n : (T) \wedge m : T$ with $m \# \Gamma, n$. By coinduction, we conclude $P \models \Gamma \wedge n : (T)$. $(\mathit{ContrPar})$ and $(\mathit{ContrRes})$ are valid by Lemma 3.5(4,5). ∎

The laws presented in Proposition 5.1 express weakening and contraction principles that confirm the "exponential" (sharing) character of simple types. Notice that these principles are justified by semantics entailments, independently of any proof theoretic considerations. We can also verify that simple types are classical, and sharing (Definition 4.7), as a consequence of Lemma 3.5(4,5). If one considers these contraction principles in (the subtyping relation of) the general type

system $\mathbf{T}$, obtaining the system $\mathbf{T} + \mathbf{ST}_{<:}$, then we may show that each rule of the Simple type system (in Figure 1) becomes admissible.

Proposition 5.2. *The Simple type system is admissible in* $\mathbf{T} + \mathbf{ST}_{<:}$.

Proof. Each judgment $\Gamma \vdash P$ is represented by $P :: \emptyset \vdash \lceil \Gamma \rceil$ in $\mathbf{T}$. Then (*ST-Void*) is admissible by (*Void*) and subtyping by (*Weakening*). (*ST-Par*) is admissible by (*Par*) and subtyping by (*ContrPar*). (*ST-Res*) is admissible by (*Res*) and subtyping by (*ContrRes*). (*ST-Inp*) is represented as follows:

$$
\begin{array}{ll}
1.\ \Gamma, n : U, m : (U) \vdash P & P :: \emptyset \vdash \lceil \Gamma \rceil \wedge \lceil n : U \rceil \wedge \lceil m : (U) \rceil \\
2. & P :: \emptyset \vdash \lceil \Gamma \rceil \wedge \lceil U \rceil(n) \wedge \lceil m : (U) \rceil \\
3. & m\langle n \rangle.P :: \emptyset \vdash m.!(\lceil U \rceil);(\lceil \Gamma \rceil \wedge \lceil m : (U) \rceil) \wedge \lceil U \rceil(n) \\
4.\ \Gamma, m : (U) \vdash m(n).P & m(n).P :: \emptyset \vdash \lceil \Gamma \rceil \wedge \lceil m : (U) \rceil \wedge \lceil n : U \rceil
\end{array}
$$

(2,3) by (*Out-S-Right*) and (3,4) (*Sub*) (by (*ContrOut*)). (*ST-In*) is similar. ■

5.2 I/O Types

We show how I/O types, along the lines of [19], may be represented in the type system $\mathbf{T}$. I/O types are similar to simple types, but now channel types are tagged with a mode $\mu \in \{+, -, \pm\}$. Standard channel types (U), now written $(U)^{\pm}$, are refined into input only $(U)^{-}$ and output only $(U)^{+}$ channel types. A logical semantics of I/O types may be given as follows.

$$
\begin{array}{l}
\lceil (T)^{\pm} \rceil(n) \triangleq \mathtt{rec}\ X.\emptyset \wedge [n.!(\lceil T \rceil^{\circ});]X \wedge [n.?(\lceil T \rceil);]X \wedge \Box X \\
\lceil (T)^{+} \rceil(n) \triangleq \mathtt{rec}\ X.\emptyset \wedge [n.!(\lceil T \rceil^{\circ});]X \wedge [n.?(\emptyset);]\mathsf{F} \wedge \Box X \\
\lceil (T)^{-} \rceil(n) \triangleq \mathtt{rec}\ X.\emptyset \wedge [n.!(\emptyset);]\mathsf{F} \wedge [n, ?(\lceil T \rceil);]X \wedge \Box X \\
\lceil (T)^{\pm} \rceil^{\circ}(n) = \mathtt{rec}\ X.\emptyset \wedge [n.!(\lceil T \rceil^{\circ});]X \wedge [n.?(\lceil T \rceil^{\circ});]X \wedge \Box X \\
\lceil (T)^{+} \rceil^{\circ}(n) = \lceil (T)^{-} \rceil^{\circ}(n) \triangleq \lceil (T)^{\pm} \rceil(n)
\end{array}
$$

Again, we notice that the translation above offers a fairly direct specification of the semantics of I/O types, and that these are introduced as an orthogonal (conservative) extension of simple types. Indeed, we can check that if T is a type containing just the $(-)^{\pm}$ type constructor, and U is the simple type "erasure" of T then $[\![T]\!] = [\![U]\!]$. We may also verify that all the subtyping principles stated in Proposition 5.1 remain valid for I/O types. Moreover, we have

Proposition 5.3. *The subtyping rules* $\mathbf{IO}_{<:}$ *are valid for any I/O types* U, T:

$$
\begin{array}{lll}
n : (T)^{+} <: n : (T)^{\pm} & & (\mathit{InpIO}) \\
n : (T)^{-} <: n : (T)^{\pm} & & (\mathit{OutIO}) \\
U <: T \Rightarrow n.?(U);(\Gamma \wedge n : (T)^{\mu}) <: \Gamma \wedge n : (T)^{\mu} & (- \in \mu) & (\mathit{ContrIOInp}) \\
T <: U \Rightarrow n.!(U);(\Gamma \wedge n : (T)^{\mu}) <: \Gamma \wedge n : (T)^{\mu} & (+ \in \mu) & (\mathit{ContrIOOut})
\end{array}
$$

$$
\frac{n : T <: n : U}{n : (T)^{-} <: n : (U)^{-}}\ (\mathit{SubInp}) \qquad
\frac{n : U <: n : T}{n : (T)^{+} <: n : (U)^{+}}\ (\mathit{SubOut}) \qquad
\frac{n : U <:> n : T}{n : (T)^{\pm} <: n : (U)^{\pm}}\ (\mathit{SubIO})
$$

$$
\begin{array}{l}
P :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm \wedge c : (T)^- \\
a(c).P :: \emptyset \vdash a.?((T)^-);(a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm) \\
P_s :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm \quad \text{by } (\mathit{ContrIOInp}) \\
C_i :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^+ \wedge c : (T)^+ \\
b(c).C_i :: \emptyset \vdash b.?((T)^+);(a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm) \\
S_i :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm \quad \text{by } (\mathit{ContrIOInp}) \\
\mathbf{0} :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm \\
b\langle p\rangle :: \emptyset \vdash b.!((T)^\pm);(a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm) \\
b\langle p\rangle :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm \quad \mathit{by}\ (\mathit{ContrIOOut}) \\
b\langle p\rangle.b\langle p\rangle :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm \quad \mathit{Identical} \\
a\langle p\rangle.b\langle p\rangle.b\langle p\rangle :: \emptyset \vdash a.!((T)^\pm);(a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm) \\
a\langle p\rangle.b\langle p\rangle.b\langle p\rangle :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm \quad \mathit{by}\ (\mathit{ContrIOOut}) \\
(P_s \mid S_1 \mid S_2 \mid I) :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm \wedge p : (T)^\pm \\
\mathit{Sys} :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm
\end{array}
$$

Fig. 7. Sample derivation of I/O types

Proof. Immediate for (*InpIO*), (*OutIO*) and (*ContrIOInp*), and (*SubInp*). For the remaining ones we first show that (a) $\lceil T\rceil \models \lceil T\rceil^\circ$, and (b) $U <: V$ implies $\lceil V\rceil^\circ \models \lceil U\rceil^\circ$. (*ContrIOOut*) follows from (a) and (b), and (*SubOut*) from (b). ■

Interestingly, the subtyping relation induced by the logical semantics satisfy the syntactically defined relation $\leq$ in [19] (reading $\geq$ as $<:$, and apart from recursive types, for which one should add a coinduction rule). All of its typing rules may be shown admissible in the extension of $\mathbf{T}+\mathbf{IO}_{<:}$: we just need to verify that I/O types are classical and invariant (by inspection), sharing, and therefore that all the sharing typing rules **S** rules are applicable to them.

For an illustration, we borrow an example from [19]. A system composed by a printer P and two clients C_1 and C_2 is set up so that the printer is only allowed to read from the clients, while clients are only allowed to write to the printer. For readability, we tag bound names with their intended types.

$$
\begin{array}{ll}
\mathit{Sys} \triangleq (\boldsymbol{\nu} p : (T)^\pm)(P_s \mid S_1 \mid S_2 \mid I) & I \triangleq a\langle p\rangle.b\langle p\rangle.b\langle p\rangle \\
P_s \triangleq a(c : (T)^-).P & S_i \triangleq b(c : (T)^+).C_i
\end{array}
$$

We can then derive $\mathit{Sys} :: \emptyset \vdash a : ((T)^-)^\pm \wedge b : ((T)^+)^\pm$, as presented in the Figure 7. Interpreting the types as the intended logical predicates, by soundness, we conclude, for instance, that $P \models \texttt{rec}\ X.[n.!(\emptyset);]\mathsf{F} \wedge \Box X$. This means that the printer will never attempt to write on channel c.

5.3 Behavioral and Session Types

Various behavioral type disciplines for π-calculi have been proposed (*e.g.*, [13,14,10]), the intention being to discipline the sequence of interactions between processes, so that certain liveness and safety properties may be obtained. Particularly interesting are session types [12], that may be used to discipline

$$Session(x) \triangleq x.!(Op) \triangleright x.!(Int) \triangleright x.!(Int) \triangleright x.?(Int) \triangleright \emptyset$$

$\mathbf{0} :: \emptyset \vdash \emptyset$
$x(u).\mathbf{0} :: \emptyset \vdash x.!(Int) \triangleright \emptyset$
$x\langle 1\rangle.x\langle 2\rangle.x(u).\mathbf{0} :: \emptyset \vdash x.!(Int) \triangleright x.!(Int) \triangleright x.?(Int) \triangleright \emptyset$
$ClientBody(x) :: \emptyset \vdash Session(x)$
$(\boldsymbol{\nu} x)(a\langle x\rangle.ClientBody(x)) :: a.?(Session) \triangleright \emptyset \vdash \emptyset$ *by* (*Out-P-Left*)

$\mathbf{0} :: \emptyset \vdash \emptyset$
$y\langle v_1 + v_2\rangle.\mathbf{0} :: op : Op \mid v_1 : Int \mid v_2 : Int \mid y.?(Int) \triangleright \emptyset \vdash \emptyset$
$y(v_1).y(v_2).y\langle v_1 + v_2\rangle.\mathbf{0} :: op : Op \mid y.!(Int) \triangleright y.!(Int) \triangleright y.?(Int) \triangleright \emptyset \vdash \emptyset$
$ServerBody(y) :: Session(y) \vdash \emptyset$
$a(y).ServerBody(y) :: \emptyset \vdash a :?(Session) \triangleright \emptyset$ *by* (*In-Right*)

$Sys :: \emptyset \vdash \emptyset$ *by* (*Seq*)

Fig. 8. Sample derivation of session types

dialogue-like interactions between exactly two parties. At least certain forms of session types may be embedded in the generic type system **T** in a rather straightforward way, by combining generic types with simple types. The basic idea is to use judgments of the form $P :: S_i \vdash S_o \wedge \Gamma$ where S_i represents the (session) types of incoming (from the process environment) sessions, S_o the (session) types of outgoing (to the process environment) sessions, and Γ, a sharing type, declares the types of shared channels. Usually, one would expect Γ to be a conjunction of sharing types, for instance, simple types, or I/O types. On the other hand, the types S_i and S_o may be quite arbitrary, as far as one ensures $fn(\Gamma)\#fn(S_o)$ (need to combine processes using (*Shared-Cut*)). We illustrate with a simple example [10]: a server that offers a integer addition service, and its client.

$$\begin{aligned}
&Sys && \triangleq (\boldsymbol{\nu} a)(Client \mid Server)\\
&ClientBody(x) && \triangleq x\langle plus\rangle.x\langle 1\rangle.x\langle 2\rangle.x(u).\mathbf{0}\\
&ServerBody(x) && \triangleq x(op).x(v_1).x(v_2).x\langle v_1 + v_2\rangle.\mathbf{0}\\
&Client && \triangleq (\boldsymbol{\nu} x)(a\langle x\rangle.ClientBody(x))\\
&Server && \triangleq a(y).ServerBody(y)
\end{aligned}$$

A possible typing for the system *Sys* in $\mathbf{T} + \mathbf{ST}_{<:}$ is shown in Figure 8, where we assume the extension of the system with some pure value types (*Int*, *Op*), along predictable lines (cf. `nil`). Notice that no sharing types have been used, and channel a is used just once. However, channel a may be shared, even if it moves around "session" partners as resources, using the spatial modalities $a.?(Session)\triangleright$ and $a.!(Session)\triangleright$. However, we may set $a : (Session)\triangleright$, where

$$\lceil (T)\triangleright \rceil(n) \triangleq \texttt{rec}\ X.\emptyset \wedge [n.!(\lceil T\rceil)\triangleright]X \wedge [n.?(\lceil T\rceil)\triangleright]X \wedge \Box X$$

The intention is to let $(T)\triangleright$ be a "ownership-transfer" version of the simple type (T). We may check that the types $(T)\triangleright$ are classical, invariant, and sharing. We denote by **OT** the expected subtyping axioms for "ownership-transfer" simple types. Using just the axioms and rules of $\mathbf{T}+\mathbf{ST}_{<:} + \mathbf{OT}_{<:}$ we may then show an alternative typing for the system *Client* | *Server*, where the name a is free.

$$Client \mid Server :: \emptyset \vdash a : (Session)_{\triangleright}$$

Again, soundness of the obtained type system is obtained for "free", after one proves certain abstract properties (*e.g.*, sharing) of new type constructions.

6 Concluding Remarks

The original understanding of types as predicates has not always been a guiding principle in the design of types for process calculi, where a syntactical view seems to be dominant (an exception is [8], where a notion of semantic subtyping for names was developed). In this paper, we have developed a formal semantic approach to types in concurrency, based on an interpretation of types as spatial logic definable properties. The feasibility of the approach was demonstrated by the proposal of a generic type system, where many interesting notions of typing for mobile processes may be embedded just by introducing suitable subtyping relations, while modularly preserving soundness (Theorems 4.5 and 4.6). Thus, our approach seems to generalize other existing proposals to generic typing [13], that rely on more standard techniques. Some of the logical characterizations we have introduced allowed us to understand notions such as sharing and linearity [14] in types for concurrency in a rather abstract setting; it would be interesting to compare ours with other interpretations of sharing [17].

The framework proposed here may be generalized along several directions. Our development is not dependent on the structure of the underlying basic safety predicate, it would then be interesting to consider different basic properties (*e.g.*, security). Different notions of sharing might also be accommodated, if replication replaces recursion in the process calculus.

We believe that spatial logics provide a suitable metalanguage in which many type-like properties of interest may be formally expressed at an adequate level of abstraction, and that soundness proofs developed along the lines we have shown here are more modular and more intuitive than purely syntactic subject reduction style proofs. The representation of a type is essentially a process predicate that explicitly affirms of the subject the safety properties of interest. Our results suggest that these techniques may be used with some advantage over purely syntactic approaches to the semantics of typing, at least in some situations, in particular when traditional subject reduction techniques do not scale so to comfortably handle an increased complexity in global proof invariants, for example, due to the introduction of rich subtyping relations [3].

References

1. Caires, L.: Behavioral and Spatial Properties in a Logic for the Pi-Calculus. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, Springer, Heidelberg (2004)
2. Caires, L.: Logical Semantics of Types for Concurrency. Technical Report 2/07, Departamento de Informatica FCT/UNL (2007)

3. Caires, L.: Spatial-Behavioral Types, Distributed Services, and Resources. In: Montanari, U., Sanella, D. (eds.) TGC 2006 2nd Intl. Symp. on Trustworthy Global Computing. LNCS, Springer, Heidelberg (2007)
4. Caires, L., Cardelli, L.: A Spatial Logic for Concurrency (Part I). Information and Computation 186(2), 194–235 (2003)
5. Caires, L., Cardelli, L.: A Spatial Logic for Concurrency (Part II). Theoretical Computer Science 3(322), 517–565 (2004)
6. Caires, L., Vieira, H.: Extensionality of Spatial Observations in Distributed Systems. Electronic Notes in Theoretical Computer Science (2007)
7. Cardelli, L., Gordon, A.D.: Anytime, Anywhere. Modal Logics for Mobile Ambients. In: 27th ACM Symp. on Principles of Programming Languages, pp. 365–377. ACM Press, New York (2000)
8. Castagna, G., De Nicola, R., Varacca, D.: Semantic Subtyping for the π-Calculus. In: 20th IEEE Symposium on Logic in Computer Science (LICS 2005), pp. 92–101. IEEE Computer Society Press, Los Alamitos (2005)
9. Curry, H.B., Feys, R.: Combinatory Logic. North-Holland, Amsterdam (1958)
10. Gay, S.J., Hole, M.: Subtyping for Session Types in the Pi-Calculus. Acta Informatica 42(2-3), 191–225 (2005)
11. Hennessy, M., Milner, R.: Algebraic Laws for Nondeterminism and Concurrency. JACM 32(1), 137–161 (1985)
12. Honda, K., Vasconcelos, V.T., Kubo, M.: Language Primitives and Type Discipline for Structured Communication-Based Programming. In: Hankin, C. (ed.) ESOP 1998 and ETAPS 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)
13. Igarashi, A., Kobayashi, N.: A Generic Type System for the Pi-Calculus. In: POPL 2001: 28th ACM Symp. on Principles of Programming Languages, ACM Press, New York (2001)
14. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the Pi-Calculus. ACM Trans. Program. Lang. Syst. 21(5), 914–947 (1999)
15. Milner, R.: The Polyadic π-Calculus: A Tutorial. Technical Report 180, University of Edinburgh LFCS (1991)
16. Milner, R., Parrow, J., Walker, D.: Modal Logics for Mobile Processes. Theoretical Computer Science 114, 149–171 (1993)
17. O'Hearn, P., Pym, D.: The Logic of Bunched Implications. The Bulletin of Symbolic Logic 5(2), 215–243 (1999)
18. O'Hearn, P.W.: Resources, Concurrency and Local Reasoning. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 49–67. Springer, Heidelberg (2004)
19. Pierce, B.C., Sangiorgi, D.: Typing and Subtyping for Mobile Processes. Mathematical Structures in Computer Science 6(5), 409–453 (1996)
20. Reynolds, J.C.: Separation Logic: A Logic for Shared Mutable Data Structures. In: Third Annual Symposium on Logic in Computer Science, Copenhagen, Denmark, IEEE Computer Society Press, Los Alamitos (2002)
21. Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
22. Tait, W.: Intensional Interpretations of Functionals of Finite Type. J. Symbolic Logic 32(2), 198–212 (1967)
23. Wright, A.K., Felleisen, M.: A Syntactic Approach to Type Soundness. Inf. Comput. 115(1), 38–94 (1994)

Deriving Bisimulation Congruences with Borrowed Contexts*
(Abstract)

Barbara König

Abteilung für Informatik und Angewandte Kognitionswissenschaft,
Universität Duisburg-Essen, Germany
barbara_koenig@uni-due.de

In the last few years the problem of deriving labelled transitions and bisimulation congruences from unlabelled reaction or rewriting rules has received great attention. This line of research was motivated by the theory of bisimulation congruences for process calculi, such as the π-calculus [19,14]. A bisimilarity defined on unlabelled reduction rules is usually not a congruence, that is, it is not closed under the operators of the process calculus. Congruence is a desirable property since it allows one to replace a subsystem with an equivalent one without changing the behaviour of the overall system and futhermore helps to make bisimilarity proofs modular.

Previous solutions have been to either require that two processes are related if and only if they are bisimilar under all possible contexts [15] or to derive a labelled transition system manually. Since the first solution needs quantification over all possible contexts, proofs of bisimilarity can be very complicated. In the second solution, proofs tend to be much easier, but it is necessary to show that the labelled variant of the transition system is equivalent to the unlabelled variant.

So the idea which was formulated in the papers of Leifer/Milner [12,13], Sewell [22] and Sassone/Sobociński [20] is to automatically derive a labelled transition system such that the resulting bisimilarity is a congruence. A central concept of this approach is to formalize the notion of minimal context which enables a process to reduce. Consider, for example, the CCS process $a.P$. It reduces when put into the contexts $_ \mid \bar{a}.Q$ and $_ \mid \bar{a}.Q \mid b.R$, but one is interested only in the first context, since it is in some sense smaller than the second one. This yields the labelled transition

$$a.P \stackrel{_\mid \bar{a}.Q}{\longrightarrow} P \mid Q,$$

saying that $a.P$ put into this contexts reacts and reduces to $P \mid Q$. Using all possible contexts as labels would also result in a (coarser) bisimulation congruence, but we do not gain anything compared to quantification over all contexts (for a more detailed study of this congruence see [3]).

In [12,13] the notion of "minimal context" is formalized as the categorical concept of relative pushout (RPO) respectively idem pushout (IPO). This notion has

* Research partially supported by the DFG project SANDS and CRUI/DAAD Vigoni "Models based on Graph Transformation Systems: Analysis and Verification".

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 36–39, 2007.
© Springer-Verlag Berlin Heidelberg 2007

also been applied to bigraphs [9]. However, the theory is complicated by the fact that one can not work with isomorphism classes of graphs, since in this case the category under consideration would not possess all necessary relative pushouts. Thus one is forced to give unique names to all edges and nodes in a graph, i.e., to add support to a category, and to either work in a precategory or to construct a suitable category starting from such a precategory. A different approach, presented by Sassone and Sobociński [20,21], that does not require the notion of support, is to construct relative pushouts (so-called GRPOs) in a 2-categorical setting. This work is based on the notion of adhesive categories [11].

We will also use adhesive categories and work with adhesive rewriting systems, which can be seen as a generalization of graph rewriting systems [18], a framework which allows to model dynamic and concurrent systems consisting of interconnected components in a natural and intuitive way. Many process calculi such as the π-calculus [8,16,10] and the ambient calculus [7] can be translated into this framework. We are specifically interested in the double-pushout (DPO) approach to rewriting [4,5]. Adding support, as explained earlier, would be possible in theory, but contradicts the philosophy behind graph rewriting where graphs (or more generally objects) are considered only up to isomorphism. Compared to other approaches, in which the derivation of labels is a somewhat complex task, our approach is rather straightforward and simple.

The approach presented here [6] is motivated by the work of Leifer and Milner and other contributions to this area, but does not directly rely on their theory. Instead we present an uncomplicated way of deriving minimal contexts—we call them borrowed contexts—which smoothly extends the DPO approach and which has a very constructive nature. The only categorical concepts that are needed are pushouts and pullbacks. The main difference to previous approaches is that in our case graphs (more generally: the structures which are being rewritten) are objects and not arrows of the category under consideration. Our arrows instead are (graph) morphisms which provide the necessary tracking information for nodes and edges which, in the case of graphs as arrows, can—as it turned out—only be provided by either adding support to a category or by working in a 2-categorical framework.

Our main result states that bisimilarity defined on labelled transitions with borrowed contexts is indeed a congruence relation. Furthermore we introduce an up-to-context proof technique and discuss the mechanization of bisimulation proofs (see also [17]).

We will compare with related work and present an application of our approach to the derivation of bisimulation congruences for CCS [2]. Finally, we give an outlook to future plans where we are working towards an inductive definition, in SOS style, of the labelled transition system associated to the reduction rules (see also [1]).

Acknowledgements. This is joint work with Hartmut Ehrig, Paolo Baldan, Filippo Bonchi, Fabio Gadducci, Guilherme Rangel and Ugo Montanari. I want to thank my coauthors for many interesting and stimulating discussions on the topic.

References

1. Baldan, P., Ehrig, H., König, B.: Composition and decomposition of DPO transformations with borrowed context. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 153–167. Springer, Heidelberg (2006)
2. Bonchi, F., Gadducci, F., König, B.: Process bisimulation via a graphical encoding. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 168–183. Springer, Heidelberg (2006)
3. Bonchi, F., König, B., Montanari, U.: Saturated semantics for reactive systems. In: Proc. of LICS '06, IEEE Computer Society Press, Los Alamitos (2006)
4. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation—part I: Basic concepts and double pushout approach. In: Rozenberg, G., (ed.) Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1: Foundations, ch. 3. World Scientific (1997)
5. Ehrig, H., Pfender, M., Schneider, H.: Graph grammars: An algebraic approach. In: Proc. 14th IEEE Symp. on Switching and Automata Theory, pp. 167–180. IEEE Computer Society Press, Los Alamitos (1973)
6. Ehrig, H., König, B.: Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. Mathematical Structures in Computer Science 16(6), 1133–1163 (2006)
7. Gadducci, F., Montanari, U.: A concurrent graph semantics for mobile ambients. In: Brookes, S., Mislove, M. (eds.) Proceedings of the 17th MFPS. Electronic Notes in Computer Science, vol. 45, Elsevier Science, Amsterdam (2001)
8. Gadducci, F., Montanari, U.: Comparing logics for rewriting: Rewriting logic, action calculi and tile logic. Theoretical Computer Science 285(2), 319–358 (2002)
9. Jensen, O.H., Milner, R.: Bigraphs and transitions. In: Proc. of POPL 2003, pp. 38–49. ACM Press, New York (2003)
10. König, B.: A graph rewriting semantics for the polyadic π-calculus. In: Proc. of GT-VMT '00 (Workshop on Graph Transformation and Visual Modeling Techniques), pp. 451–458. Carleton Scientific (2000)
11. Lack, S., Sobociński, P.: Adhesive and quasiadhesive categories. RAIRO – Theoretical Informatics and Applications 39(3) (2005)
12. Leifer, J.J.: Operational congruences for reactive systems. PhD thesis, University of Cambridge Computer Laboratory (September 2001)
13. Leifer, J.J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, Springer, Heidelberg (2000)
14. Milner, R.: Communicating and Mobile Systems: the π-Calculus. Cambridge University Press, Cambridge (1999)
15. Milner, R., Sangiorgi, D.: Barbed bisimulation. In: Kuich, W. (ed.) Automata, Languages and Programming. LNCS, vol. 623, Springer, Heidelberg (1992)
16. Montanari, U., Pistore, M.: Concurrent semantics for the π-calculus. Electronic Notes in Theoretical Computer Science 1 (1995)
17. Rangel, G., König, B., Ehrig, H.: Bisimulation verification for the DPO approach with borrowed contexts. In: Proc. of GT-VMT '07 (Workshop on Graph Transformation and Visual Modeling Techniques), Electronic Communications of the EASST (to appear)

18. Rozenberg, G.(ed.): Handbook of Graph Grammars and Computing by Graph Transformation, Foundations, vol. 1 World Scientific (1997)
19. Sangiorgi, D., Walker, D.: The π-calculus—A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
20. Sassone, V., Sobociński, P.: Deriving bisimulation congruences: 2-categories vs precategories. In: Gordon, A.D. (ed.) ETAPS 2003 and FOSSACS 2003. LNCS, vol. 2620, pp. 409–424. Springer, Heidelberg (2003)
21. Sassone, V., Sobociński, P.: Reactive systems over cospans. In: Proc. of LICS '05, pp. 311–320. IEEE Computer Society Press, Los Alamitos (2005)
22. Sewell, P.: From rewrite rules to bisimulation congruences. Theoretical Computer Science 274(1–2), 183–230 (2002)

Symmetry and Concurrency
(Extended Abstract)

Glynn Winskel

University of Cambridge Computer Laboratory, England
Glynn.Winskel@cl.cam.ac.uk
http://www.cl.cam.ac.uk/users/gw104

Abstract. A category of event structures with symmetry is introduced and its categorical properties investigated. Applications to the event-structure semantics of higher order processes, nondeterministic dataflow and the unfolding of higher-dimensional automata and Petri nets with multiple tokens are indicated.

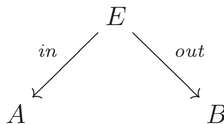
Keywords: Event structures, symmetry, pseudo monads, spans, higher order processes, nondeterministic dataflow, unfolding, Petri nets, higher dimensional automata.

1 Introduction

In the paper introducing event structures [15] a 'curious mismatch' was noted. There event structures represent domains, so types. But they also represent processes which belong to a type. How are we to reconcile these two views?

One answer has arisen in recent work under the banner of 'domain theory for concurrency' (see [17] for a summary). This slogan stands for an attempt to push the methodology of domain theory and denotational semantics into the areas of interactive/concurrent/distributed computation, where presently more syntactic, operational or more informal methodologies prevail. Certain generalized relations (profunctors [4]) play a strong unifying role and it was discovered that in several contexts that they could be represented in a more informative operational way by spans of event structures [16,28,19].

A span of event structures is typically of the form



where *in* and *out* are maps of event structures—the maps are not necessarily of the same kind. The event structure E represents a process computing from an input type, represented by the event structure A, to output type represented by B. A span with no input amounts to just a single map $E \xrightarrow{out} B$ which we can read

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 40–64, 2007.
© Springer-Verlag Berlin Heidelberg 2007

as expressing that the process E has type B. So spans are a way to reconcile the double role that event structures can take, as processes and as types.

Of course spans should compose. So one would like systematic ways to vary the *in* and *out* maps of spans which ensure they do. One way is to derive the maps by a Kleisli construction from monads on a fundamental category of event structures. With respect to suitable monads S and T satisfying a suitable distributivity law, one can form a bicategory of more general spans

$$S(A) \leftarrow E \rightarrow T(B)\ .$$

It becomes important that event structures are able to support a reasonable repertoire of monads, including monads which produce multiple, essentially similar, copies of an event structure. For this the introduction of symmetry seems essential.[1]

In fact, there are several reasons for introducing symmetry to event structures and related models:

- It's there—at least informally. Symmetry often plays a role in the analysis of distributed algorithms. In particular, symmetry has always been present at least informally in the model of strand spaces, and has recently been exploited in exploring their behaviour [8], and was used to understand their expressivity [6]. Strand spaces are forms of event structures used in the analysis of security protocols. They comprise a collection of strands of input and output events, possibly with the generation of fresh values. Most often there are collections of strands which are essentially indistinguishable and can be permuted one for another without changing the strand space's behaviour.
- To obtain categorical characterizations of unfoldings of higher-dimensional automata [7], and more specifically Petri nets in which places may hold with multiplicity greater than one. There are well-known ways to unfold such general nets; for example by distinguishing the tokens through 'colours,' splitting the places and events accordingly and reducing the problem to the unfolding in [15]. But the folding maps are not unique (w.r.t. an obvious cofreeness property). They are however unique 'up to symmetry.'
- Event structures are sometimes criticized for not being abstract enough. One precise way in which this manifests itself is that the category of event structures does not support monads and comonads of the kind discovered for more general presheaf models [4]. The computation paths of an event structure, its configurations, are ordered by inclusion. In contrast the paths of presheaf models can be related more generally by maps. Some (co)monads used for presheaf models allow the explicit copying of processes and produce a proper category of paths even when starting with a partial order of paths—this arises because of the similarity of one copy of a process with another.

[1] Symmetry was introduced into game semantics specifically to support a 'copying' comonad [1].

The last point is especially pertinent to the versatility of spans of event structures. This paper presents a definition of a symmetry on an event structure. Roughly a symmetry will express the similarity of *finite* behaviours of an event structure. The introduction of symmetries to event structures will, in effect, put the structure of a category on their finite configurations, and so broaden the structure of computation paths event structures can represent. The ensuing category of event structures with symmetries will support a much richer class of (pseudo) monads, from which we can then obtain more general kinds of span. The category of event structures with symmetry with rigid maps emerges as fundamental; other maps on event structures can be obtained by a Kleisli construction or as instances of general spans starting from rigid maps.

Several applications, to be developed in future work, are outlined in Section 6:

– *Event types:* One reason why so-called 'interleaving' models for concurrency have gained prevalence is that they support definitions by cases on the initial actions processes can do; another is that they readily support higher-order processes. Analogous facilities are lacking, at least in any reasonable generality, in 'true-concurrency' models—models like Petri nets and event structures, in which causal dependence and independence are represented explicitly. It is sketched how processes can be associated with 'event types' which specify the kinds of events they can do, and how event types can support definitions by cases on events. There are difficulties and much more needs to be done. But the examples do demonstrate the key role that symmetry and the copying of processes could play in obtaining flexible event types and event-based definitions.
– *Nondeterministic dataflow and affine-HOPLA:* 'Stable' spans of event structures, a direct generalisation of Berry's stable functions [2], have been used to give semantics to nondeterministic dataflow [19] and the higher-order process language affine-HOPLA [16]. Stable spans can be obtained as instances of general spans. The realization of the 'demand' maps used there as a Kleisli construction on rigid maps provides a striking example of the power of symmetry.
– *Unfoldings:* One obvious application is to the unfolding of a general Petri net to an event structure with symmetry; the symmetry reflects that present in the original net through the interchangeability of tokens. Another related issue is the unfolding of higher-dimensional automata, where identifications of edges are reflected in the symmetry of the events to which they unfold.

This presentation concentrates on the model of (prime) event structures. But the same techniques apply to many other models, including more algorithmically-amenable models such as Petri nets or versions of transition systems. The model of stable families [23] plays a significant, if hidden role, in the proofs—they deserve a more forthright treatment in future. The work reported is based on an extended article which appears in the Gordon Plotkin Festschrift [29], where further details may be found. As well as streamlining the presentation, I have taken the opportunity here to make corrections (chiefly in the unfinished work on 'Event types', Section 6.2), additions (on 'Event types', Section 6.2 and on

'Unfoldings,' Section 6.4), and replaced the condition of countability on event structures by the weaker condition of 'consistent-countable,' which suffices for the proofs in [29] and allows extra results, *e.g.* in Sections 6.2 and 6.4.

2 Event Structures

Event structures [15,22,25,26] are a model of computational processes. They represent a process as a set of event occurrences with relations to express how events causally depend on others, or exclude other events from occurring. In one of their simpler forms they consist of a set of events on which there is a consistency relation expressing when events can occur together in a history and a partial order of causal dependency—writing $e' \leq e$ if the occurrence of e depends on the previous occurrence of e'.

An *event structure* comprises $(E, \mathrm{Con}, \leq)$, consisting of a set E, of *events* which are partially ordered by $\leq$, the *causal dependency relation*, and a *consistency relation* Con consisting of finite subsets of E, which satisfy

$$\begin{aligned}
&\{e' \mid e' \leq e\} \text{ is finite for all } e \in E,\\
&\{e\} \in \mathrm{Con} \text{ for all } e \in E,\\
&Y \subseteq X \in \mathrm{Con} \Rightarrow Y \in \mathrm{Con}, \quad \text{and}\\
&X \in \mathrm{Con} \;\&\; e \leq e' \in X \Rightarrow X \cup \{e\} \in \mathrm{Con}.
\end{aligned}$$

Here we insist that an event structure is *consistent-countable*,[2] *i.e.* that there is a function χ from its events to the natural numbers ω such that $\{e_1, e_2\} \in \mathrm{Con}$ and $\chi(e_1) = \chi(e_2)$ implies $e_1 = e_2$.

The events are to be thought of as event occurrences; in any history an event is to appear at most once. A configuration is a set of events which have occurred by some stage in a process. According to our understanding of the consistency predicate and causal dependency relations a configuration should be consistent and such that if an event appears in a configuration then so do all the events on which it causally depends. Here we restrict attention to finite configurations.

The *(finite) configurations*, $\mathcal{C}^o(E)$, of an event structure E consist of those finite subsets $x \subseteq E$ which are

Consistent: $x \in \mathrm{Con}$ and
Down-closed: $\forall e, e'.\ e' \leq e \in x \Rightarrow e' \in x$.

The configurations of an event structure are ordered by inclusion, where $x \subseteq x'$, *i.e.* x is a sub-configuration of x', means that x is a sub-history of x'. Note that an individual configuration inherits an order of causal dependency on its events from the event structure so that the history of a process is captured through a partial order of events. For an event e the set $\{e' \in E \mid e' \leq e\}$ is a configuration describing the whole causal history of the event e.

---

[2] The condition of consistent-countability replaces the stronger condition of countability of event structures in [29]. Proofs there still go through with the weaker condition, while the extra generality makes new results possible—see Sections 6.2, 6.4.

When the consistency relation is determined by the pairwise consistency of events we can replace it by a binary relation or, as is more usual, by a complementary binary conflict relation on events. It can be awkward to describe operations such as certain parallel compositions directly on the simple event structures here, because an event determines its whole causal history. One closely related and more versatile model is that of stable families, described in Appendix B.

Let E and E' be event structures. A *partial map* of event structures $f : E \rightharpoonup E'$ is a partial function on events $f : E \rightharpoonup E'$ such that for all configurations x of E its direct image fx is a configuration of E' for which

$$\text{if } e_1, e_2 \in x \text{ and } f(e_1) = f(e_2) \in E', \text{ then } e_1 = e_2.$$

The map expresses how the occurrence of an event e in E induces the coincident occurrence of the event $f(e)$ in E' whenever it is defined. The partial function f respects the instantaneous nature of events: two distinct event occurrences which are consistent with each other cannot both coincide with the occurrence of a common event in the image. Maps of event structures compose as partial functions.

We will say the map is *total* iff the function f is total. Notice that for a total map f the condition on maps now says it is *locally injective*, in the sense that w.r.t. any configuration x of the domain the restriction of f to a function from x is injective; the restriction of f to a function from x to fx is thus bijective.

We say the map f is *rigid* iff it is total and for all $x \in \mathcal{C}^o(E)$ and $y \in \mathcal{C}^o(E')$

$$y \subseteq f(x) \Rightarrow \exists z \in \mathcal{C}^o(E).\ z \subseteq x \text{ and } fz = y\ .$$

(The configuration z is necessarily unique.)

A rigid map of event structures preserves the causal dependency relation "rigidly," so that the causal dependency relation on the image fx is a copy of that on a configuration x of E; this is not so for general maps where x may be augmented with extra causal dependency over that on fx. (Special forms of rigid maps appeared as *rigid embeddings* in Kahn and Plotkin's work on concrete domains [12].)

Here we concentrate on the category of event structures with total maps.

Definition 1. *Write $\mathcal{E}$ for the category of event structures with total maps. (In future, unless further specified, by a map of event structures we will mean a total map.)*

Proposition 1. *The category $\mathcal{E}$ of event structures with total maps of event structures has (binary) products and pullbacks (though no terminal object).*

In defining symmetries on event structures we will make use of open maps w.r.t. finite elementary event structures (*i.e.* finite event structures in which all subsets of events are consistent) as the particular choice of paths [11].

Say a map $h : A \to B$, between event structures A and B, is *open* iff for all maps $j : p \to q$ between finite elementary event structures, any commuting square

$$\begin{array}{ccc} p & \xrightarrow{x} & A \\ {\scriptstyle j}\downarrow & & \downarrow{\scriptstyle h} \\ q & \xrightarrow[y]{} & B \end{array}$$

can be split into two commuting triangles

$$\begin{array}{ccc} p & \xrightarrow{x} & A \\ {\scriptstyle j}\downarrow & \overset{z}{\nearrow} & \downarrow{\scriptstyle h} \\ q & \xrightarrow[y]{} & B. \end{array}$$

That the square commutes means that the path $h \circ x$ in B can be extended via j to a path y in B. That the two triangles commute means that the path x can be extended via j to a path z in A which matches y.

Open maps are a generalisation of functional bisimulations, known from transition systems.

Proposition 2. *A map $h : A \to B$ of event structures is open iff h is rigid and satisfies:* $\forall x \in \mathcal{C}^o(A), y' \in \mathcal{C}^o(B).\ hx \subseteq y' \Rightarrow \exists x' \in \mathcal{C}^o(E).\ x \subseteq x' \ \&\ hx' = y'$.

3 Event Structures with Symmetry

We shall present a general definition of symmetry, concentrating on the category $\mathcal{E}$ of event structures with total maps. This category has (binary) products and pullbacks (though no terminal object) and supports a notion of open map. For the definition of symmetry we are about to give this is all we require.

A symmetry on an event structure should specify which events are similar in such a way that similar events have similar pasts and futures. This is captured, somewhat abstractly, by the following definition.

Definition 2. *An* event structure with symmetry (E, l, r) *comprises an event structure E together with* open *maps $l : S \to E$ and $r : S \to E$ from a common event structure S such that the map $\langle l, r\rangle : S \to E \times E$ is an equivalence relation (i.e., the map $\langle l, r\rangle$ is monic—equivalently, l, r are jointly monic—and satisfies the standard diagramatic properties of reflexivity, symmetry and transitivity [10]. See Appendix A).*

A bisimulation is given by a span of open maps [11], in the case of the above definition by the pair of open maps l and r. So the definition expresses a symmetry on an event structure as a bisimulation equivalence. The definition has the advantage of being abstract in that it readily makes sense for any category with binary products and pullbacks for which there is a sensible choice of paths in order to define open maps. It is sensible for the categories of event structures with rigid and partial maps, for stable families, transition systems, trace

languages and Petri nets [21], because these categories also have products, pullbacks and open maps; both categories of event structures with rigid and partial maps would have the same class of open maps and so lead to precisely the same event structures with symmetry as objects. We shall mainly concentrate on the category with total maps to connect directly with the particular examples we shall treat here.[3]

For the specific model of event structures there is an alternative way to present a symmetry. We can express a symmetry $l, r : S \to E$ on an event structure E equivalently as a relation of similarity between its finite configurations. More precisely, two finite configurations x, y of E are related by a bijection $\theta_z =_{\text{def}} \{(l(s), r(s)) \mid s \in z\}$ if they arise as images $x = l\,z$ and $y = r\,z$ of a common finite configuration z of S; because l and r are locally injective θ_z is a bijection between x and y. Because l and r are rigid the bijection is an order isomorphism between x and y with the order of causal dependency inherited from E. In this way a symmetry on E will determine an *isomorphism family* expressing when and how two finite configurations are similar, or symmetric, in the sense that one can replace the other. As expected, such similarity forms an equivalence relation, and if two configurations are similar then so are their pasts (restrictions to subconfigurations) and futures (extensions to larger configurations).

Definition 3. *An* isomorphism family *of an event structure E consists of a family $\mathbb{S}$ of bijections*

$$\theta : x \cong y$$

between pairs of finite configurations of E such that:
(i) the identities $\mathrm{id}_x : x \cong x$ are in $\mathbb{S}$ for all $x \in \mathcal{C}^o(E)$; if $\theta : x \cong y$ is in $\mathbb{S}$, then so is the inverse $\theta^{-1} : y \cong x$; and if $\theta : x \cong y$ and $\varphi : y \cong z$ are in $\mathbb{S}$, then so is their composition $\varphi \circ \theta : x \cong z$.
(ii) for $\theta : x \cong y$ in $\mathbb{S}$ whenever $x' \subseteq x$ with $x' \in \mathcal{C}^o(E)$, then there is a (necessarily unique) $y' \in \mathcal{C}^o(E)$ with $y' \subseteq y$ such that the restriction of θ to $\theta' : x' \cong y'$ is in $\mathbb{S}$.
(iii) for $\theta : x \cong y$ in $\mathbb{S}$ whenever $x \subseteq x'$ for $x' \in \mathcal{C}^o(E)$, then there is an extension of θ to $\theta' : x' \cong y'$ in $\mathbb{S}$ for some (not necessarily unique) $y' \in \mathcal{C}^o(E)$ with $y \subseteq y'$.
[Note that (i) implies that the converse forms of (ii) and (iii) also hold. Note too that (ii) implies that the bijections in the family $\mathbb{S}$ respect the partial order of causal dependency on configurations inherited from E; the bijections in an isomorphism family are isomorphisms between the configurations regarded as elementary event structures.]

Theorem 1. *Let E be an event structure.*
(i) A symmetry $l, r : S \to E$ determines an isomorphism family $\mathbb{S}$: defining $\theta_z = \{(l(s), r(s)) \mid s \in z\}$ for z a finite configuration of S, yields a bijection $\theta_z : l\,z \cong r\,z$; the family $\mathbb{S}$ consisting of all bijections $\theta_z : l\,z \cong r\,z$, for z a finite configuration of S.

[3] There is a strong case for regarding rigid maps as *the* fundamental maps of event structures, in that other maps on event structures can then ultimately be obtained as Kleisli maps w.r.t. suitable pseudo monads once we have introduced symmetry.

(ii) An isomorphism family $\mathbb{S}$ *of* E *determines a symmetry* $l, r : S \to E$*: the family* $\mathbb{S}$ *forms a stable family; the event structure* S *is obtained as* $\Pr(\mathbb{S})$ *for which the events are primes* $[(e_1, e_2)]_\theta$ *for* θ *in* $\mathbb{S}$ *and* $(e_1, e_2) \in \theta$*; the maps* l *and* r *send a prime* $[(e_1, e_2)]_\theta$ *to* e_1 *and* e_2 *respectively.*

The operations of (i) and (ii) are mutually inverse (regarding relations as subobjects).

Through the addition of symmetry event structures can represent a much richer class of 'path categories' [4] than mere partial orders. The finite configurations of an event structure with symmetry can be extended by inclusion or rearranged bijectively under an isomorphism allowed by the symmetry. In this way an event structure with symmetry determines, in general, a *category* of finite configurations with maps obtained by repeatedly composing the inclusions and allowed isomorphisms. By property (ii) in Definition 3 any such map factors uniquely as an isomorphism of the symmetry followed by an inclusion. While by property (iii) any such map factors (not necessarily uniquely) as an inclusion followed by an isomorphism of the symmetry.

Example 1. Any event structure E can be identified with the event structure with the identity symmetry $(E, \mathrm{id}_E, \mathrm{id}_E)$. Its isomorphism family consists of all identities $\mathrm{id}_x : x \cong x$ on finite configurations $x \in \mathcal{C}^o(E)$.

Example 2. Identify the natural numbers ω with the event structure with events ω, trivial causal dependency given by the identity relation and in which all finite subsets of events are in the consistency relation. Define S to be the product of event structures $\omega \times \omega$ in $\mathcal{E}$; the product comprises events all pairs $(i, j) \in \omega \times \omega$ with trivlal causal dependency, and consistency relation consisting of all finite subsets of $\omega \times \omega$ which are bijective (so we take two distinct pairs (i, j) and (i', j') to be in conflict iff $i = i'$ or $j = j'$.) Define l and r to be the projections $l : S \to E$ and $r : S \to E$. Then $\varpi =_{\mathrm{def}} (\omega, l, r)$ forms an event structure with symmetry. The corresponding isomorphism family in this case coincides with all finite bijections between finite subsets of ω. Any finite subset of events of ϖ is similar to any other. Of course, an analogous construction works for any countable, possibly finite, set.

Example 3. Let $E = (E, l : S \to E, r : S \to E)$ be an event structure with symmetry. Define an event structure with symmetry $!E = (E_!, l_! : S_! \to E_!, r_! : S_! \to E_!)$ comprising ω similar copies of E as follows. The event structure $E_!$ has the set of events $\omega \times E$ with causal dependency

$$(i, e) \leq_! (i', e') \text{ iff } i = i' \ \& \ e \leq_E e'$$

and consistency relation

$$C \in \mathrm{Con}_! \text{ iff } C \text{ is finite } \& \ \forall i \in \omega. \ \{e \mid (i, e) \in C\} \in \mathrm{Con}_E \ .$$

The symmetry $S_!$ has events $\omega \times \omega \times S$ with causal dependency

$$(i, j, s) \leq_{S_!} (i', j', s') \text{ iff } i = i' \ \& \ j = j' \ \& \ s \leq_S s' \ .$$

A finite subset $C \subseteq S_!$ is in the consistency relation $\mathrm{Con}_{S_!}$ iff

$$\{(i,j) \mid \exists s.\ (i,j,s) \in C\} \text{ is bijective } \ \&\ \forall i,j \in \omega.\ \{s \mid (i,j,s) \in C\} \in \mathrm{Con}_S\ .$$

Define $l_!(i,j,s) = (i, l(s))$ and $r_!(i,j,s) = (j, r(s))$ for $i,j \in \omega$, $s \in S$.

The finite configurations of $E_!$ correspond to tuples (or indexed families) $\langle x_i \rangle_{i \in I}$ of configurations $x_i \in \mathcal{C}^o(E)$ indexed by $i \in I$, where I is a finite subset of ω. With this view of the configurations of $E_!$, the isomorphism family corresponding to $S_!$ specifies isomorphisms between tuples

$$(\sigma, \langle \theta_i \rangle_{i \in I}) : \langle x_i \rangle_{i \in I} \cong \langle y_j \rangle_{j \in J}$$

consisting of a bijection between indices $\sigma : I \cong J$ together with $\theta_i : x_i \cong y_{\sigma(i)}$ from the isomorphism family of S, for all $i \in I$.

The event structure with symmetry ϖ reappears as the special case $!1$, where 1 is the event structure with a single event.

We conclude this section with a general method for constructing symmetries. Just as there is a least symmetry on an event structure, *viz.* the identity symmetry, so is there a greatest. Moreover any bisimulation on an event structure generates a symmetry on it. We take a *bisimulation* on an event structure A to be a pair of open maps $l, r : R \to A$ from an event structure R for which $\langle l, r \rangle$ is monic. (In general we might specify a bisimulation on an event structure just by a pair of open maps from a common event structure, and not insist that the pair is monic. But here, no real generality is lost as such a pair of open maps on event structures will always factor through its image, a bisimulation with monicity.) In fact, the proof proceeds most easily by first establishing an analogous property for isomorphism families, a property which depends on the notion of a *bisimulation family*, defined to be a family of bijections between finite configurations of A which satisfy (ii) and (iii) in Definition 3.

Proposition 3. *Let A be an event structure.*
(i) For any bisimulation family $\mathcal{R}$ on A there is a least isomorphism family $\mathbb{S}$ for which $\mathcal{R} \subseteq \mathbb{S}$.
(ii) For any bisimulation $\langle l_0, r_0 \rangle : R \to A$ there is a least symmetry $\langle l, r \rangle : S \to A$ (understood as a subobject) for which R is a subobject of S. There is a greatest symmetry on A (which coincides with the greatest bisimulation on A).

4 Maps Preserving Symmetry

Maps between event structures with symmetry are defined as maps between event structures which preserve symmetry. Let (A, l_A, r_A) and (B, l_B, r_B) be event structures with symmetry. A map $f : (A, l_A, r_A) \to (B, l_B, r_B)$ is a map of event structures $f : A \to B$ such that there is a (necessarily unique) map of event structures $h : S_A \to S_B$ ensuring

$$\langle l_B, r_B \rangle \circ h = (f \times f) \circ \langle l_A, r_A \rangle\ .$$

Maps between event structures with symmetry compose as maps of event structures and share the same identity maps.

Definition 4. *We define $\mathcal{SE}$ to be category of event structures with symmetry.*

We can characterize when maps of event structures preserve symmetry in terms of isomorphism families. A map preserving symmetry should behave as a functor both w.r.t. the inclusion between finite configurations and the isomorphisms of the symmetry.

Proposition 4. *A map of event structures $f : A \to B$ is a map $f : (A, l_A, r_A) \to (B, l_B, r_B)$ of event structures with symmetry iff whenever $\theta : x \cong y$ is in the isomorphism family of A then $f\theta : f\,x \cong f\,y$ is in the isomorphism family of B, where $f\theta =_{\mathrm{def}} \{(f(e_1), f(e_2)) \mid (e_1, e_2) \in \theta\}$.*

We explore properties of the category $\mathcal{SE}$. It is more fully described as a category enriched in the category of equivalence relations and so, because equivalence relations are a degenerate form of category, as a 2-category in which the 2-cells are instances of the equivalence $\sim$. This view informs the constructions in $\mathcal{SE}$ which are often very simple examples of the (pseudo- and bi-) constructions of 2-categories.

Definition 5. *Let $f, g : (A, l_A, r_A) \to (B, l_B, r_B)$ be maps of event structures with symmetry between (A, l_A, r_A) and (B, l_B, r_B). Define $f \sim g$ iff there is a (necessarily unique) map of event structures $h : A \to S_B$ such that*

$$\langle f, g\rangle = \langle l_B, r_B\rangle \circ h \ .$$

Straightforward diagrammatic proofs show:

Proposition 5. *The relation $\sim$ is an equivalence relation on maps $\mathcal{SE}(A, B)$ between event structures with symmetry A and B. The relation $\sim$ respects composition in the sense that if $f \sim g$ then $h \circ f \circ k \sim h \circ g \circ k$, for composable maps h and k.*

The category $\mathcal{SE}$ is enriched in the category of equivalence relations (comprising equivalence relations with functions which preserve the equivalence).

We can characterize the equivalence of maps between event structures with symmetry in terms of isomorphism families which makes apparent how $\sim$ is an instance of natural isomorphism between functors.

Proposition 6. *Let $f, g : (A, l_A, r_A) \to (B, l_B, r_B)$ be maps of event structures with symmetry. Then, $f \sim g$ iff $\theta_x : f\,x \cong g\,x$ is in the isomorphism family of (B, l_B, r_B) for all $x \in \mathcal{C}^o(A)$, where $\theta_x =_{\mathrm{def}} \{(f(a), g(a)) \mid a \in x\}$.*

Equivalence on maps yields an equivalence on objects:

Definition 6. *Let A and B be event structures with symmetry. An* equivalence *from A to B is a pair of maps $f : A \to B$ and $g : B \to A$ such that $f \circ g \sim \mathrm{id}_B$ and $g \circ f \sim \mathrm{id}_A$; then we say A and B are equivalent and write $A \simeq B$.*

The category $\mathcal{SE}$ has products.

Theorem 2. *Let* (A, l_A, r_A) *and* (B, l_B, r_B) *be event structures with symmetry. Their product in* $\mathcal{SE}$ *is given by* $(A \times B, l_A \times l_B, r_A \times r_B)$*, based on the product* $A \times B$ *of their underlying event structures in* $\mathcal{E}$*, and sharing the same projections,* $\pi_1 : A \times B \to A$ *and* $\pi_2 : A \times B \to B$*.*

The isomorphism family of the product consists of all order isomorphisms $\theta : x \cong x'$ *between finite configurations* x, x' *of* $A \times B$*, with order inherited from the product, for which* $\theta_A = \{(\pi_1(p), \pi_1(p')) \mid (p, p') \in \theta\}$ *is in the isomorphism family of* A *and* $\theta_B = \{(\pi_2(p), \pi_2(p')) \mid (p, p') \in \theta\}$ *is in the isomorphism family of* B*.*

Let $f, f' : C \to A$ *and* $g, g' : C \to B$ *in* $\mathcal{SE}$*. If* $f \sim f'$ *and* $g \sim g'$*, then* $\langle f, g \rangle \sim \langle f', g' \rangle$*.*

The category $\mathcal{SE}$ does not have a terminal object. However, the event structure with symmetry ϖ defined in Example 2 satisfies an appropriately weakened property (it is a simple instance of a biterminal object):

Proposition 7. *For any event structure with symmetry* A *there is a map* $f : A \to \varpi$ *in* $\mathcal{SE}$ *and moreover for any two maps* $f, g : A \to \varpi$ *we have* $f \sim g$*.*

The category $\mathcal{SE}$ does not have pullbacks and equalizers in general. However:

Theorem 3.
(i) Let $f, g : A \to B$ *be two maps between event structures with symmetry. They have a pseudo equalizer, i.e. an event structure with symmetry* E *and map* $e : E \to A$ *such that* $f \circ e \sim g \circ e$ *which satisfies the further property that for any event structure with symmetry* E' *and map* $e' : E' \to A$ *such that* $f \circ e' \sim g \circ e'$*, there is a unique map* $h : E' \to E$ *such that* $e' = e \circ h$*.*
(ii) Let $f : A \to C$ *and* $g : B \to C$ *be two maps between event structures with symmetry. They have a pseudo pullback, i.e. an event structure with symmetry* D *and maps* $p : D \to A$ *and* $q : D \to B$ *such that* $f \circ p \sim g \circ q$ *which satisfies the further property that for any event structure with symmetry* D' *and maps* $p' : D' \to A$ *and* $q' : D' \to B$ *such that* $f \circ p' \sim g \circ q'$*, there is a unique map* $h : D' \to D$ *such that* $p' = p \circ h$ *and* $q' = q \circ h$*.*

There are obvious weakenings of the conditions of (i) and (ii) in which the uniqueness is replaced by uniqueness up to $\sim$ and equality by $\sim$—these are simple special cases of bilimits called biequalizers and bipullbacks when we regard $\mathcal{SE}$ as a 2-category. As in Theorem 3, we follow tradition and call the stricter construction described in (ii) a *pseudo pullback*. In Theorem 2, that pairing of maps preserves $\sim$ means that the products described are 2-products in $\mathcal{SE}$ regarded as a 2-category. For an accessible introduction to limits in 2-categories see [18].

5 Functors and Pseudo Monads

Certain functors on $\mathcal{E}$, the category of event structures, straightforwardly induce functors on $\mathcal{SE}$, the enriched category of event structures with symmetry. Say a functor $F : \mathcal{A} \to \mathcal{B}$ has *monic mediators for products* when for all products

$A \times A, \pi_1, \pi_2$ in $\mathcal{A}$ and $F(A) \times F(A), p_1, p_2$ in $\mathcal{B}$ the unique mediating map h in the commuting diagram

$$\begin{array}{ccccc} & & F(A \times A) & & \\ & \swarrow^{F(\pi_1)} & \downarrow h & \searrow^{F(\pi_2)} & \\ F(A) & \xleftarrow[p_1]{} & F(A) \times F(A) & \xrightarrow[p_2]{} & F(A) \end{array}$$

is monic. A functor on several, even infinitely many, arguments $F : \mathcal{E} \times \cdots \times \mathcal{E} \times \cdots \to \mathcal{E}$ which preserves pullbacks, open maps and has monic mediatiors for products will induce a functor on event structures with symmetry respecting $\sim$ on homsets. (A map in a product of categories, such as $\mathcal{E} \times \cdots \times \mathcal{E} \times \cdots$, is taken to be open iff it is open in each component.) We consider some examples.

5.1 Operations

Simple Parallel Composition. For example, consider the functor $\|: \mathcal{E} \times \mathcal{E} \to \mathcal{E}$ which given two event structures puts them in parallel. Let $(A, \mathrm{Con}_A, \leq_A)$ and $(B, \mathrm{Con}_B, \leq_B)$ be event structures. The events of $A \| B$ are $(\{0\} \times A) \cup (\{1\} \times B)$; with $(0, a) \leq (0, a')$ iff $a \leq_A a'$ and $(1, b) \leq (1, b')$ iff $b \leq_B b'$; and with a subset of events C consistent in $A \| B$ iff $\{a \mid (0, a) \in C\} \in \mathrm{Con}_A$ and $\{b \mid (1, b) \in C\} \in \mathrm{Con}_B$. The operation extends to a functor—put the two maps in parallel. It is not hard to check that the functor $\|$ preserves pullbacks and open maps, and that the mediating maps $(A \times A) \| (B \times B) \to (A \| B) \times (A \| B)$ are monic. Consequently it induces a functor $\|: \mathcal{SE} \times \mathcal{SE} \to \mathcal{SE}$ which preserves $\sim$ on homsets. On the same lines the functor giving the parallel composition $\|_{i \in I} A_i$ of countably-indexed event structures $A_i, i \in I$, extends to a functor on event structures with symmetry.

Sum. Similarly, the coproduct or sum of two event structures extends to the sum of event structures with symmetry. Let $(A, \mathrm{Con}_A, \leq_A)$ and $(B, \mathrm{Con}_B, \leq_B)$ be event structures. The events of the sum $A + B$ are $(\{0\} \times A) \cup (\{1\} \times B)$; with $(0, a) \leq (0, a')$ iff $a \leq_A a'$ and $(1, b) \leq (1, b')$ iff $b \leq_B b'$; but now a subset of events C is consistent in $A + B$ iff there is $C_0 \in \mathrm{Con}_A$ such that $C = \{(0, a) \mid a \in C_0\}$ or there is $C_1 \in \mathrm{Con}_B$ such that $C = \{(1, a) \mid a \in C_1\}$. We can also form a sum $\Sigma_{i \in I} A_i$ of event structures A_i indexed by a set I. Again this extends to a functor on event structures with symmetry.

5.2 Pseudo Monads

That $\mathcal{SE}$ is enriched over equivalence relations ensures that it supports the definitions pseudo functors and pseudo natural transformations, which here parallel those of functor and natural transformation, but with equality replaced by $\sim$. In the same spirit a pseudo monad on $\mathcal{SE}$ satisfies variants of the usual monad laws but expressed in terms of $\sim$ rather than equality (we can ignore the extra coherence conditions [5] as they trivialize in the simple situation here).

As examples we consider two particular pseudo monads which we can apply to the semantics of higher-order nondeterministic processes.

The Copying Pseudo Monad. The copying operation ! of Example 3 extends to a functor on $\mathcal{SE}$. Let $f : A \to B$ be a map of event structures with symmetry. Define $!f : !A \to !B$ by taking $!f(i, a) = (i, f(a))$ for all events a of A. The functor ! preserves $\sim$ on homsets. (It is not induced by a functor on $\mathcal{E}$.)

The component of the unit $\eta^!_E : E \to !E$ acts so $\eta^!_E(e) = (0, e)$ for all events $e \in E$—it takes an event structure with symmetry E into its zeroth copy in $!E$.

The multiplication map relies on a subsidiary pairing function on natural numbers $[_, _] : \omega \times \omega \to \omega$ which we assume is injective. The component of the multiplication $\mu^!_E : !!E \to !E$ acts so $\mu^!_E(i, j, e) = ([i, j], e)$.

It can be checked that the unit and the multiplication are natural transformations and that the usual monad laws, while they do not hold up to equality, do hold up to $\sim$. The somewhat arbitrary choice of the zeroth copy in the definition of the unit and pairing function on natural numbers in the definition of the multiplication don't really matter in the sense that other choices would lead to components $\sim$-equivalent to those chosen. (Different choices lead to natural transformations related by modifications with $\sim$ at all components.)

The Partiality Pseudo Monad. Let E be an event structure with symmetry. Define $E_* =_{\text{def}} E \parallel \varpi$, *i.e.* it consists of E and ϖ put in parallel.

The component of the unit $\eta^*_E : E \to E_*$ acts so $\eta^*_E(e) = (0, e)$ for all events $e \in E$—so taking E to its copy in $E \parallel \varpi$.

The component of the multiplication $\mu^*_E : (E_*)_* \to E_*$ acts so $\mu^*_E(0, (0, e)) = (0, e)$ and $\mu^*_E(0, (1, j)) = [0, j]$ and $\mu^*_E(1, k) = [1, k]$, where we use the pairing function on natural numbers above to map the two disjoint copies of ω injectively into ω.

Both η^* and μ^* are natural transformations and the usual monad laws hold up to $\sim$ making a pseudo monad. Again, the definition of multiplication is robust; if we used some alternative way to inject $\omega + \omega$ into ω the resulting multiplication would be $\sim$-related at each component to the one we have defined.

The category of event structures with partial maps has played a central role in the event structure semantics of synchronizing processes [23]. It readily generalizes to accommodate symmetry and reappears as the Kleisli bicategory of $(_)_*$.

Definition 7. *Let (A, l_A, r_A) and (B, l_B, r_B) be event structure with symmetry. A* partial map *of event structures with symmetry $f : (A, l_A, r_A) \rightharpoonup (B, l_B, r_B)$ consists of a partial map of event structures $f : A \rightharpoonup B$ for which there is a (necessarily unique) partial map of event structures $h : S_A \rightharpoonup S_B$ ensuring*

$$\langle l_B, r_B \rangle \circ h = (f \times f) \circ \langle l_A, r_A \rangle \ .$$

Partial maps of event structures with symmetry form a category; they compose as partial maps of event structures and share the same identity maps. We can define an equivalence relation $\sim$ on partial maps of event structures with symmetry by the obvious analogue of Definition 5. The category is enriched over equivalence

relations. (The full subcategory of event structures with identity symmetry is isomorphic to the category of event structures with partial maps.)

Proposition 8. *The Kleisli bicategory of the pseudo monad $(-)_*$ and the category of event structures with symmetry and partial maps (regarded as a 2 category) are biequivalent; the biequivalence is the identity on objects and takes maps $f : A \to B_*$ in the Kleisli bicategory to partial maps $\bar{f} : A \rightharpoonup B$, undefined precisely when the image is in ϖ.*

Equivalences. We have enough operations to derive some useful equivalences. Below we use 1 to denote the single-event event structure with symmetry and $\otimes$ for the product of event structures with symmetry with partial maps.

Proposition 9. *For event structures with symmetry:*

(i) $!A \,\|\, !B \simeq !(A+B)$ *and* $\|_{k\in K} !A_k \simeq !\Sigma_{k\in K} A_k$ *where* K *is a countable set.*
(ii) $\varpi \simeq !1$ *and* $A \times \varpi \simeq A$.
(iii) $A_* \simeq A \parallel \varpi$, $(!A)_* \simeq !(A+1)$ *and* $(A \otimes B)_* \simeq A_* \times B_*$.

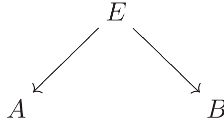
The equivalence $!A \,\|\, !B \simeq !(A+B)$, and its infinite version in (i), express the sense in which copying obviates choice. More importantly, they and the other the equivalences enable definitions by case analysis on events, also in the presence of asynchrony.

6 Applications

Here we present some unfinished applications, the subject of current work.

6.1 Spans

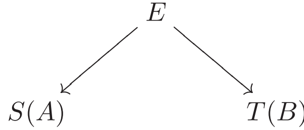
Because $\mathcal{SE}$ has pseudo pullbacks—Theorem 3, we can imitate the standard construction of the bicategory of spans (see [14]) to produce a bicategory $\text{Span}_{\mathcal{SE}}$. Its objects are event structures with symmetry. Its maps $\text{Span}_{\mathcal{SE}}(A, B)$, from A to B, are spans



composed using the pseudo pullbacks of of Theorem 3 (ii). $\text{Span}_{\mathcal{SE}}$ has a tensor and function space given by the product of $\mathcal{SE}$.

An individual span can be thought of as a process computing from input of type A to output of type B. But given the nature of maps in $\mathcal{SE}$ such a process is rather restricted; from a computational view the process is unnaturally symmetric and 'ultra-linear' because any output event is synchronized with an event of input.

We wish to modify the maps of a span to allow for different regimes of input and output. A systematic way to do this is through the use of pseudo monads on $\mathcal{SE}$ and build more general spans



for pseudo monads S and T. For example a span in which $S = (_)_*$ and $T =!(_)$ would permit output while ignoring input and allow the output of arbitrarily many similar events of type B. But for such general spans to compose, we require that S and T satisfy several conditions, which we can only indicate here:

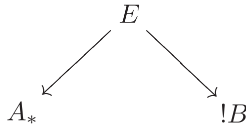
– in order to lift to pseudo comonads and monads on spans, S and T should be 'cartesian' pseudo monads, now w.r.t. pseudo/bipullbacks (adapting [3]);
– in order to obtain a comonad-monad distributive law for the liftings of S and T to spans it suffices to have a 'cartesian' distributive law for S and T, with commutativity up to $\sim$, with extra pseudo/bipullback conditions on two of the four diagrams (adapting [13]).

The two pseudo monads $S = (_)_*$ and $T =!(_)$ do satisfy these requirements with a distributive law with components $\lambda_E : (!E)_* \to!(E_*)$ such that $\lambda_E(0,(j,e)) = (j,(0,e))$ and $\lambda_E(1,k) = (0,(1,k))$.

The paper has concentrated on the categories of event structures $\mathcal{E}$ and $\mathcal{SE}$ with total maps. In particular, general spans have been described for maps in $\mathcal{SE}$. Analogous definitions and results hold for rigid maps, and for spans in $\mathcal{SE}_r$—event structures with symmetry and rigid maps. Total maps on event structures with symmetry can be obtained as Kleisli maps w.r.t. a monad $\mathcal{S}aug$ on $\mathcal{SE}_r$—see [29]. It appears that we can ground all the maps and spans of event structures of interest in $\mathcal{SE}_r$. The category $\mathcal{SE}_r$ is emerging as *the* fundamental category of event structures.

6.2 Event Types

The particular bicategory of spans



is already quite an interesting framework for the semantics of higher-order processes. It supports types including:

- Prefix types $\bullet!T$: in which a single event $\bullet$ prefixes $!T$ for an event structure with symmetry T.

- Sum types $\Sigma_{\alpha \in A} T_\alpha$: the sum of a collection T_α, for $\alpha \in A$, of event structures with symmetry—the sum functor is described in Section 5.1. Sum types may also be written $a_1 T_1 + \cdots + a_n T_n$ when the indexing set is finite. The empty sum type is the empty event structure $\emptyset$.
- Tensor types $T_1 \otimes T_2$: the product in $\mathcal{SE}_p$.
- Function types $T_1 \multimap T_2$: a form of function space, defined as the product $(T_1)_* \times !T_2$ in $\mathcal{SE}$.[4]
- Recursively defined types: treated for example as in [23,25].

The types describe the events and basic causalities of a process, and in this sense are examples of *event types*, or *causal types*, of a process. (One can imagine other kinds of spans and variations in the nature of event types.)

As an example, the type of a process only able to do actions within $a_1, \cdots, a_k$ could be written

$$a_1 \bullet !\emptyset + \cdots + a_k \bullet !\emptyset \,,$$

which we condense to $a_1 + \cdots + a_k$, as it comprises the event structure with events $a_1, \cdots, a_k$ made in pairwise-conflict, with the identity relation of causal dependency. The judgement that a closed process, represented by an event structure with symmetry E, has this type would be associated with a degenerate span from the biterminal $\emptyset_*$ to $!(a_1 + \cdots + a_k)$, so essentially with a map

$$l : E \rightarrow !(a_1 + \cdots + a_k)$$

in $\mathcal{SE}$, 'labelling' events by their actions. By Proposition 9 (i), there is an equivalence

$$!a_1 \parallel \cdots \parallel !a_k \;\simeq\; !(a_1 + \cdots + a_k)\,,$$

and a process of this type can only do actions $a_1, \cdots, a_k$, though with no bound on how many times any action can be done.

The type of CCS, with channels A, can be written as

$$Act = \tau \bullet !\emptyset \;+\; \Sigma_{\bar{a} \in \bar{A}} \bullet !\emptyset \;+\; \Sigma_{a \in A} \bullet !\emptyset\,.$$

We can describe the parallel composition of CCS by a partial function from the events $Act \otimes Act$ to the events $!Act$, expressing how events combine to form synchronization events (the second line), or can occur asynchronously (the first):

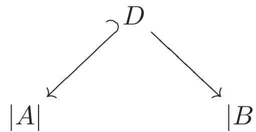
$$\begin{array}{l}(\alpha, *) \mapsto \mu^!_{Act}(0, \eta^!_{Act}(\alpha))\,, \quad (*, \alpha) \mapsto \mu^!_{Act}(1, \eta^!_{Act}(\alpha))\,,\\ (a, \bar{a}),\ (\bar{a}, a) \mapsto \eta^!_{Act}(\tau), \quad \text{and undefined otherwise.}\end{array}$$

This partial function is also a partial map of event structures from $Act \otimes Act$ to $!Act$—it would have violated local injectivity and not been a map of event structures, had we chosen simply $\eta^!_{Act}(\alpha)$ as the resulting events in the first two clauses. The partial function is readily interpreted as a span from $Act \otimes Act$ to

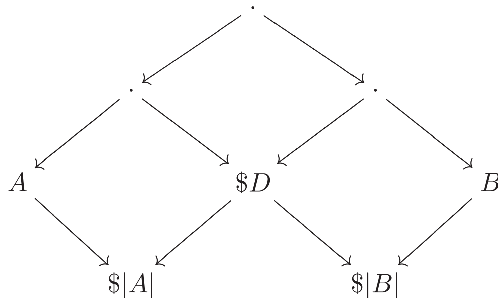
[4] Although this function space seems hard to avoid for this choice of span and tensor, we don't quite have $_ \otimes B$ a left biadjoint to $B \multimap _$.

$!Act$—its vertex is essentially the domain of definition of the partial function. Post-composing its left 'leg' with $\eta^*_{Act\otimes Act}$ we obtain a span from $(Act\otimes Act)_*$ to $!Act$ which denotes the parallel composition of CCS. Given two CCS processes represented by degenerate spans, we can combine them to a process with event type $Act\otimes Act$, denoting a degenerate span ending in $!(Act\otimes Act)$. Its composition with the span for parallel composition can be shown to give the traditional event-structure semantics of parallel composition in CCS [23,25,26,21].

In fact there is a general way to define spans from partial functions on events which respect symmetry. There is a functor from event structures with symmetry $\mathcal{SE}$ to equivalence relations; it takes an event structure with symmetry A to the equivalence relation $|A|$ induced by the symmetry on the set of events. The functor is enriched in equivalence relations and has a right biadjoint \$ which takes an equivalence relation $(L, R \subseteq L \times L)$ to the event structure with symmetry $!(L, l, r : R \to L)$, where we understand L as an event structure with events in pairwise conflict with trivial causal dependency, R similarly, and with symmetry maps given as the obvious projections from R to L. (The biadjunction between event structures with symmetry and equivalence relations relies on the event structures being consistent-countable.) For event structures with symmetry A and B, a partial function respecting equivalence relations from $|A|$ to $|B|$ can be regarded as a span



in the category of equivalence relations—the equivalence relation D being where the partial function is defined. The unit of the biadjunction with equivalence relations has components $A \to \$|A|$ and $B \to \$|B|$, so by applying \$ to the span above and taking successive pseudo pullbacks we obtain a span from A to B:



A partial function between on events may not be so simple to define directly by case analysis on events. This is because the events that arise in products of event structures can be quite complicated; the events of a product $A \otimes B$ of event structures A and B are perhaps best seen as prime configurations of a product of stable families—see Appendix B. Their complexity contrasts with

the simplicity of the events arising in constructions on stable families; the events of the corresponding product of stable families are simple pairs $(a, *)$, $(*, b)$ and (a, b), where a and b are events of the components. For this reason it can be easiest to define a partial map on event structures (so a partial function on their events) via a partial map between their representations as stable families. This is so below, in a putative 'true concurrency' definition of a version of higher-order CCS and its parallel composition.

A form of higher-order CCS could reasonably be associated with the recursive type

$$T = \tau \bullet !T \;+\; \Sigma_{\bar{a} \in \bar{A}} \bullet !(T \otimes T) \;+\; \Sigma_{a \in A} \bullet !(T \multimap T)\,,$$

specifying that an event of a higher-order CCS process is either a 'process' event following a τ-event, a 'concretion' event following an output synchronization $\bar{a} \in \bar{A}$, or an 'abstraction' event following an input synchronization $a \in A$. Why is the first component in the type T of the form $\tau \bullet !T$ and not just $\tau \bullet !\emptyset$? Without the present choice I cannot see how to ensure that in the parallel composition an interaction between a concretion and abstraction event always follows a corresponding synchronization at their channels.

Parallel composition in higher-order CCS would be associated with a typing judgment $x : T, y : T \vdash (x \,|\, y) : T$. The typing judgment should denote a span from $(T \otimes T)_*$ to $!T$. As above, we can define a tentative parallel composition via a partial function from $|T \otimes T|$ to $|!T|$. The partial function should describe when and how events of T combine. Because events of the product $T \otimes T$ are quite complicated we must face the difficulties outlined above. However, first we need a makeshift syntax for events in T. Events of higher-order CCS are either internal events τ, subsequent process events $\tau.(i, t)$, output synchronizations $\bar{a}$, subsequent concretion events $\bar{a}.(i, c)$, input synchronizations a, or subsequent abstraction events $a.(j, f)$—the natural numbers i, j index the copies in !-types. In the notation for events of the product $T \otimes T$ we exploit the way it is built from a product of stable families; in the product of stable families out of which $T \otimes T$ is constructed events have the simple form of pairs $(t, *)$, $(*, t)$ or (t, t'), where t and t' are events of T. We can define a partial map from this stable family to the stable family of $!T$ by case analysis on events:

$$t \,|\, * \;=\; \mu^!_T(0, \eta^!_T(t))\,, \quad * \,|\, t \;=\; \mu^!_T(1, \eta^!_T(t))\,,$$

$$a \mid \bar{a} \;=\; \bar{a} \mid a \;=\; \eta^!_T(\tau)\,,$$

$$a.(i, f) \mid \bar{a}.(j, c) \;=\; \bar{a}.(i, c) \mid a.(j, f) \;=\; \eta^!_T(\tau.\mu^!_T([i, j],\ (f \,|\, c)))$$

provided $(f \,|\, c)$ is defined,

$$\tau.(i, t) \mid \tau.(j, t') \;=\; \mu^!_T([i, j], (t \,|\, t')) \;\text{ provided } (t \,|\, t') \text{ is defined,}$$

$$\tau.(i, t) \mid \alpha \;=\; \mu^!_T(i, (t \,|\, \alpha))\,, \quad \alpha \mid \tau.(j, t) \;=\; \mu^!_T(j, (\alpha \,|\, t))$$

provided α is not of the form $\tau.(k, t'')$, and undefined otherwise.

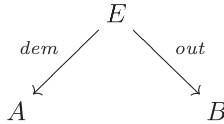
We have combined indices i, j using an injective pairing $[i, j]$ of natural numbers. The definition above relies on our simultaneously defining not just how process

events combine, but also how 'abstraction' events f in type $T \multimap T$ and 'concretion' events c in type $T \otimes T$ combine to form a process event $(f \mid c)$ in type $!T$. We postpone the full definition. Although provisional, I hope the example helps illustrate the aims and present difficulties—there may be difficulties that I'm not aware of.

Clearly the syntax of operations to accompany the types is unfinished and really needed. But I believe the examples indicate the potential of a more thorough study of event types and give a flavour of the style of definition they might support, a method of definition which breaks away from traditional 'interleaving' approaches to concurrency.

6.3 Nondeterministic Dataflow and Affine-HOPLA

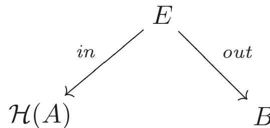
'Stable' spans of event structures have been used to give semantics to nondeterministic dataflow [19] and the higher-order process language affine-HOPLA [16]. They are generalisations of Berry's stable functions [2]: deterministic stable spans correspond to stable functions—see [19]. A stable span



consists of a 'demand' map $dem : E \to A$ and a rigid map $out : E \to B$. That dem is a demand map means that it is a function from $\mathcal{C}^o(A)$ to $\mathcal{C}^o(B)$ which preserves unions of configurations when they exist. An equivalent way to view the demand map dem is as a function from the events of E to finite configurations of A such that if $e \leq e'$ then $dem(e) \subseteq dem(e')$, and if $X \in \mathrm{Con}$ then $demX\uparrow$, *i.e.*, the demands are compatible. The intuition is that $dem(e)$ is the minimum input required for the event e to occur; when it does $out(e)$ is observed in the output. (The stable span is *deterministic* when $demX\uparrow$ implies $X \in \mathrm{Con}$, for X a finite subset of events in E.)

On the face of it demand maps are radically different from rigid maps of event structures. They can however be recovered as Kleisli maps associated with a pseudo monad $\mathcal{H}$ on event structures with symmetry and rigid maps.

Roughly the pseudo monad $\mathcal{H}$ adjusts the nature of events so that they record the demand history on the input. This enables stable spans to be realized as spans



of rigid maps in $\mathcal{SE}_r$. Such spans are a special case of the general spans of Section 6.1, with the identity monad on the right-hand-side. Because of 'Seely conditions' $\mathcal{H}(E \parallel F) \simeq \mathcal{H}(E) \times \mathcal{H}(F)$ and $\mathcal{H}(\emptyset) \simeq \top$ relating parallel composition $\parallel$ and its unit, the empty event structure $\emptyset$, to product $\times$ and the biterminal

object $\top$ in $\mathcal{SE}_r$, we obtain a description of the function space, w.r.t. parallel composition $A \parallel B$, as $A \multimap B = \mathcal{H}(A) \times B$. A very different route to the definition of function space using stable families is described in the PhD thesis [16]. The pseudo monad $\mathcal{H}$ and the biadjunction which induces it are described in [29].

6.4 Unfoldings

Another application of symmetry is to the unfolding of Petri nets with multiple tokens, and the unfolding of higher-dimensional automata (hda's) [7]. Unfoldings of 1-safe Petri nets to occurrence nets and event structures were introduced in [15], and have since been applied in a variety of areas from model checking to self-timed circuits and the fault diagnosis of communication networks. The unfoldings were given a universal characterisation a little later in [24] (or see [21]) and this had the useful consequence of providing a direct proof that unfolding preserved products and so many parallel compositions. There is an obstacle to an analogous universal characterisation of the unfolding of nets in which places/conditions hold with multiplicities: the symmetry between the multiple occurrences in the original net is lost in unfoldings to standard occurrence nets or event structures, and this spoils universality through non-uniqueness. However through the introduction of symmetry uniqueness up to symmetry obtains, and a universal characterisation can be regained [9].

We can illustrate the role symmetry plays in the unfolding of nets and hda's through a recent result relating event structures with symmetry to certain presheaves.[5] Let $\mathbb{P}$ be the category of finite elementary event structures (so essentially finite partial orders) with rigid maps. Form the presheaf category $\widehat{\mathbb{P}}$ which by definition is the functor category $[\mathbb{P}^{op}, \mathbf{Set}]$. From [27] we obtain that event structures with rigid maps (called 'strong' in [27]) embed fully and faithfully in $\widehat{\mathbb{P}}$ and are equivalent to those presheaves which are *separated* w.r.t. the Grothendieck topology with basis collections of jointly surjective maps in $\mathbb{P}$, and satisfy a further *mono* condition. Presheaves over $\widehat{\mathbb{P}}$ are thus a kind of generalised event structure.

There is clearly an inclusion functor $I : \mathbb{P} \hookrightarrow \mathcal{SE}_r$ of finite elementary event structures into event structures with symmetry and rigid maps. Thus there is a functor $F : \mathcal{SE}_r \to \widehat{\mathbb{P}}$ taking an event structure with symmetry E to the presheaf $\mathcal{SE}_r(I(_), E)/\!\sim$. Event structures with symmetry yield more than just separated presheaves, and quite which presheaves they give rise to is not yet understood. But by restricting to event structures with symmetry $(E, l, r : S \to E)$ for which the symmetry is *strong*, in the sense that the mono $\langle l, r\rangle : S \to E \times E$ reflects consistency, we will always obtain nonempty separated presheaves. Let $\mathcal{SSE}_r$ be the category of event structures with strong symmetry and rigid maps. Let Sep($\mathbb{P}$) be the full subcategory of non-empty separated presheaves. So restricted, we obtain a functor $F : \mathcal{SSE}_r \to \widehat{\mathbb{P}}$ taking an event structure with strong symmetry E to the nonempty separated presheaf $\mathcal{SSE}_r(J(_), E)/\!\sim$.

[5] The result is inspired by joint work with the Sydney Concurrency Group: Richard Buckland, Jon Cohen, Rob van Glabbeek and Mike Johnstone.

The functor F can be shown to have a right biadjoint, a functor G, producing an event structure with strong symmetry from a nonempty separated presheaf. The right biadjoint G is full and faithful (once account is taken of the the equivalence $\sim$ on maps). (The existence of G relies on the event structures being consistent-countable.) It shows how separated presheaves embed via a reflection fully and faithfully in event structures with symmetry:

$$\mathcal{SSE}_r \underset{G}{\overset{F}{\rightleftarrows}} \mathrm{Sep}(\mathbb{P}) \, . \tag{1}$$

The proof of the biadjunction has only been carried out for rigid maps, the reason why we have insisted that the maps of event structures in this section be rigid. (One could hope for a similar biadjunction without restricting F to strong symmetries.)

Higher-dimensional automata [7] are most concisely described as cubical sets, *i.e.* as presheaves over $\mathbb{C}$, a category of cube shapes of all dimensions with maps including *e.g.* 'face' maps, specifying how one cube may be viewed as a (higher-dimensional) face of another. We can identify the category of hda's with the presheaf category $\widehat{\mathbb{C}}$. There are some variations in the choice of maps in $\mathbb{C}$, according to whether the cubes are oriented and whether degeneracy maps are allowed. For simplicity we assume here that the cubes are not oriented and have no degeneracy maps, so the maps are purely face maps. Roughly, then the maps of $\mathbb{P}$ and $\mathbb{C}$ only differ in that maps in $\mathbb{P}$ fix the initial empty configuration whereas face maps in $\mathbb{C}$ are not so constrained. By modifying the maps of $\mathbb{P}$ to allow the initial configuration to shift under maps, we obtain a category $\mathbb{A}$ into which both $\mathbb{P}$ and $\mathbb{C}$ include:

$$\mathbb{P} \overset{J}{\hookrightarrow} \mathbb{A} \overset{K}{\hookleftarrow} \mathbb{C}$$

Now we can construct a functor from $H : \mathbb{P} \to \widehat{\mathbb{C}}$; it takes p in $\mathbb{P}$ to the presheaf $\mathbb{A}(K(_), J(p))$. Taking its left Kan extension over the Yoneda embedding of $\mathbb{P}$ in $\widehat{\mathbb{P}}$ we obtain a functor

$$H_! : \widehat{\mathbb{P}} \to \widehat{\mathbb{C}} \, .$$

For general reasons [4], the functor $H_!$ has a right adjoint H^* taking an hda Y in $\widehat{\mathbb{C}}$ to the presheaf $\widehat{\mathbb{C}}(H(_), Y)$ in $\widehat{\mathbb{P}}$:

$$\widehat{\mathbb{P}} \underset{H^*}{\overset{H_!}{\rightleftarrows}} \widehat{\mathbb{C}} \, . \tag{2}$$

We cannot quite compose the biadjunctions (1) and the adjunction (2) because (1) is only for separated presheaves. However restricting to hda's which are separated, now w.r.t. a basis of jointly surjective maps in $\mathbb{C}$,[6] will ensure that

[6] For a separated hda, cubes which share the same 1-dimensional edges must be equal (so 'no ravioli').

they are sent to separated presheaves over $\mathbb{P}$ and so to event structures with symmetry. General Petri nets give rise to separated hda's (for example, with the 'self-concurrent individual token interpretation' of [7]). So we obtain a rather abstract construction of an unfolding of general nets to event structures with symmetry. Again, much more needs to be done, both mathematically in seeking a generalisation of the biadjunction (1) to all event structures with symmetry, and in understanding unfoldings concretely so that they can be made amenable algorithmically.

Acknowledgments. I'm grateful to Marcelo Fiore, Martin Hyland, Gordon Plotkin, Lucy Saunders-Evans, Pawel Sobocinski, Sam Staton, Dominic Verity and the Sydney Concurrency Group for discussions and encouragement. I acknowledge the partial support of EPSRC grant GR/T22049/01.

References

1. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full Abstraction for PCF. Information and Computation 163, 409–470 (2000)
2. Berry, G.: Modèles completement adéquats et stables des λ-calculs typés. Thèse de Doctorat d'Etat, Université de Paris VII (1979)
3. Burroni, A.: T-catégories. Cahiers de topologie et géométrie différentielle, XII(3) (1971)
4. Cattani, G.L., Winskel, G.: Profunctors, open maps and bisimulation. In: MSCS (2005)
5. Cheng, E., Hyland, J.M.E., Power, A.J.: Pseudo-distributive laws. ENTCS 83 (2004)
6. Crazzolara, F., Winskel, G.: Composing Strand Spaces. In: Agrawal, M., Seth, A.K. (eds.) FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 2556, Springer, Heidelberg (2002)
7. van Glabbeek, R.J.: On the expressiveness of higher dimensional automata. EXPRESS 2004. ENTCS 128(2) (2005)
8. Doghmi, S.F., Guttman, J.D., Thayer, F.J.: Searching for shapes in cryptographic protocols. In: TACAS'07 (2007)
9. Hayman, J., Winskel, G.: The unfolding of general Petri nets. Forthcoming.
10. Johnstone, P.: Sketches of an elephant, a topos theory compendium, vol.1. OUP (2002)
11. Joyal, A., Nielsen, M., Winskel, G.: Bisimulation from open maps. LICS '93 special issue of Information and Computation 127(2), 164–185 (1996) Available as BRICS report, RS-94-7
12. Kahn, G., Plotkin, G.D.: Concrete domains. TCS 121(1& 2), 187–277 (1993)
13. Koslowski, J.: A monadic approach to polycategories. Theory and Applications of Categories 14(7), 125–156 (2005)
14. Mac Lane, S.: Categories for the Working Mathematician. Springer, Heidelberg (1971)
15. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains. TCS 13(1), 85–108 (1981)
16. Nygaard, M.: Domain theory for concurrency. PhD Thesis, University of Aarhus (2003)

17. Nygaard, M., Winskel, G.: Domain theory for concurrency. TCS 316, 153–190 (2004)
18. Power, A.J.: 2-Categories. BRICS Lecture Notes, Aarhus University (March 1998)
19. Saunders-Evans, L., Winskel, G.: Event structure spans for non-deterministic dataflow. In: Proc. Express'06, ENTCS (2006)
20. Saunders-Evans, L.: Events with persistence. Forthcoming PhD thesis, University of Cambridge Computer Laboratory (2007)
21. Winskel, G., Nielsen, M.: Models for Concurrency. Handbook of Logic and the Foundations of Computer Science 4, 1–148 OUP (1995)
22. Winskel, G.: Events in Computation. PhD thesis, Univ. of Edinburgh (1980) Available from http://www.cl.cam.ac.uk/users/gw104
23. Winskel, G.: Event structure semantics of CCS and related languages. In: Nielsen, M., Schmidt, E.M. (eds.) Automata, Languages, and Programming. LNCS, vol. 140, Springer, Heidelberg (1982), http://www.cl.cam.ac.uk/users/gw104
24. Winskel, G.: A new definition of morphism on Petri Nets. In: STACS'84: pp. 140–150 (1984)
25. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Advances in Petri Nets 1986. Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986. LNCS, vol. 255, Springer, Heidelberg (1987)
26. Winskel, G.: An introduction to event structures. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. LNCS, vol. 354, Springer, Heidelberg (1989)
27. Winskel, G.: Event structures as presheaves—two representation theorems. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, Springer, Heidelberg (1999)
28. Winskel, G.: Relations in concurrency. In: Invited talk, LICS'05 (2005)
29. Winskel, G.: Event structures with symmetry. In: the Plotkin Festschrift. ENTCS 172 (2007), See http://www.cl.cam.ac.uk/users/gw104 for corrections

A Appendix: Equivalence Relations [10]

Assume a category with pullbacks. Let E be an object of the category. A *relation* on E is a pair of maps $l, r : S \to E$ for which l, r are *jointly monic*, *i.e.* for all maps $x, y : D \to S$, if $lx = ly$ and $rx = ry$, then $x = y$. Equivalently, if the category has binary products, a relation on E is a pair of maps $l, r : S \to E$ for which the mediating map $\langle l, r \rangle : S \to E \times E$ is monic. The relation is an *equivalence relation* in the category iff it is:
Reflexive: there is a (necessarily unique) map ρ such that

$$\begin{array}{ccccc} & & E & & \\ & \swarrow^{\mathrm{id}_E} & \vdots \rho & \searrow^{\mathrm{id}_E} & \\ E & \xleftarrow[l]{} & S & \xrightarrow[r]{} & E \end{array}$$

commutes;

Symmetric: there is a (necessarily unique) map σ such that

$$\begin{array}{ccccc} & & S & & \\ & \swarrow^{r} & \vdots^{\sigma} & \searrow^{l} & \\ E & \xleftarrow[l]{} & S & \xrightarrow[r]{} & E \end{array}$$

commutes;
Transitive: there is a (necessarily unique) map τ such that

$$\begin{array}{ccccccc} & & & P & & & \\ & & \swarrow^{f} & \vdots^{\tau} & \searrow^{g} & & \\ & S & & S & & S & \\ & {}^{l}\swarrow \; \searrow^{r} & & {}^{l}\swarrow \; \searrow^{r} & & {}^{l}\swarrow \; \searrow^{r} & \\ E & & & E & & & E \end{array}$$

commutes, where P, f, g is a pullback of r, l.

B Appendix: Stable Families

So event structures can be obtained from finitary prime algebraic domains. One convenient way to construct finitary prime algebraic domains is from stable families [23]. The use of stable families facilitates constructions such as products and pullbacks of event structures.

The use of stable families facilitates definitions on event structures.

Definition. A *stable family* comprises $\mathcal{F}$, a family of finite subsets, called *configurations*, satisfying:
Completeness: $Z \subseteq \mathcal{F}$ & $Z\uparrow \Rightarrow \bigcup Z \in \mathcal{F}$;
Coincidence-freeness: For all $x \in \mathcal{F}$, $e, e' \in x$ with $e \neq e'$,

$$(\exists y \in \mathcal{F}.\ y \subseteq x \ \&\ (e \in y \iff e' \notin y))\ ;$$

Stability: $\forall Z \subseteq \mathcal{F}.\ Z \neq \emptyset$ & $Z\uparrow \Rightarrow \bigcap Z \in \mathcal{F}$.

For $Z \subseteq \mathcal{F}$, we write $Z\uparrow$ to mean compatibility, *i.e.*

$$\exists x \in \mathcal{F} \forall z \in Z.\ z \subseteq x\ .$$

Configurations of stable families each have their own local order of causal dependency, so their own prime sub-configurations generated by their events. We can build an event structure by taking the events of the event structure to comprise the set of all prime sub-configurations of the stable family.

Definitions and Proposition. Let x be a configuration of a stable family $\mathcal{F}$. For $e, e' \in x$ define

$$e' \leq_x e \text{ iff } \forall y \in \mathcal{F}.\ y \subseteq x \ \&\ e \in y \Rightarrow e' \in y.$$

When $e \in x$ define the prime configuration

$$[e]_x = \bigcap \{y \in \mathcal{F} \mid y \subseteq x \ \&\ e \in y\}\ .$$

Then $\leq_x$ is a partial order and $[e]_x$ is a configuration such that

$$[e]_x = \{e' \in x \mid e' \leq_x e\}.$$

Moreover the configurations $y \subseteq x$ are exactly the down-closed subsets of $\leq_x$.

Definition and Proposition. Let $\mathcal{F}$ be a stable family. Then, $\Pr(\mathcal{F}) =_{\text{def}} (P, \text{Con}, \leq)$ is an event structure where:

$$P = \{[e]_x \mid e \in x \ \&\ x \in \mathcal{F}\}\ ,$$
$$Z \in \text{Con iff } Z \subseteq P \ \&\ \bigcup Z \in \mathcal{F} \text{ and,}$$
$$p \leq p' \text{ iff } p, p' \in P \ \&\ p \subseteq p'\ .$$

This proposition furnishes a way to construct an event structure with events the prime configurations of a stable family. In fact we can equip the class of stable families with maps (the definitions are the same as those for event structures). The configurations of an event structure form a stable family, so in this sense event structures are included in stable families. With respect to any of the maps (rigid, total or partial), the "inclusion" functor from the category of event structures to the category of stable families has a right adjoint, which on objects is the construction we have just given, producing an event structure from a stable family. The products w.r.t. total and partial maps are hard to define directly on the event structures of this article. It is however straightforward to define the products of stable families [23,29]. Right adjoints preserve limits, and so products in particular. Consequently we obtain products of event structures by first regarding them as stable families, and then producing the event structure from the product of the stable families. Pullbacks of event structures are obtained by restricting products to the appropriate equalizing set. See [29] for more details.

Ready to Preorder: Get Your BCCSP Axiomatization for Free!*

Luca Aceto[1], Wan Fokkink[2,3], and Anna Ingólfsdóttir[1]

[1] Reykjavík University, School of Science and Engineering
Ofanleiti 2, 103 Reykjavík, Iceland
[2] Vrije Universiteit, Section Theoretical Computer Science
Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
[3] CWI, Embedded Systems Group
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
luca@ru.is, wanf@cs.vu.nl, annai@ru.is

Abstract. This paper contributes to the study of the equational theory of the semantics in van Glabbeek's linear time - branching time spectrum over the language BCCSP, a basic process algebra for the description of finite synchronization trees. It offers an algorithm for producing a complete (respectively, ground-complete) equational axiomatization of any behavioral congruence lying between ready simulation equivalence and partial traces equivalence from a complete (respectively, ground-complete) inequational axiomatization of its underlying precongruence—that is, of the precongruence whose kernel is the equivalence. The algorithm preserves finiteness of the axiomatization when the set of actions is finite.

1 Introduction

The lack of consensus on what constitutes an appropriate notion of observable behaviour for reactive systems has led to a large number of proposals for behavioural equivalences and preorders for concurrent processes. In his by now classic paper [13], van Glabbeek presented the linear time - branching time spectrum of behavioural preorders and equivalences for finitely branching, concrete, sequential processes. The semantics in this spectrum are based on simulation notions and on decorated traces.

Van Glabbeek [13] studied the semantics in his spectrum in the setting of the process algebra BCCSP, which contains only the basic process algebraic operators from CCS [18] and CSP [17], but is sufficiently powerful to express all finite synchronization trees. In the aforementioned reference, van Glabbeek gave, amongst a wealth of other results, (in)equational axiomatizations for the preorders and equivalences in the spectrum, such that two closed BCCSP terms can be equated by the axioms if, and only if, they are related by the preorder or

* The first and third author were partly supported by the project "The Equational Logic of Parallel Processes" (nr. 060013021) of The Icelandic Research Fund.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 65–79, 2007.
© Springer-Verlag Berlin Heidelberg 2007

equivalence in question. Groote [14] obtained ω-completeness results for most of the axiomatizations, in case the alphabet of actions is infinite. (An axiomatization E is *ω-complete* when an equation can be derived from E if, and only if, all of its closed instantiations can be derived from E.) The papers [2,6,8,9,10] offer positive and negative results on the existence of finite (in)equational axiomatizations for several behavioural equivalences and preorders in the spectrum over the language BCCSP, both in the setting of finite and infinite sets of actions.

The work we present in this paper stems from the observation that all of the extant axiomatization results presented in the aforementioned studies are based on separate, and often rather similar, developments for preorders and equivalences. For the semantics in the spectrum lying between 2-nested simulation semantics and partial traces semantics, the equivalences are the *kernels* of the preorders—meaning that two processes are considered equivalent if, and only if, each is a refinement of the other with respect to the preorder—, which are therefore more basic than the equivalences. Since the equivalences are defined in terms of the preorders in a canonical fashion, it would be very satisfying, in order to achieve a higher degree of generality and to highlight the commonalities in the technical developments pertaining to axiomatization results for the semantics in the spectrum, to develop a general strategy for obtaining complete axiomatizations of the equivalences in the spectrum from complete axiomatizations of the preorders. This is the aim of this paper.

Our Contribution. We offer an algorithm for producing an ω-complete (respectively, ground-complete) equational axiomatization of any behavioral congruence lying between ready simulation equivalence and partial traces equivalence from an ω-complete (respectively, ground-complete) inequational axiomatization of its underlying precongruence—that is, of the precongruence whose kernel is the equivalence. The algorithm we give in this paper preserves finiteness of the axiomatization when the set of actions is finite. It follows that each equivalence in the spectrum whose discriminating power lies in between that of ready simulation and partial traces equivalence is finitely axiomatizable over the language BCCSP if so is its defining preorder.

Our algorithm may be seen as isolating and axiomatizing the ingredients that all of the extant proofs of completeness results for the class of behavioural equivalences we study have in common. It also eliminates the need to reprove, essentially from scratch, completeness results for a large fragment of behavioural equivalences in the spectrum once a completeness result has been obtained for their underlying preorders. The axiomatizations that are automatically generated by our algorithm are very similar, when not identical, to those presented in the literature. (See, for instance, the two specific examples of applications of our algorithm that are provided in Section 6.)

Our algorithm takes as input a sound and ω-complete (respectively, ground-complete) inequational axiomatization E for BCCSP modulo a preorder in the linear time - branching time spectrum that includes the ready simulation preorder. Without loss of generality, we assume that the four classic equations from [16] that

completely axiomatize bisimulation equivalence [18] are contained in E, and that so do the defining inequational axioms for ready simulation for each action a:

$$ax \preccurlyeq ax + ay \ .$$

The axiomatization $\mathcal{A}(E)$ generated by our algorithm from E contains the axioms for bisimulation equivalence together with the following equations, for each inequational axiom $t \preccurlyeq u$ in E:

- $t + u \approx u$; and
- $b(t + x) + b(u + x) \approx b(u + x)$ (for each action b, and some variable x that does not occur in $t + u$).

The main technical result in the paper is a theorem to the effect that the axiomatization $\mathcal{A}(E)$ is ω-complete (respectively, ground-complete) for the equivalence if E is ω-complete (respectively, ground-complete) for the preorder (Theorem 1). The proof of this statement is non-trivial, and relies on a careful analysis of the so-called *cover equations* [10] for the semantics in the linear time - branching time spectrum we consider in this study. Cover equations give us an explicit description of the equational theory for a particular semantics in terms of equations having a rather simple, and canonical, form.

Roadmap of the Paper. The paper is organized as follows. Section 2 reviews the syntax and the operational semantics for the language BCCSP, introduces the linear time time - branching time spectrum, and discusses the very basic notions of (in)equational logic used in this study. We present our algorithm in Section 3, where we also state the main theorem in the paper (Theorem 1) to the effect that the algorithm is guaranteed to produce an ω-complete (respectively, ground-complete) equational axiomatization of any behavioral congruence lying between ready simulation equivalence and partial traces equivalence from an ω-complete (respectively, ground-complete) inequational axiomatization of its underlying precongruence. The bulk of the rest of the paper (Sections 4–5) is devoted to a proof of our main result. Section 6 presents applications of our algorithm in the setting of simulation and failures semantics. We end the paper with some concluding remarks, and a detailed comparison with related work (Section 7).

2 Preliminaries

Syntax of BCCSP BCCSP(A) is a basic process algebra for expressing finite process behaviour. Its syntax consists of closed (process) terms p, q that are constructed from a constant $\mathbf{0}$, a binary operator $_+_$ called *alternative composition*, and unary *prefix* operators $a_$, where a ranges over some nonempty set A of *actions* (with typical elements a, b, c, d). (We write $|A|$ for the cardinality of the set A.) Open terms p, q, r, s, t, u can moreover contain occurrences of variables from a countably infinite set V (with typical elements w, x, y, z).

A (closed) substitution maps variables in V to (closed) terms. For every term t and (closed) substitution σ, the (closed) term $\sigma(t)$ is obtained by replacing every occurrence of a variable x in t by $\sigma(x)$. We often write t^σ in lieu of $\sigma(t)$.

A *context* $C[]$ is a BCCSP(A) term with exactly one occurrence of a hole $[]$ in it. For every context $C[]$ and term p, we write $C[p]$ for the term that results by placing p in the hole in $C[]$.

Transition Rules. Intuitively, closed BCCSP(A) terms represent finite process behaviours, where $\mathbf{0}$ does not exhibit any behaviour, $p+q$ is the nondeterministic choice between the behaviours of p and q, and ap executes action a to transform into p. This intuition is captured, in the style of Plotkin, by the transition rules below, which give rise to A-labelled transitions between closed terms.

$$\frac{}{ax \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}$$

The operational semantics is extended to open terms by assuming that variables do not exhibit any behaviour.

Linear Time - Branching Time Spectrum. Van Glabbeek [13] presented the linear time - branching time spectrum of behavioural preorders and equivalences; see Figure 1. The semantics in this spectrum are based on simulation notions and on decorated traces. In what follows, we use $\precsim$ to denote a preorder in this spectrum, and $\simeq$ to denote the corresponding equivalence (i.e., $\precsim \cap \precsim^{-1}$). The equivalence induced by a preorder is also known as its *kernel*. When we want to refer to a specific preorder in the spectrum, we shall subscribe the symbol $\precsim$ with the initials of the intended semantics. For instance, we shall use $\precsim_{\mathrm{RS}}$ to denote the ready simulation preorder, $\precsim_{\mathrm{S}}$ for the simulation preorder, $\precsim_{\mathrm{F}}$ for the failures preorder, $\precsim_{\mathrm{CT}}$ for the completed traces preorder, and $\precsim_{\mathrm{PT}}$ for the partial traces preorder. A similar notational convention applies to the kernels of the preorders.

Each preorder in the linear time - branching time spectrum is a *precongruence* over the algebra of closed BCCSP(A) terms. That is, $p_1 \precsim q_1$ and $p_2 \precsim q_2$ imply $ap_1 \precsim aq_1$, for each $a \in A$, and $p_1 + p_2 \precsim q_1 + q_2$. Likewise, the equivalences in the spectrum constitute a *congruence* over closed BCCSP(A) terms.

Given a preorder $\precsim$ over closed terms, for open terms t and u, we define $t \precsim u$ if $\rho(t) \precsim \rho(u)$ for each closed substitution ρ; the corresponding equivalence $\simeq$ is lifted to open terms likewise.

Equations and Inequations. An *(in)equational axiomatization* (often abbreviated to *axiomatization*) E is a collection of either inequations $t \preccurlyeq u$ or equations $t \approx u$, where t and u are BCCSP(A) terms. We write $E \vdash t \preccurlyeq u$ or $E \vdash t \approx u$ if this (in)equation can be derived from the (in)equations in E using the standard rules of (in)equational logic, where the rule for symmetry can be applied for equational derivations but not for inequational ones. An axiomatization E is *sound* modulo $\precsim$ (or $\simeq$) if, for all open terms t, u, from $E \vdash t \preccurlyeq u$ (or $E \vdash t \approx u$) it follows that $t \precsim u$ (or $t \simeq u$). An axiomatization E is *ground-complete* modulo $\precsim$ (or $\simeq$) if $p \precsim q$ (or $p \simeq q$) implies $E \vdash p \preccurlyeq q$ (or $E \vdash p \approx q$), for all closed terms p and q. We say that E is *ω-complete* if for all open terms t, u with $E \vdash \rho(t) \preccurlyeq \rho(u)$ (or $E \vdash \rho(t) \approx \rho(u)$) for all closed substitutions ρ, we have $E \vdash t \preccurlyeq u$ (or $E \vdash t \approx u$).

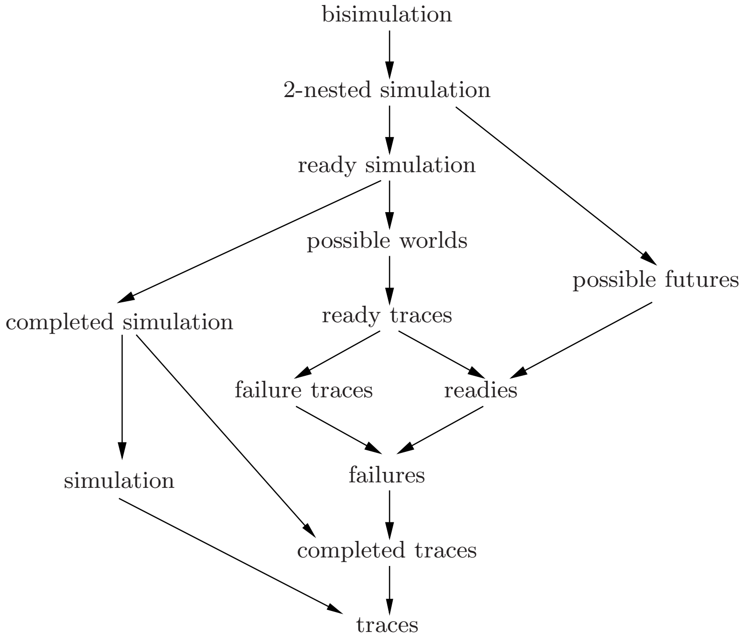


Fig. 1. The linear time - branching time spectrum

The core axioms A1–4 for BCCSP(A) given below are ω-complete [19], and sound and ground-complete [16,18] modulo bisimulation equivalence, which is the finest semantics in the linear time - branching time spectrum.

$$\begin{array}{ll} \text{A1} & x + y \approx y + x \\ \text{A2} & (x+y)+z \approx x+(y+z) \\ \text{A3} & x + x \approx x \\ \text{A4} & x + \mathbf{0} \approx x \end{array}$$

In the remainder of this paper, process terms are considered modulo A1–4. A term x or at is a *summand* of each term $x+u$ or $at+u$, respectively. We use *summation* $\sum_{i=1}^{n} t_i$ (with $n \geq 0$) to denote $t_1 + \cdots + t_n$, where the empty sum denotes $\mathbf{0}$. As binding convention, alternative composition and summation bind more weakly than prefixing. Modulo the equations A1–4 each BCCSP(A) term t can be written in the form $\sum_{i=1}^{n} t_i$, where each t_i is either a variable or is of the form at' for some action a and term t'.

In his paper [13], van Glabbeek offered, amongst a host of other results, (in)equational axiomatizations for the preorders and equivalences in the spectrum. The proofs of the completeness results in that reference mostly employ the method of graph transformations. Groote [14] obtained ω-completeness results for most of the axiomatizations, in case the alphabet of actions is infinite.

In the remainder of this paper, in case of an infinite alphabet, occurrences of action names in axioms should be interpreted as action variables.

3 Producing an Axiomatization

Consider a preorder $\precsim$ in the linear time - branching time spectrum that includes the ready simulation preorder. Let E be a sound and ground-complete inequational axiomatization for BCCSP(A) modulo $\precsim$. We give an algorithm to produce an axiomatization $\mathcal{A}(E)$ that is sound and ground-complete for BCCSP(A) modulo $\simeq$, namely the kernel of the preorder $\precsim$. Moreover, if E is ω-complete, then so is $\mathcal{A}(E)$.

Without loss of generality, we assume that the axioms A1–4 are present in E, together with the defining inequational axioms for ready simulation equivalence for each $a \in A$:

$$ax \preccurlyeq ax + ay \ .$$

The axiomatization $\mathcal{A}(E)$ is constructed as follows. The axioms A1–4 are by default included in $\mathcal{A}(E)$. Furthermore, for each inequational axiom $t \preccurlyeq u$ in E, we add to $\mathcal{A}(E)$:

A. $t + u \approx u$; and
B. $b(t+x) + b(u+x) \approx b(u+x)$ (for all $b \in A$, and some x that does not occur in $t + u$).

Note that $\mathcal{A}(E)$ is finite whenever A and E are finite. Moreover, using an action variable in step B in lieu of a concrete action $b \in A$, the axiomatization $\mathcal{A}(E)$ contains only finitely many axiom schemas when E does, even in the presence of an infinite collection of actions.

Remark 1. Since $ax \preccurlyeq ax + ay$ is assumed to be present in E for each $a \in A$, by step B of the algorithm, the defining axioms for ready simulation from [6], namely

$$b(ax + z) + b(ax + ay + z) \approx b(ax + ay + z) \ ,$$

are present in $\mathcal{A}(E)$, for all $a, b \in A$.

We are now ready to present the main result of the paper to the effect that the algorithm defined above delivers axiomatizations for the kernels of the preorders that are sound, and ground- or ω-complete.

Theorem 1. *Let $\precsim$ be a preorder in the linear time - branching time spectrum with $\precsim_{\mathrm{RS}} \subseteq \precsim$. Let E be a sound and ground-complete inequational axiomatization for* BCCSP(A) *modulo $\precsim$. Then the equational axiomatization $\mathcal{A}(E)$ is sound and ground-complete for* BCCSP(A) *modulo $\simeq$. Moreover, if E is ω-complete, then so is $\mathcal{A}(E)$.*

Since the algorithm presented above preserves finiteness of the axiomatization when the set of actions A is finite, it follows that each equivalence in the spectrum whose discriminating power lies in between that of ready simulation and partial traces equivalence is finitely axiomatizable over the language BCCSP(A) if so is its defining preorder.

The remainder of the paper will be essentially devoted to a proof of the above theorem. Our proof of Theorem 1 relies on the isolation of a collection of

equations, the so-called *cover equations*, that have a simple form and completely characterize the equational theory of BCCSP(A) modulo any of the behavioural equivalences whose discriminating power lies in between that of ready simulation and partial traces equivalence. Restricting ourselves to cover equations will help us overcome the technical complications in the proof-theoretic argument we shall use in Section 5 to complete the proof of Theorem 1.

In light of the key role cover equations play in the proof of Theorem 1, we now proceed to introduce them and to analyze the properties that make them a crucial ingredient in our proof of that result.

4 Cover Equations

For bisimulation semantics, and thus for all process semantics in the linear time - branching time spectrum, axiom A3 is sound. So if an equation $t \approx u$ is sound, then $u + t \approx t$ and $t + u \approx u$ are sound too; and from the last two equations one can derive $t \approx u$. Furthermore, for all process semantics in the linear time - branching time spectrum, if $t_1 + t_2 + u \approx u$ is sound, then $t_1 + u \approx u$ and $t_2 + u \approx u$ are sound; and from the last two equations one can derive $t_1 + t_2 + u \approx u$. Hence, from the point of view of provability, it suffices only to consider sound equations of the form $at + u \approx u$ and $x + u \approx u$. We call these the *cover equations*. We present three lemmas that limit the form that cover equations can have for the semantics in the spectrum we study in this paper. (In the statements of the lemmas below, t and u range over the collection of open BCCSP(A) terms.)

Lemma 1. *If $t + x \precsim u$, and either $\precsim \subseteq \precsim_{\mathrm{CT}}$, or $\precsim \subseteq \precsim_{\mathrm{PT}}$ and $|A| > 1$, then x is a summand of u.*

If $|A| = 1$, then the partial traces preorder and the simulation preorder coincide—see, e.g., [3]. For this special case, Lemma 1 fails. Namely, let $A = \{a\}$. Then $x \precsim ax$ is sound for the partial traces (and simulation) preorder.

Lemma 2. *Let $\simeq$ be an equivalence in the linear time - branching time spectrum. If $at + u + bv \simeq u + bv$ with $a \neq b$, then $at + u \simeq u$.*

This lemma is trivial to check for each of the equivalences in the linear time - branching time spectrum. The key idea is that since $a \neq b$, the non-empty (decorated) traces of at and bu are disjoint, and bu cannot (ready/completed) simulate at.

The following lemma states a kind of cancellation result for the preorders in the spectrum.

Lemma 3. *Let $\precsim$ be a preorder in the linear time - branching time spectrum. If $t + x \precsim u + x$, and x is not a summand of $t + u$, then $t \precsim u$.*

The condition in Lemma 3 that x is not a summand of $t + u$ is essential. For instance, $x + x \precsim_{\mathrm{PT}} \mathbf{0} + x$, but $x \not\precsim_{\mathrm{PT}} \mathbf{0}$. And $\mathbf{0} + x \precsim_{\mathrm{CT}} x + x$, but $\mathbf{0} \not\precsim_{\mathrm{CT}} x$.

Lemma 3 needs to be proved separately for each preorder in the linear time - branching time spectrum. Despite the naturalness of its statement, which appears obvious, these proofs are not trivial, and quite technical. Fokkink and Nain [10] proved such a lemma for failures semantics, with the aim to obtain an ω-completeness result for this semantics, and their proof is rather delicate. The details of the proof of Lemma 3 can be found in the full version of this paper [4].

From the three lemmas above, one can conclude that in order to prove ω-completeness (or ground-completeness) of an equational axiomatization, it suffices to derive all sound equations (or all sound closed equations) of the form

$$at + \sum_{i=1}^{n} au_i \approx \sum_{i=1}^{n} au_i \qquad (n \geq 1)$$

and, only for the case of partial traces semantics with $|A| = 1$, all sound equations of the form

$$x + u \approx u \ .$$

In our proof of Theorem 1, we shall therefore focus on showing that the equational axiomatization $\mathcal{A}(E)$ generated by our algorithm is powerful enough to prove all of the sound equations of the above two forms.

5 Proof of Theorem 1

Proof. Let $\precsim$ be a preorder in the linear time - branching time spectrum, with $\precsim_{\mathrm{RS}} \subseteq \precsim$. Let E be a sound and ground-complete inequational axiomatization for BCCSP(A) modulo $\precsim$.

It is not hard, albeit tedious, to see that the equational axiomatization $\mathcal{A}(E)$ is sound for BCCSP(A) modulo $\simeq$. We prove that ω-completeness of E implies ω-completeness of $\mathcal{A}(E)$. The proof that $\mathcal{A}(E)$ is ground-complete is identical, but assumes that all terms that occur in the proof below are closed. (It is well known that if an axiomatization proves a closed (in)equation, then there is a closed proof for that (in)equation.)

We note that, for each of the preorders in the linear time - branching time spectrum, $ar + as + t \precsim u$ if, and only if, both $ar + t \precsim u$ and $as + t \precsim u$. This, together with the presence of the axiom A3, implies that the inequational axiomatization E that we start with can be pre-processed so that there are no multiple a-summands on the left-hand sides of the inequational axioms in E.

Moreover, in view of Lemmas 1 and 3, if $\precsim \subseteq \precsim_{\mathrm{CT}}$ or $|A| > 1$, then variable summands on the left-hand sides of inequational axioms can be omitted. Concluding, in this case we can assume that the inequational axiomatization E that we start with only contains inequational axioms of the form $ap \preccurlyeq \sum_{i=1}^{n} aq_i$ (with $n \geq 1$) or $\mathbf{0} \preccurlyeq q$.

For the case of partial traces semantics with $|A| = 1$, Lemma 1 does not apply. Note, however, that $r + s \precsim_{\mathrm{PT}} u$ if, and only if, both $r \precsim_{\mathrm{PT}} u$ and $s \precsim_{\mathrm{PT}} u$. Hence, for this special case it suffices to allow also for inequational axioms of the form $x \preccurlyeq q$.

We start with showing that all cover equations of the form $at + u \approx u$ can be derived from $\mathcal{A}(E)$. (Cover equations of the form $x + u \approx u$ will be considered later.) In view of Lemmas 2 and 3, it suffices to only consider those equations where u is of the form $\sum_{i=1}^{n} au_i$ with $n \geq 1$. Let

$$at + \sum_{i=1}^{n} au_i \simeq \sum_{i=1}^{n} au_i \ .$$

We show that the corresponding cover equation can be derived from $\mathcal{A}(E)$. It is not hard to see that, for the semantics in the linear time - branching time spectrum, the above equivalence implies

$$at \precsim \sum_{i=1}^{n} au_i \ .$$

So by ω-completeness of E,

$$E \vdash at \preccurlyeq \sum_{i=1}^{n} au_i \ .$$

We prove, using induction on the length of such a derivation, not counting applications of axioms A1–4, that

$$\mathcal{A}(E) \vdash at + \sum_{i=1}^{n} au_i \approx \sum_{i=1}^{n} au_i \ .$$

Base case: $t = u_i$ for some i. Trivial using A1–3.

Inductive case: We distinguish two cases, which deal with instantiations of inequational axioms in context.

Case 1: The first step of the derivation is

$$E \vdash aC[p^\sigma] \preccurlyeq aC[q^\sigma] \ .$$

That is, $t = C[p^\sigma]$ for some context $C[]$, substitution σ, and inequational axiom $p \preccurlyeq q$. Then clearly $aC[p^\sigma]$ is of the form $D[b(p^\sigma + r)]$ and $aC[q^\sigma]$ is of the form $D[b(q^\sigma + r)]$ for some context $D[]$, action b, and term r.

Since $E \vdash aC[q^\sigma] \preccurlyeq \sum_{i=1}^{n} au_i$ by a shorter derivation, by induction,

$$\mathcal{A}(E) \vdash aC[q^\sigma] + \sum_{i=1}^{n} au_i \approx \sum_{i=1}^{n} au_i \ .$$

Furthermore,

$$\mathcal{A}(E) \vdash aC[p^\sigma] + aC[q^\sigma] \approx aC[q^\sigma] \ .$$

This equation can indeed be derived from the axiom $b(p+x)+b(q+x) \approx b(q+x)$, which is present in $\mathcal{A}(E)$ for each $b \in A$ according to step B in the algorithm, together with the defining axiom for ready simulation, $b(cx + z) + b(cx + cy + z) \approx b(cx + cy + z)$, which by assumption is present in $\mathcal{A}(E)$ for all $b, c \in A$

(see Remark 1). The derivation of the above equation is by induction on the depth of the occurrence of the context symbol [] within $C[]$.

- Let [] occur at depth zero in $C[]$, i.e., $C[] = [] + r$ for some term r. Let the substitution ρ coincide with σ on variables in p and q, and let $\rho(x) = r$. (Recall that an assumption in step B of the algorithm was that x does not occur in $p + q$.) The derivation simply consists of applying the substitution ρ to the axiom $a(p + x) + a(q + x) \approx a(q + x)$.
- Let $C[] = dC'[] + s$. By induction on the depth of the occurrence of [], $\mathcal{A}(E) \vdash dC'[p^\sigma] + dC'[q^\sigma] \approx dC'[q^\sigma]$. So

$$\begin{aligned}\mathcal{A}(E) \;\vdash\;\; & aC[p^\sigma] + aC[q^\sigma] \;=\; a(dC'[p^\sigma] + s) + a(dC'[q^\sigma] + s)\\ \approx\;\; & a(dC'[p^\sigma] + s) + a(dC'[p^\sigma] + dC'[q^\sigma] + s)\\ \approx\;\; & a(dC'[p^\sigma] + dC'[q^\sigma] + s)\\ \approx\;\; & a(dC'[q^\sigma] + s) \;=\; aC[q^\sigma] \ .\end{aligned}$$

Hence,

$$\begin{aligned}\mathcal{A}(E) \;\vdash\;\; & aC[p^\sigma] + \sum_{i=1}^{n} au_i \;\approx\; aC[p^\sigma] + aC[q^\sigma] + \sum_{i=1}^{n} au_i\\ \approx\;\; & aC[q^\sigma] + \sum_{i=1}^{n} au_i \;\approx\; \sum_{i=1}^{n} au_i \ ,\end{aligned}$$

which was to be shown.

CASE 2: The first step of the derivation is

$$E \vdash ap^\sigma \preccurlyeq \sum_{j=1}^{m} aq_j^\sigma \qquad (m \geq 1) \ .$$

That is, $t = p^\sigma$ for some substitution σ and inequational axiom $ap \preccurlyeq \sum_{j=1}^{m} aq_j$.

By the soundness of E, clearly $aq_j^\sigma \precsim \sum_{i=1}^{n} au_i$ for $j = 1, \ldots, m$. So by ω-completeness, $E \vdash aq_j^\sigma \preccurlyeq \sum_{i=1}^{n} au_i$ for $j = 1, \ldots, m$. By one of our assumptions, the inequational axioms in E do not contain multiple occurrences of a-summands on their left-hand sides. This implies that each of these derivations is not longer than the derivation of $E \vdash \sum_{j=1}^{m} aq_j^\sigma \preccurlyeq \sum_{i=1}^{n} au_i$. So by induction,

$$\mathcal{A}(E) \vdash aq_j^\sigma + \sum_{i=1}^{n} au_i \approx \sum_{i=1}^{n} au_i$$

for $j = 1, \ldots, m$. Furthermore, according to step A of the algorithm, the axiom $p + \sum_{j=1}^{m} aq_j \approx \sum_{j=1}^{m} aq_j$ is present in $\mathcal{A}(E)$. Hence,

$$\begin{aligned}\mathcal{A}(E) \;\vdash\;\; & ap^\sigma + \sum_{i=1}^{n} au_i \;\approx\; ap^\sigma + \sum_{j=1}^{m} aq_j^\sigma + \sum_{i=1}^{n} au_i\\ \approx\;\; & \sum_{j=1}^{m} aq_j^\sigma + \sum_{i=1}^{n} au_i \;\approx\; \sum_{i=1}^{n} au_i \ .\end{aligned}$$

This completes the proof for the case of cover equations of the form $at + \sum_{i=1}^{n} au_i \simeq \sum_{i=1}^{n} au_i$.

It remains to prove that cover equations of the form $x + u \approx u$ can be derived from $\mathcal{A}(E)$. If $\precsim \subseteq \precsim_{\mathrm{CT}}$ or $|A| > 1$, then in view of Lemma 1, such cover equations can be derived using A3. So we are left to consider the special case that $\precsim = \precsim_{\mathrm{PT}}$ and $|A| = 1$. Let

$$x + u \simeq_{\mathrm{PT}} u \enspace .$$

Clearly, this implies

$$x \precsim_{\mathrm{PT}} u \enspace .$$

So, by ω-completeness of E,

$$E \vdash x \preccurlyeq u \enspace .$$

We prove, using induction on the length of such a derivation, not counting applications of A1–4, that

$$\mathcal{A}(E) \vdash x + u \approx u \enspace .$$

Base case: x is a summand of u. Trivial.

Inductive case: The first step of the derivation is

$$E \vdash y^\sigma \preccurlyeq q^\sigma \enspace .$$

That is, $\sigma(y) = x$ for some substitution σ and inequational axiom $y \preccurlyeq q$ in E.

By the soundness of E, clearly $r \precsim_{\mathrm{PT}} u$ for each summand r of q^σ. So by ω-completeness, $E \vdash r \preccurlyeq u$. By assumption, the inequational axioms in E are all of the form $as \preccurlyeq \sum_{i=1}^{n} as_i$ (with $n \geq 1$) or $\mathbf{0} \preccurlyeq s$ or $z \preccurlyeq s$, for some variable z. This implies that each of these derivations is not longer than the derivation of $E \vdash q^\sigma \preccurlyeq u$. So by induction and A3,

$$\mathcal{A}(E) \vdash q^\sigma + u \approx u \enspace .$$

Furthermore, according to step A of the algorithm, the axiom $y + q \approx q$ is present in $\mathcal{A}(E)$. Hence,

$$\mathcal{A}(E) \ \vdash \ y^\sigma + u \ \approx \ y^\sigma + q^\sigma + u \ \approx \ q^\sigma + u \ \approx \ u \enspace .$$

The proof of the theorem is now complete. □

6 Examples

We show how our algorithm produces equational axiomatizations for two equivalences in the linear time - branching time spectrum—namely simulation and failures—from the inequational axiomatizations for the corresponding preorders. For the simulation preorder, we leave out the pre-supposed inequational axiom $ax \preccurlyeq ax + ay$, since it can be derived from the defining inequational axioms for that preorder.

6.1 Simulation

Let $|A| > 1$. Then A1–4 plus one inequational axiom

$$\mathbf{0} \preccurlyeq x$$

is a sound and ground-complete axiomatization for BCCSP(A) modulo the simulation preorder [13].

Step A of the algorithm produces the already present axiom A4:

$$\mathbf{0} + x \approx x \ .$$

Step B of the algorithm produces the defining axioms for simulation equivalence for each $b \in B$:

$$b(\mathbf{0} + y) + b(x + y) \approx b(x + y) \ .$$

6.2 Failures

Let $|A| \geq 1$. The axiomatization consisting of A1–4 plus one inequational axiom

$$a(x + y) \preccurlyeq ax + a(y + z)$$

for each $a \in A$ is sound and ground-complete for BCCSP(A) modulo the failures preorder [13].

Step A of the algorithm produces, for all $a \in A$:

$$a(x + y) + ax + a(y + z) \approx ax + a(y + z) \ .$$

This axiom is one of the two defining axioms for failures equivalence. (The second defining axiom for failures equivalence is the ready simulation axiom, which is assumed to be present from the start.)

Step B of the algorithm produces, for all $a, b \in A$:

$$b(a(x + y) + w) + b(ax + a(y + z) + w) \approx b(ax + a(y + z) + w) \ .$$

This axiom is redundant; it can be derived from the other axioms as follows. (The subterm to which an axiom is applied is underlined.)

$$\begin{array}{ll} & b(\underline{ax + a(y + z)} + w) \\ \approx & \underline{b(a(x + y) + ax + a(y + z) + w)} \\ \approx & \underline{b(a(x + y) + a(y + z) + w)} + b(a(x + y) + ax + a(y + z) + w) \\ \approx & b(a(x + y) + w) + \underline{b(a(x + y) + a(y + z) + w) + b(a(x + y) + ax + a(y + z) + w)} \\ \approx & b(a(x + y) + w) + b(\underline{a(x + y) + ax + a(y + z)} + w) \\ \approx & b(a(x + y) + w) + b(ax + a(y + z) + w) \end{array}$$

7 Conclusions and Comparison with Related Work

In this paper, we have offered an algorithm for generating a ground-complete (respectively, ω-complete) axiomatization for behavioural equivalences in the linear time - branching time spectrum starting from a ground-complete (respectively, ω-complete) axiomatization for their underlying preorders—that is, of the preorders that have the equivalences as their kernels. Our algorithm applies to all of the process semantics in the spectrum whose discriminating power lies in between that of ready simulation semantics and of partial traces semantics. Moreover, in the presence of a finite set of actions, our procedure preserves finiteness of axiomatizations, and thus can be used to obtain automatically finite basis results for behavioural equivalences in the spectrum from similar results for their underlying preorders. In fact, our results apply to *any* behavioural precongruence whose discriminating power lies in between that of the ready simulation preorder and of the partial traces preorder, provided that Lemmas 1–3 hold for the precongruence in question.

Our algorithm may thus be considered as isolating and axiomatizing the ingredients that all of the extant proofs of completeness results for the class of behavioural equivalences we study have in common. (See, for example, the references [5,6,8,9,10,13,14] for a sample of such results.) It also eliminates the need to reprove, essentially from scratch, completeness results for a large fragment of behavioural equivalences in the spectrum once a completeness result has been obtained for their underlying preorders. As witnessed by the examples we provided in Section 6, the axiomatizations that are automatically generated by our algorithm are very similar, when not identical, to those presented in the literature. In this respect, this study may be seen as a companion to [1]. That paper offered an algorithm that generates a finite, ground-complete axiomatization for bisimulation equivalence from an operational specification of a language in GSOS format [7]. That procedure relies on the axiomatization of bisimulation equivalence over the language BCCSP. Here we have focused on the algorithmic generation of complete axiomatizations for other equivalences in the spectrum over the language BCCSP.

The spirit of our study is also very similar to the one in [12]. In that reference, independent of our work and building on their previous paper [11], de Frutos-Escrig and Gregorio-Rodríguez show, amongst other things, how to generate an inequational axiomatization for preorders in the spectrum from equational axiomatizations for the corresponding equivalence. They generate this inequational axiomatization by simply adding the defining inequational axioms for the ready simulation preorder to the axiomatization for the equivalence—see Theorem 5.1 in [12]. That result applies to behavioural equivalences in the linear time - branching time spectrum that (1) include ready simulation equivalence, and (2) whose underlying preorders only equate processes having the same set of initial actions. That second condition is not met by completed simulation, simulation, completed traces and partial traces semantics. Furthermore, the result from [12] only applies to ground-complete axiomatizations.

There are some interesting general connections between the technical developments in this paper and those in [12]. For instance, Lemma 3.11 in [12] gives a soundness proof for the equations generated by step A in our algorithm for the preorders in the spectrum that satisfy condition 2 above. However, the equations generated by step A are sound also for completed simulation, simulation, completed traces and partial traces semantics. So Lemma 3.11 in [12] is not as general as it could be.

It would also be interesting to investigate the possible relation between the cover equations approach, used in this paper to reduce the class of equations to be considered in the proof of completeness, and the condition of action factorization mentioned in the statement of Theorem 2.6 of [12]. (Action factorization means that if $p \precsim q$, then, for each action a, the sum of the a-summands of p is also dominated by the sum of the a-summands of q with respect to $\precsim$.)

In summary, our work differs from [12] in the following fundamental ways.

– We show how to produce an equational axiomatization for an equivalence from an inequational axiomatization of its underlying preorder. Since the equivalences in the linear time - branching time spectrum that include ready simulation equivalence are the kernels of their underlying preorders, to our mind, the preorders are a more basic notion to build on in this setting.
– Unlike Theorem 5.1 of [12], our main result applies to all of the semantics in the spectrum whose discriminating power lies in between that of ready simulation semantics and partial traces semantics.
– Unlike Theorem 5.1 of [12], our results apply to ω-complete as well as to ground-complete axiomatizations.

It would be interesting to extend our algorithm so that it applies also to nested simulation semantics [15] and to possible futures semantics [20]. However, as shown in [2], unlike the semantics we have considered in this study, nested simulation and possible futures semantics afford no finite ground-complete axiomatization over BCCSP even in the presence of a single action. This indicates that such a generalization of our results will not be easy to achieve without recourse to conditional equations. We leave such generalizations of our results and proof techniques as a topic for future investigations.

References

1. Aceto, L., Bloom, B., Vaandrager, F.W.: Turning SOS rules into equations. Information and Computation 111(1), 1–52 (1994)
2. Aceto, L., Fokkink, W., van Glabbeek, R.J., Ingolfsdottir, A.: Nested semantics over finite trees are equationally hard. Information and Computation 191(2), 203–232 (2004)
3. Aceto, L., Fokkink, W., Ingolfsdottir, A.: A menagerie of non-finitely based process semantics over BPA*—from ready simulation to completed traces. Mathematical Structures in Computer Science 8(3), 193–230 (1998)
4. Aceto, L., Fokkink, W., Ingolfsdottir, A.: Ready to preorder: Get your BCCSP axiomatization for free! Report RS-07-3, BRICS Research Series (2007)

5. Aceto, L., Fokkink, W., Ingolfsdottir, A., Luttik, B.: Finite equational bases in process algebra: Results and open questions. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 338–367. Springer, Heidelberg (2005)
6. Blom, S., Fokkink, W., Nain, S.: On the axiomatizability of ready traces, ready simulation and failure traces. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 109–118. Springer, Heidelberg (2003)
7. Bloom, B., Istrail, S., Meyer, A.: Bisimulation can't be traced. Journal of the ACM 42, 232–268 (1995)
8. Chen, T., Fokkink, W.: On finite alphabets and infinite bases III: Simulation. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 421–434. Springer, Heidelberg (2006)
9. Chen, T., Fokkink, W., Nain, S.: On finite alphabets and infinite bases II: Completed and ready simulation. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006 and ETAPS 2006. LNCS, vol. 3921, pp. 1–15. Springer, Heidelberg (2006)
10. Fokkink, W., Nain, S.: A finite basis for failure semantics. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 755–765. Springer, Heidelberg (2005)
11. de Frutos-Escrig, D., Gregorio-Rodríguez, C.: Bisimulations up-to for the linear time branching time spectrum. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 278–292. Springer, Heidelberg (2005)
12. de Frutos-Escrig, D., Gregorio-Rodríguez, C.: Simulations up-to and canonical preorders (extended abstract). In: Proc. SOS'07, ENTCS, Elsevier (to appear)
13. van Glabbeek, R.J.: The linear time – branching time spectrum I. The semantics of concrete, sequential processes. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 3–99. Elsevier, Amsterdam (2001)
14. Groote, J.F.: A new strategy for proving ω-completeness with applications in process algebra. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 314–331. Springer, Heidelberg (1990)
15. Groote, J.F., Vaandrager, F.W.: Structured operational semantics and bisimulation as a congruence. Information and Computation 100, 202–260 (1992)
16. Hennessy, M.C.B., Milner, R.: Algebraic laws for nondeterminism and concurrency. Journal of the ACM 32(1), 137–161 (1985)
17. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (1985)
18. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
19. Moller, F.: Axioms for Concurrency. PhD thesis, Department of Computer Science, University of Edinburgh (July 1989)
20. Rounds, W., Brookes, S.: Possible futures, acceptances, refusals and communicating processes. In: Proc. FOCS'81, pp. 140–149. IEEE Computer Society Press, Los Alamitos (1981)

Impossibility Results for the Equational Theory of Timed CCS*

Luca Aceto[1], Anna Ingólfsdóttir[1], and MohammadReza Mousavi[1,2]

[1] Department of Computer Science, Reykjavík University, Kringlan 1, IS-103, Reykjavík, Iceland
[2] Department of Computer Science, Eindhoven University of Technology, NL-5600MB Eindhoven, The Netherlands

Abstract. We study the equational theory of Timed CCS as proposed by Wang Yi in CONCUR'90. Common to Wang Yi's paper, we particularly focus on a class of linearly-ordered time domains exemplified by the positive real or rational numbers. We show that, even when the set of basic actions is a singleton, there are parallel Timed CCS processes that do not have any sequential equivalent and thus improve on the Gap Theorem for Timed CCS presented by Godskesen and Larsen in FSTTCS'92. Furthermore, we show that timed bisimilarity is not finitely based both for single-sorted and two-sorted presentations of Timed CCS. We further strengthen this result by showing that, unlike in some other process algebras, adding the untimed or the timed left-merge operator to the syntax and semantics of Timed CCS does not solve the axiomatizability problem.

1 Introduction

In [12], Wang Yi proposed Timed CCS (TCCS) as a possible timed extension of Milner's CCS [7]. (See [10] for another timed extension of CCS.) There, he gave syntax and operational semantics of the calculus as well as a number of equational laws, including a form of *expansion law* that allows one to resolve parallelism and transform parallel processes into a nondeterministic composition of sequential processes.

However, it turned out that the expansion law of [12] is not sound with respect to timed bisimilarity [5,13]. In [13], Wang Yi put forward an alternative correct version of the expansion law of [12]. However, the correction involved the introduction of the so-called *time variables* and a substantially more complicated calculus. A natural question was then whether there is a sound expansion theorem for the simple calculus of [12] and whether the calculus of [12] affords a finite complete (respectively, ω-complete) axiomatization.

The former question was answered negatively in [5] by showing that for all $n > 0$ there are expressions with $n + 1$ parallel components for which there are

* The work of the authors has been partially supported by the project "The Equational Logic of Parallel Processes" (nr. 060013021) of The Icelandic Research Fund.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 80–95, 2007.
© Springer-Verlag Berlin Heidelberg 2007

no bisimilar terms with n parallel components or less (Gap Theorem). In other words, parallel composition cannot in general be eliminated from TCCS terms.

The latter question has been addressed partially in [14] and [1], which present complete axiomatizations for the finite [14] and regular [1] fragments of TCCS, respectively. However, it has remained an open question whether the full calculus, including parallel composition, affords a finite (ω-)complete axiomatization or not.

The aim of this paper is to re-visit this question. We show that different presentations of TCCS cannot be finitely axiomatized modulo timed bisimilarity. We further strengthen this result by showing that, unlike in some other process algebras, adding the left-merge operator (timed or un-timed) to the theory of TCCS does not solve the axiomatizability problem. We also present an improved version of the gap theorem and show that even in the presence of a single action, parallel composition cannot be resolved in TCCS.

The rest of this paper is organized as follows. In Section 2, we present the basic definitions concerning TCCS, timed bisimilarity and equational logic. Section 3 is devoted to the single-sorted presentation of TCCS and shows that it cannot be finitely axiomatized modulo timed bisimilarity. In Section 4, we study the two-sorted presentation of TCCS and show, first of all, that, even in the presence of just one action, parallelism cannot be resolved in TCCS (Gap Theorem). We also prove that the theory of the two-sorted presentation of TCCS cannot be finitely axiomatized either. Section 5 studies the addition of the untimed as well as the timed lef-merge operator to TCCS and shows that adding neither of these operators solves the axiomatizability problem. Section 6 concludes the paper and presents directions of ongoing and future research.

2 Preliminaries

2.1 TCCS: Syntax and Semantics

Following [6], we define a monoid $(X, +, 0)$ to be:

- *left-cancellative* iff $(x + y = x + z) \Rightarrow (y = z)$, and
- *anti-symmetric* iff $(x + y = 0) \Rightarrow (x = y = 0)$.

We define a partial order on X as $x \leq y$ iff $x + z = y$ for some $z \in X$. A *time domain* is a left-cancellative anti-symmetric monoid $(D, +, 0)$ such that $\leq$ is a total order. If $d_0 \leq d_1$, then we write $d_1 - d_0$ for the unique d such that $d_1 = d_0 + d$. A time domain is *non-trivial* if D contains at least two elements. Note that every non-trivial time domain does not have a largest element. A time domain has 0 as *cluster point* iff for each $d \in D$ such that $d \neq 0$ there is a $d' \in D$ such that $0 < d' < d$. In the remainder of the paper, we assume that our time domain, denoted henceforth by D, is non-trivial and has 0 as a cluster point.

The syntax of TCCS *processes* (closed terms) is given by the following grammar.

$$t ::= 0 \mid \mu.t \mid \epsilon(d).t \mid s + t \mid s \,||\, t$$

In the grammar given above, 0 stands for the deadlocking process (not to be confused with 0 in the time domain), $\mu._$ represents action-prefix operators for $\mu \in A$ where A is the set of (delayable) actions. Given a time domain D, $\epsilon(d)._$ is an operator for each $d \in D$, and represents a time delay of length at least d before proceeding with the remaining process. For the sake of simplicity, we assume that all delays are non-zero; $\epsilon(0).t$ can be interpreted as a syntactic sugar for t, but we avoid zero delays altogether throughout the rest of this paper. Nondeterministic choice is denoted by $+$ and parallel composition is denoted by $||$.

We write μ for $\mu.0$ and $\mu(d)$ for $\epsilon(d).\mu$. Our proofs, in the remainder of this paper, remain sound even when the set A of actions is a singleton $\{a\}$. We write a^n to stand for 0 if $n = 0$, and $a.a^{n-1}$, otherwise.

TCCS (open) *terms* are constructed inductively using the operators of the syntax and a countably infinite set of variables V, with typical members x, x_0, y, y_0, ... The *size* of a term is its length in symbols. The set of variables appearing in term t is denoted by $vars(t)$. A substitution σ is a function from variables to TCCS terms. The range of a *closing substitution* is the set of TCCS processes. The domain of a substitution is lifted naturally from variables to terms.

We take two different approaches to formalizing the syntax of TCCS in a term algebra.

1. The first approach is to use a single-sorted algebra with the only available sort representing processes. Then, both $a._$ and $\epsilon(d)._$ are sets of unary operators for each $a \in A$ and $d \in D$.
2. The other approach is to take two different sorts, one for time and one for processes, denoted by $\mathbb{T}$ and $\mathbb{P}$, respectively. Then, $\epsilon(_)$ is a single function symbol with arity $\mathbb{T} \times \mathbb{P} \rightarrow \mathbb{P}$. When using this approach, we use d, d', d_0, ... as variables of sort $\mathbb{T}$ and closing substitutions map variables of sort $\mathbb{T}$ to elements of the time domain D.

The Plotkin-style rules defining the operational semantics of TCCS are given below.

$$\frac{}{0 \xrightarrow{\epsilon(d)} 0} \qquad \frac{}{\mu.x \xrightarrow{\mu} x} \qquad \frac{}{\mu.x \xrightarrow{\epsilon(d)} \mu.x}$$

$$\frac{}{\epsilon(d).x \xrightarrow{\epsilon(d)} x} \qquad \frac{}{\epsilon(d+e).x \xrightarrow{\epsilon(d)} \epsilon(e).x} \qquad \frac{x \xrightarrow{\epsilon(e)} y}{\epsilon(d).x \xrightarrow{\epsilon(d+e)} y}$$

$$\frac{x_0 \xrightarrow{\mu} y}{x_0 + x_1 \xrightarrow{\mu} y} \qquad \frac{x_1 \xrightarrow{\mu} y}{x_0 + x_1 \xrightarrow{\mu} y} \qquad \frac{x_0 \xrightarrow{\epsilon(d)} y_0 \quad x_1 \xrightarrow{\epsilon(d)} y_1}{x_0 + x_1 \xrightarrow{\epsilon(d)} y_0 + y_1}$$

$$\frac{x_0 \xrightarrow{\mu} y_0}{x_0 \,||\, x_1 \xrightarrow{\mu} y_0 \,||\, x_1} \qquad \frac{x_1 \xrightarrow{\mu} y_1}{x_0 \,||\, x_1 \xrightarrow{\mu} x_0 \,||\, y_1} \qquad \frac{x_0 \xrightarrow{\epsilon(d)} y_0 \quad x_1 \xrightarrow{\epsilon(d)} y_1}{x_0 \,||\, x_1 \xrightarrow{\epsilon(d)} y_0 \,||\, y_1}$$

The rules above, define two types of transition relations: $\xrightarrow{a}$, where $a \in A$, for action transitions and $\xrightarrow{\epsilon(d)}$, where $d \in D$, for time-delay transitions. We use $\xrightarrow{a^n}$ to denote n consecutive a-transitions (whose intermediate processes are irrelevant). The following lemma lists some interesting properties of the semantics of TCCS.

Lemma 1. *The following statements hold for each process p and time delay d.*

1. *There exists a unique process p_d such that $p \stackrel{\epsilon(d)}{\rightarrow} p_d$.*
2. *If p does not contain parallel composition and $p \stackrel{a}{\rightarrow} p'$, then $p_d \stackrel{a}{\rightarrow} p'$, where p_d is defined above.*

The notion of equivalence over TCCS that we are interested in is the following notion of timed bisimilarity.

Definition 2. *A symmetric relation R on TCCS processes is a timed bisimulation relation when for all $(p, q) \in R$,*

1. *for all actions a and processes p', if $p \stackrel{a}{\rightarrow} p'$ then there exists a process q' such that $q \stackrel{a}{\rightarrow} q'$ and $(p', q') \in R$;*
2. *for all time delays d and processes p', if $p \stackrel{\epsilon(d)}{\rightarrow} p'$ then there exists a process q' such that $q \stackrel{\epsilon(d)}{\rightarrow} q'$ and $(p', q') \in R$.*

Two processes p and q are timed bisimilar, or just bisimilar, denoted by $p \underline{\leftrightarrow} q$ when there exists a timed bisimulation relation R such that $(p, q) \in R$.

The notion of bisimilarity generalizes naturally to open terms: s and t are bisimilar, when $\sigma(s) \underline{\leftrightarrow} \sigma(t)$ for each closing substitution σ.

It is well-known that timed bisimilarity is a congruence over TCCS [12].

We define the following notion of time insensitive processes and show that a TCCS process is time insensitive if and only it is bisimilar to a CCS process.

Definition 3. *A TCCS process p is* initially time insensitive *when for all d, if $p \stackrel{\epsilon(d)}{\rightarrow} p_d$, then $p_d \underline{\leftrightarrow} p$.*

The set of time insensitive *processes is the largest set P_{TI} of TCCS processes such that, whenever $p \in P_{TI}$, (i) p is initially time-insensitive, and (ii) if $p \stackrel{a}{\rightarrow} p_a$ then $p_a \in P_{TI}$ is time insensitive for each $a \in A$ and each process p_a.*

For example, $a(d)$ is not (initially) time insensitive, $a.a(d)$ is initially time insensitive but not time insensitive, and $a.\epsilon(d).0$ is (initially) time insensitive.

Theorem 4. *A TCCS process p is time insensitive if and only if there exists a process q such that $p \underline{\leftrightarrow} q$ and q does not contain time-delay prefixing operators, i.e., q is a CCS process.*

2.2 Equational Theory

Given a signature Σ, a set E of equations $t = t'$, where t and t' are terms (of the same sort), is called an *axiom system*.

We write $E \vdash t = t'$ when $t = t'$ is derivable from E by the following set of deduction rules. Deduction rule is a rule schema for each operator f in the signature.

$$\frac{}{E \vdash t = t} \quad \frac{E \vdash t = t'}{E \vdash t' = t} \quad \frac{E \vdash t_0 = t_1 \quad E \vdash t_1 = t_2}{E \vdash t_0 = t_2}$$

$$\frac{E \vdash t_0 = t'_0 \quad \ldots \quad E \vdash t_n = t'_n}{E \vdash f(t_0, \ldots, t_n) = f(t'_0, \ldots, t'_n)} \quad \frac{t = t' \in E}{E \vdash \sigma(t) = \sigma(t')}$$

Without loss of generality, we assume that E is closed under symmetry, i.e., $t = t' \in E$ if and only if $t' = t \in E$, so that need not be considered in proofs. It is well-known that if an equation relating two closed terms can be proven from an axiom system E, then there is a closed proof for it.

An equation $t = t'$ is *sound* (modulo timed bisimilarity) if the terms t and t' are timed bisimilar. An axiom system is sound if each of its equations is sound. An example of a collection of equations from [12] that are sound with respect to timed bisimilarity is given below. The axioms A4, M1 and D1 (used from left to right) are enough to establish that each TCCS term that is bisimilar to 0 is also provably equal to 0. Thus, in the technical developments from Section 4 onwards, we shall assume, without loss of generality, that each axiom system we consider includes the equations given below. This assumption means, in particular, that our axiom systems allows us to identify each term that is bisimilar to 0 with 0.

A1	$x + y = y + x$	A2	$(x + y) + z = x + (y + z)$
A3	$x + x = x$	A4	$x + 0 = x$
M1	$0 \,\|\|\, x = 0$	M2	$x \,\|\|\, 0 = x$
D1	$\epsilon(d).0 = 0$	D2	$\epsilon(d).(x + y) = \epsilon(d).x + \epsilon(d).y$
D3	$\epsilon(d).(x \,\|\|\, y) = \epsilon(d).x \,\|\|\, \epsilon(d).y$	D4	$\epsilon(d).\epsilon(d').x = \epsilon(d + d').x$
P	$a.x = a.x + \epsilon(d).a.x$		

Henceforth, process terms are considered modulo associativity and commutativity of $+$ and $||$. We use a *summation* and *a product*, denoted by $\sum_{i \in \{1,\ldots,k\}} s_i$ and $\prod_{j \in \{1,\ldots,k'\}} t_j$, to stand for $s_1 + \cdots + s_k$ and $t_1 \,||\, \cdots \,||\, t_{k'}$, respectively, where the empty sum and product represent 0. We say that a term t has a 0 *factor* if it contains a subterm of the form $\prod_{j \in \{1,\ldots,k'\}} t_j$, where some t_j is bisimilar to 0. It is easy to see that, modulo the equations given above, every TCCS term s can be written as $\sum_{i \in I} s_i$, for some finite index set I, and terms s_i $(i \in I)$ that are not 0 and do not have themselves the form $s' + s''$, for some terms s' and s''. The terms s_i $(i \in I)$ will be referred to as the *summands* of t. Again modulo the equations given above, each s_i can be assumed to have no 0 factors.

3 Single-Sorted TCCS

In this section, as a warm up for the more complex results to follow, we show that single-sorted TCCS has no finite basis provided that the time domain is infinite. (Note that each time domain D that we consider in this paper does not have a largest element and is therefore infinite.)

Theorem 5. *If time domain D is infinite, then bisimilarity over single-sorted TCCS has no finite basis.*

We start with proving the following lemma which implies the above theorem.

Lemma 6. *Assume that E is a finite axiom system that is sound modulo bisimilarity. Let d be greater than the maximal delay prefixing mentioned in terms in E. For all provable equations $t = u$ such that either t or u contain $\epsilon(d)$, then both t and u contain $\epsilon(d)$.*

Proof. To prove Lemma 6, we proceed by an induction on the derivation structure for $E \vdash t = u$ and make a case distinction based on the last deduction rule applied to derive $E \vdash t = u$. The cases for , and are either trivial or follow immediately from the induction hypothesis. The most involved case is when the last deduction rule is .

For a TCCS process p, we define the action depth of p, denoted by $adepth(p)$, as the length of the maximal action trace that p affords (by omitting the time-delay transitions in between). It then follows that, for any two TCCS terms s and t, if $s \underline{\leftrightarrow} t$ then $adepth(\sigma(s)) = adepth(\sigma(t))$ for all closing substitutions σ.

Lemma 7. *Let t, u be bisimilar TCCS terms. Then $vars(t) = vars(u)$.*

Proof. Assume $x \in vars(t) \setminus vars(u)$. Construct a substitution σ that maps x to a^n, for some n larger than the sizes of both t and u, and all other variables to 0. Then, $adepth(\sigma(t)) \geq n > adepth(\sigma(u))$ and hence t and u are not bisimilar.

We are now ready to complete the proof of Lemma 6. Assume that $t' = u' \in E$, $t = \sigma(t')$, $u = \sigma(u')$ and $\epsilon(d)$ occurs in t. Since d is greater than the largest constant appearing in E, neither t' nor u' contain occurrences of $\epsilon(d)$. Thus, there exists a variable $x \in vars(t')$ such that $\sigma(x)$ has an occurrence of $\epsilon(d)$. By Lemma 7 and the soundness of the equation $t' = u'$ modulo bisimilarity, $x \in vars(u')$. Thus, $\sigma(u')$ also contains $\sigma(x)$ as a subterm, which in turn has an occurrence of $\epsilon(d)$. ⊠

Proof of Theorem 5. Assume that single-sorted TCCS affords a finite complete axiomatization E and d is greater than the largest delay appearing in E. (If no element of D appears in terms in E then let d be an arbitrary element of D.) Axiom D1 is sound. However, it follows from Lemma 6 that the instance of D1 for $d \in D$ is not derivable from E and thus Theorem 5 follows. ⊠

The lesson to be drawn from the above result is that, in the presence of an infinite time domain, when studying the equational theory of TCCS, it is much more natural to consider a two-sorted presentation of the calculus. For this reason, the rest of this paper is devoted to the study of the equational theory of two-sorted TCCS.

4 Two-Sorted TCCS

4.1 Gap Theorem

In this section, we present and prove the so-called *gap theorem* for TCCS, originally offered in [5], which shows that parallel composition cannot be eliminated

in general from TCCS terms. Our presentation and the proof of this theorem improves on that of [5] in two ways; first, our version of the gap theorem holds even in the presence of a single action while the gap theorem of [5] requires the presence of countably many different actions. Secondly, our proof is purely process algebraic in nature while the proof of [5] goes through a translation of TCCS to timed automata [2] and the argument is based on the number of clocks in the translated timed automata.

Theorem 8. *Define p as $a(d_0) \,||\, \prod_{i\in\{1,\ldots,n\}} a.a(d_i)$, for some action $a \in A$, positive integer n and delays $d_0, d_1, \ldots, d_n \in D$. There exists no q such that $p \,\underline{\leftrightarrow}\, q$ and $q = \sum_{j\in J} \prod_{i\in\{1,\ldots,n_j\}} q_{ij}$ where $n_j \leq n$ and q_{ij} does not contain parallel composition.*

Informally, the above theorem states that for all $n > 0$, there are TCCS processes with $n+1$ parallel components which do not have any bisimilar counterpart with (summands comprising) n or fewer parallel components. *Proof.* Assume, towards a contradiction, that $p \,\underline{\leftrightarrow}\, q$ and $q \equiv \sum_{j\in J} \prod_{i\in\{1,\ldots,n_j\}} q_{ij}$. By the definition of p, we have that $p \stackrel{a^n}{\rightarrow} p' \equiv \prod_{i\in\{0,\ldots,n\}} a(d_i)$. Hence there should exist a $j \in J$ such that $q_j \equiv \prod_{i\in\{1,\ldots,n_j\}} q_{ij} \stackrel{a^n}{\rightarrow} q'$ for some q' such that $q' \,\underline{\leftrightarrow}\, \prod_{i\in\{0,\ldots,n\}} a(d_i)$. Then, either all parallel components of q_j contribute exactly one action to the trace a^n or there exists a component in q_j that contributes more than one action to a^n. Next, we analyze these two possibilities and show that both lead to a contradiction.

1. Assume that all parallel components of q_j contribute exactly one action to the trace a^n, i.e., $n_j = n$, $q' = \prod_{i\in\{1,\ldots n\}} q'_{ij}$, for some q'_{ij} such that for all $i \leq n$, $q_{ij} \stackrel{a}{\rightarrow} q'_{ij}$, and $\prod_{i\in\{0,\ldots n\}} a(d_i) \,\underline{\leftrightarrow}\, \prod_{i\in\{1,\ldots,n\}} q'_{ij}$.
   Since D has 0 as a cluster point, there is a $d' \in D$ such that $0 < d' < d_0$. It follows from Lemma 1.(2) that $q_{ij} \stackrel{\epsilon(d')}{\rightarrow} q''_{ij} \stackrel{a}{\rightarrow} q'_{ij}$; thus, $q \stackrel{\epsilon(d')}{\rightarrow} q'' \stackrel{a^n}{\rightarrow} q'$. Furthermore, $p \stackrel{\epsilon(d')}{\rightarrow} p'' \equiv a(d_0 - d') \,||\, \prod_{i\in\{1,\ldots n\}} a.a(d_i)$ and it should hold that $p'' \,\underline{\leftrightarrow}\, q''$. However, $p'' \stackrel{a^n}{\rightarrow} a(d_0 - d') \,||\, \prod_{i\in\{1,\ldots n\}} a(d_i)$ (as $d' < d_0$, this is the only a^n-derivative of p''), which is clearly not bisimilar to $\prod_{i\in\{0,\ldots n\}} a(d_i)$, and hence, not bisimilar to q'.
2. Assume that there is a component in q_j that contributes more than one action to a^n, i.e., there exists an $l \in \{1, \ldots, n_j\}$ such that $q_{lj} \stackrel{a^{k-2}}{\rightarrow} q_{a^{k-2}lj} \stackrel{a}{\rightarrow} q_{a^{k-1}lj} \stackrel{a}{\rightarrow} q_{a^k lj}$ for some $k > 1$ and for some $q_{a^{k-2}lj}$, $q_{a^{k-1}lj}$ and $q_{a^k lj}$. For notational convenience, we assume that $k = 2$ but the proof technique can easily be adapted for $k > 2$. Note that by Lemma 1.(2) $q_{lj} \stackrel{a}{\rightarrow} q_{alj} \stackrel{\epsilon(d')}{\rightarrow} q_{ad'lj} \stackrel{a}{\rightarrow} q_{ad'alj} \equiv q_{a^2lj}$ for an arbitrary d'.
   It follows from the semantics of parallel composition and nondeterministic choice that $q \stackrel{a^{n-1}}{\rightarrow} q_{a^{n-1}} \equiv q_{alj} \,||\, \prod_{i\in\{1,\ldots,n_j\}\setminus\{l\}} q'_{ij}$ and $q_{a^{n-1}} \underline{\leftrightarrow}\, a.a(d_m) \,||\, \prod_{i\in\{0,\ldots,n\}\setminus\{m\}} a(d_i)$ for some $0 < m \leq n$. From the above bisimilarity, we have that, for any d' such that $d' > d_i$, for each $i \leq n$,

$q_{a^{n-1}} \stackrel{\epsilon(d')}{\rightarrow} q_{a^{n-1}d'}$ for some $q_{a^{n-1}d'} \equiv q_{ad'lj} \,||\, \prod_{i \in \{1,\ldots,n_j\} \setminus \{l\}} q'_{d'ij}$ such that $q_{a^{n-1}d'} \leftrightarrow a.a(d_m) \,||\, \prod_{i \in \{0,\ldots,n-1\}} a$. Furthermore, $q_{a^{n-1}d'}$ can make one further a-transition, due to $q_{ad'lj}$ resulting in some $q_{a^n d'}$ such that $q_{a^n d'} \leftrightarrow a(d_m) \,||\, \prod_{i \in \{0,\ldots,n-1\}} a$ or $q_{a^n d'} \leftrightarrow a.a(d_m) \,||\, \prod_{i \in \{0,\ldots,n-2\}} a$. It follows from the aforementioned bisimilarities that $q_{a^n d'} \stackrel{a^n}{\rightarrow} q' \leftrightarrow a(d_m)$.
Either all a-transitions in the latter a^n-trace are due to $q_{d'ij}$ with $i \neq l$, or some of them are performed by $q_{ad'alj} \equiv q_{a^2 lj}$.
In the former case, then $q_{a^{n-2}} \equiv q_{lj} \,||\, \prod_{i \in \{1,\ldots,n_j\} \setminus \{l\}} q'_{ij} \stackrel{\epsilon(d')}{\rightarrow} \stackrel{a^{n+2}}{\rightarrow} \overline{q}$ for some $\overline{q} \leftrightarrow a(d_m)$ since, first, $\prod_{i \in \{1,\ldots,n_j\} \setminus \{l\}} q'_{d'ij}$ can make n consecutive a-transitions, and $q_{d'lj}$ can make two a-transition afterwards. However, p cannot mimic this behavior, i.e., $\stackrel{a^{n-1}}{\rightarrow} \stackrel{\epsilon(d')}{\rightarrow} \stackrel{a^{n+2}}{\rightarrow}$, for it has only $n+1$ a-transitions enabled after an initial a^{n-2} trace and a time delay of d', which results in some process bisimilar to $a(d_m) \,||\, a(d'_m)$, for some $m' \neq m \in \{1, \ldots n\}$.
In the latter case, i.e., $q_{ad'alj} \equiv q_{a^2 lj}$ contributes to some of the a-transitions, in a^n, say some u a-transitions such that $u > 0$, then $q_{lj} \stackrel{a^2}{\rightarrow} q_{a^2 lj} \stackrel{a^u}{\rightarrow} q'$ for some q' and hence $q \stackrel{a^{n+u}}{\rightarrow} q''$ for some q''. But p can initially make at most n consecutive a-transitions, hence a contradiction follows. ⊠

As a corollary to the above theorem, one can conclude that TCCS affords no expansion theorem, i.e., parallel composition cannot be resolved in TCCS.

Corollary 9. *Two-sorted dense-time TCCS has no expansion theorem.*

4.2 Axiomatizability

Our next milestone in this section is to prove a theorem witnessing that TCCS, does not have a finite basis modulo timed bisimilarity. The problem underscored by the proof of this result is the inability of any finite and sound axiom system E to "expand" the initial behavior of terms of the form $p \,||\, q$ when either p or q have sufficiently many summands (namely, larger than the size of terms in the equations in E). All of the impossibility results presented henceforth also hold for conditional equations of the form $P \Rightarrow t = u$ where P is an arbitrary predicate over the time domain.

Theorem 10. *Timed bisimilarity over two-sorted dense-time TCCS has no finite basis.*

The above result dates back to [11] (for the case of CCS without time) and our proof follows the same structure as that of [11]; namely, we prove that for each finite and sound set of axioms E for TCCS modulo bisimilarity and for sufficiently large n, with respect to the size of the largest term appearing in E, the following sound equation is not provable

$$E \vdash a \,||\, \Phi_n = a.\Phi_n + \sum_{i \in \{1,\ldots,n\}} a.(a \,||\, \phi_i) \ ,$$

where $\Phi_n = \sum_{i \in \{1,\ldots,n\}} a.\phi_i$ and $\phi_n = \sum_{i \in \{1,\ldots,n\}} a^i$.

As we show in the next section, unlike in the setting of Milner's CCS, even adding two variations on the left-merge operator does not improve the situation with respect to axiomatizability.

5 Two-Sorted TCCS with Left-Merge

Classical Left-Merge. Bergstra and Klop suggested to add an auxiliary left-merge operator, denoted by $\lfloor\!\lfloor$, which would allow for a finite axiomatization of parallel composition in CCS. The semantics of the left-merge operator is captured by the following deduction rule.

$$\frac{x_0 \xrightarrow{a} y_0}{x_0 \lfloor\!\lfloor x_1 \xrightarrow{a} y_0 \,||\, x_1}$$

However, adding the left-merge operator with the above semantics does not result in a finitely axiomatizable theory.

Theorem 11. *Timed bisimilarity over two-sorted dense-time TCCS extended with the untimed left-merge operator has no finite basis.*

Timed Left-Merge. Following the tradition of Bergstra and Klop, the left-merge operator was given a timed semantics as follows [4].

$$\frac{x_0 \xrightarrow{a} y_0}{x_0 \lfloor\!\lfloor x_1 \xrightarrow{a} y_0 \,||\, x_1} \qquad \frac{x_0 \xrightarrow{\epsilon(d)} y_0 \quad x_1 \xrightarrow{\epsilon(d)} y_1}{x_0 \lfloor\!\lfloor x_1 \xrightarrow{\epsilon(d)} y_0 \lfloor\!\lfloor y_1}$$

This operator enjoys most of the axioms for the classic left-merge operator that lead to a finite axiomatization of bisimilarity [3]. The following lemma lists the most important properties that this timed left-merge operator possesses. Note that Lemma 1 also remains valid over TCCS extended with the above left-merge operator.

Lemma 12. *For the left-merge operator with the semantics given above, the following axioms are sound:*

$$\begin{array}{ll} x \,||\, y = (x \lfloor\!\lfloor y) + (y \lfloor\!\lfloor x) & \qquad 0 \lfloor\!\lfloor x = 0 \\ (x + y) \lfloor\!\lfloor z = (x \lfloor\!\lfloor z) + (y \lfloor\!\lfloor z) & \qquad x \lfloor\!\lfloor 0 = x. \end{array}$$

Thanks to axioms and , one can show that terms bisimilar to 0 can be removed as arguments of the left-merge. Henceforth, when we write p does not contain 0-factors, we mean that it does not contain a parallel composition *or a left-merge* with an argument bisimilar to 0.

However, the new left-merge operator does not help in giving TCCS a finite basis either, as we prove in the remainder of this section. The reason is that the axiom $(a.x) \lfloor\!\lfloor y = a.(x \,||\, y)$, which is a sound axiom in the untimed setting, is in general unsound over TCCS. For example, consider the process $a \lfloor\!\lfloor \epsilon(d).a$; after

making a time delay of length d, it results in $a \lfloor\!\lfloor a$, which is capable of performing two consecutive a-transitions. However, $a.(0 \,||\, \epsilon(d).a)$ after a time delay of length d remains the same process and can only perform one a-transition since the second a-transition still has to wait for some time, i.e., d, before becoming enabled.

However, axiom is sound for the class of TCCS processes that are initially time insensitive; see Definition 3. Indeed if q is a process such that $q \stackrel{\epsilon(d)}{\to} q_d$ implies $q \leftrightarrow q_d$ for each delay d, then it holds that $(a.p) \lfloor\!\lfloor q \leftrightarrow a.(p \,||\, q)$. For instance, $a \lfloor\!\lfloor \Phi_n \leftrightarrow a.\Phi_n$ for each $n \geq 0$, where the process Φ_n is defined as in Section 4.2. Unfortunately, the class of initially time insensitive processes cannot be characterized by a finite (head-)normal form and this constitutes the key idea in our non-finite axiomatizability proof, given below.

Theorem 13. *Two-sorted TCCS extended with the timed left-merge operator affords no finite axiomatization modulo timed bisimilarity.*

Proof. Towards a contradiction, we assume that TCCS with left-merge does have a finite axiomatization E. We prove the theorem by showing that the following lemma holds. (In the remainder of this proof, we assume that all terms appearing in equations do not contain parallel composition since, by axiom , parallel composition is a derived operator.)

Lemma 14. *Consider the equality $a \lfloor\!\lfloor \Phi_n = a.\Phi_n$. Let n_0 be the size of the biggest term t or u, appearing in equations $(t = u) \in E$. The above equation is not derivable from E for $n > max(n_0, 2)$.*

Once the above lemma is proven the theorem follows since the above equality is sound yet not derivable from E for $n > \max(n_0, 2)$. Lemma 14 is a consequence of the following result that establishes a property of equations that are derivable from E but that is not afforded by the equation $a \lfloor\!\lfloor \Phi_n = a.\Phi_n$ for suitably large values of n.

Lemma 15. *If $E \vdash p = q$,*

1. *p and q do not contain 0 summands or factors,*
2. *$p \leftrightarrow a \lfloor\!\lfloor \Phi_n$ for $n > max(n_0, 2)$, and*
3. *p has a summand of the form $p_0 \lfloor\!\lfloor p_1$ where $p_0 \leftrightarrow a$ and $p_1 \leftrightarrow \Phi_n$,*

then q has a summand of the form $q_0 \lfloor\!\lfloor q_1$ where $q_0 \leftrightarrow a$ and $q_1 \leftrightarrow \Phi_n$.

If we prove the above lemma then it follows that $a \lfloor\!\lfloor \Phi_n = a.\Phi_n$ for $n > \max(2, n_0)$ is not provable from E because the left-hand side satisfies the requirements of the statement but the right-hand side does not contain any summand of the form $q_0 \lfloor\!\lfloor q_1$.

In the proof of Lemma 15, we shall have some use for the following definition and the subsequent lemma [8,11].

Definition 16. *Process p is irreducible when for all p_0 and p_1, if $p \leftrightarrow p_0 \,||\, p_1$ then $p_0 \leftrightarrow 0$ or $p_1 \leftrightarrow 0$. We say that p is prime when it is irreducible and is not bisimilar to 0.*

Lemma 17. *The following processes are prime:*

1. ϕ_i, *for an arbitrary* $i \geq 1$;
2. Φ_i, *for all* $i > 1$;
3. $a.\Phi_i$, *for all* $i > 1$.

The proof of the above lemma is standard and is omitted for brevity. Note that neither item 2 nor item 3 in the above lemma hold for $i = 1$ since $\Phi_1 \equiv a.\phi_1 \equiv a.a \underline{\leftrightarrow} a \,||\, a$.

To prove Lemma 15, we use an induction on the derivation structure for $p = q$ and distinguish the following cases based on the last deduction rule applied in the derivation. (Since p and q have neither 0 summands nor factors, reasoning as in [9], we may assume that none of the terms mentioned in the proof of $p = q$ has 0 summands or factors.) The statement is trivial if $E \vdash p = q$ is due to . If $E \vdash p = q$ is due to , then there exists a term r such that $E \vdash p = r$ and $E \vdash r = q$ and the lemma follows by applying the induction hypothesis first on $E \vdash p = r$ and then on $E \vdash r = q$. If the last applied deduction rule is , then we distinguish the following cases based on the head operator of p and q.

1. $p \equiv a.p'$ and $q \equiv a.q'$; this case is vacuous since p should contain at least one summand which is of the form $p_1 \lfloor\lfloor\, p_2$;
2. $p \equiv \epsilon(d).p'$ and $q \equiv \epsilon(d).q'$; impossible, see above.
3. $p \equiv p_0 + p_1$, $q \equiv q_0 + q_1$, $E \vdash p_0 = q_0$ and $E \vdash p_1 = q_1$; without loss of generality, we assume that p_0 contains a summand of the form $p'_0 \lfloor\lfloor\, p'_1$ where $p'_0 \underline{\leftrightarrow} a$ and $p'_1 \underline{\leftrightarrow} \Phi_n$. It is not hard to see that $p_0 \underline{\leftrightarrow} a \lfloor\lfloor\, \Phi_n$ because $p \underline{\leftrightarrow} a \lfloor\lfloor\, \Phi_n$. It follows from the induction hypothesis that q_0 contains a summand that is of the form $q'_0 \lfloor\lfloor\, q'_1$ such that $q'_0 \underline{\leftrightarrow} a$, $q'_1 \underline{\leftrightarrow} \Phi_n$ and hence, so does q.
4. $p \equiv p_0 \lfloor\lfloor\, p_1$, $q \equiv q_0 \lfloor\lfloor\, q_1$, $E \vdash p_0 = q_0$ and $E \vdash p_1 = q_1$; it follows from the hypothesis of the lemma that $p_0 \underline{\leftrightarrow} a$ and $p_1 \underline{\leftrightarrow} \Phi_n$. By the soundness of E, we have that $p_0 \underline{\leftrightarrow} q_0 \underline{\leftrightarrow} a$ and $p_1 \underline{\leftrightarrow} q_1 \underline{\leftrightarrow} \Phi_n$ and thus the lemma follows.

It remains to consider the case where the last deduction rule applied is a closed instantiation of an axiom $(t = u) \in E$. In this case, there exists a substitution σ such that $\sigma(t) = p$ and $\sigma(u) = q$. Assume that $t \equiv \sum_{i \in I} t_i$ and $u \equiv \sum_{j \in J} u_j$ such that the t_i's and u_j's are not bisimilar to 0 and do not have + as their head operator. Let t_i be a summand of t such that $\sigma(t_i)$ has a summand of the form $p_0 \lfloor\lfloor\, p_1$ and $p_0 \underline{\leftrightarrow} a$ and $p_1 \underline{\leftrightarrow} \Phi_n$. We analyze the following cases based on the structure of t_i.

- $t_i \equiv x$ It is not difficult to prove that, since $t = u$ is sound modulo bisimilarity, there exists $j \in J$ such that $u_j \equiv x$. Then the lemma follows since $\sigma(x)$, and hence $\sigma(u)$, contains a summand of the form $p_0 \lfloor\lfloor\, p_1$ and $p_0 \underline{\leftrightarrow} a$ and $p_1 \underline{\leftrightarrow} \Phi_n$.
- $t_i \equiv a.t'_i$ Impossible since $\sigma(t_i)$ must have a summand of the form $p_0 \lfloor\lfloor\, p_1$.
- $t_i \equiv \epsilon(d).t'_i$ Impossible since $\sigma(t_i)$ must have a summand of the form $p_0 \lfloor\lfloor\, p_1$.
- $t_i \equiv t'_i \lfloor\lfloor\, t''_i$ Then, it is not hard to see that $\sigma(t'_i) \underline{\leftrightarrow} a$ and $\sigma(t''_i) \underline{\leftrightarrow} \Phi_n$. Write $t''_i = \sum_{k \in K} v_k$ where no v_k is bisimilar to 0 or has + as head operator. Since $\Phi_n \xrightarrow{a} \phi_i$, for each $0 < i \leq n$, the term $\sigma(t''_i)$ should mimic these

transitions, and because $2|K| < n$, there exists a $k \in K$ such that $\sigma(v_k) \xrightarrow{a} p'_i \leftrightarrow \phi_i$ for at least three different i's. By a case distinction on the structure of v_k, we argue that v_k can only be a variable:

(a) $v_k \equiv a.v'_k$: This leads to a contradiction. Indeed, then $\sigma(v'_k) \leftrightarrow \phi_i \leftrightarrow \phi_j$ for different i and j.

(b) $v_k \equiv \epsilon(d).v'_k$: Then $\sigma(v_k)$ cannot make an a-transition, which is a contradiction.

(c) $v_k \equiv v'_k \lfloor\lfloor v''_k$: Recall that $\sigma(v_k)$ can make an a-transition to ϕ_i and ϕ_j for some $i \neq j$. Hence $\sigma(v_k) \equiv \sigma(v'_k) \lfloor\lfloor \sigma(v''_k) \xrightarrow{a} p_i \,||\, \sigma(v''_k) \leftrightarrow \phi_i$ for some p_i. Since ϕ_i is prime (Lemma 17) $p_i \leftrightarrow 0$ and $\sigma(v''_k) \leftrightarrow \phi_i$. Similarly, $\sigma(v_k) \equiv \sigma(v'_k) \lfloor\lfloor \sigma(v''_k) \xrightarrow{a} p_j \,||\, \sigma(v''_k) \leftrightarrow \phi_j$ for some p_j and thus, $\sigma(v''_k) \leftrightarrow \phi_j$. Concluding, $\phi_i \leftrightarrow \sigma(v''_k) \leftrightarrow \phi_j$ for $i \neq j$, which is a contradiction.

Therefore $v_k \equiv x$ for some variable x and $\sigma(x)$ can make a-transitions to ϕ_i, ϕ_j and ϕ_k for different i, j and k. Then, x is not a summand of t, for this would contradict our assumption that $p \leftrightarrow a \lfloor\lfloor \Phi_n$. Indeed, the action depth of $\sigma(x)$ after an a-transition is at most n (the action depth of ϕ_n) while the action depth of $a \lfloor\lfloor \Phi_n$ after an a-transition is $n+1$ (the action depth of Φ_n). Furthermore, $x \notin vars(t'_i)$ or otherwise it would not hold that $\sigma(t'_i) \leftrightarrow a$ since $\sigma(t'_i)$ would have an action depth larger than 1. Also, x can only appear in the summands of t''_i that are of the form x or $\epsilon(d).x$. Indeed, if x occurred in summands that have any form other than x or $\epsilon(d).x$, then $\sigma(t''_i) \leftrightarrow \Phi_n$ would not be sound since $\sigma(t''_i)$ could then make two or more a-transitions (possibly interleaved with time delays) resulting in ϕ_i for some $i > 1$, which cannot be mimicked by Φ_n. Hence, we conclude that $t''_i = x + t'' + \sum_{i' \in I'} \epsilon(d_{i'}).x$ for some term t'' such that $x \notin vars(t'')$.

Consider the new substitution σ' defined to map x to $\epsilon(d).a.\Phi_n$, where d is *smaller than* each delay occurring in p or q, and to agree with σ on all other variables. (Such d exists since D has 0 as a cluster point.)

We have that $\sigma'(t_i) \xrightarrow{\epsilon(d)} p'_d \equiv \sigma(t'_i)_d \lfloor\lfloor (a.\Phi_n + \sigma(t'')_d + \sum_{i' \in I'} \epsilon(d_{i'} - d).\sigma'(x))$ where $\sigma(t'_i) \xrightarrow{\epsilon(d)} \sigma(t'_i)_d$ and $\sigma(t'') \xrightarrow{\epsilon(d)} \sigma(t'')_d$. Furthermore, as $\sigma(t'_i) \leftrightarrow a$ and axiom P is sound, $p'_d \xrightarrow{a} p'_{da} \leftrightarrow a.\Phi_n + \sigma(t'')_d$. Observe that the action depth of $\sigma(t'')_d$ can at most be $n+1$. It follows from $(t = u) \in E$ and the soundness of E that $\sigma'(t) \leftrightarrow \sigma'(u)$ and hence $\sigma'(u) \xrightarrow{\epsilon(d)} \sigma'(u)_d \xrightarrow{a} q'_{da} \leftrightarrow p'_{da}$ for some q'_{da}. Thus, there exists a summand u_j of u (for some $j \in J$) such that $\sigma'(u_j) \xrightarrow{\epsilon(d)} q'_d \xrightarrow{a} q'_{da}$. It holds that $x \in vars(u'_j)$ since otherwise, $\sigma(u_j) \equiv \sigma'(u_j)$ and q'_{da} would have action depth of at most $n+1$ (since $\sigma(u) \leftrightarrow a \lfloor\lfloor \Phi_n$). We distinguish the following cases based on the structure of u_j.

$u_j \equiv x$ Impossible since then $q'_{da} \equiv \Phi_n$ which is not bisimilar to $a.\Phi_n + \sigma(t'')_d \leftrightarrow p'_{da}$.

$u_j \equiv \epsilon(e).u'_j$ Impossible since d is smaller than each delay in p and q, which means that $d < e$ and thus, $\sigma(u_j)$ cannot perform an action after a time delay of length d.

$u_j \equiv a.u'_j$ We argue that this case leads to a contradiction. To this end, observe that, first of all, variable x can appear only in summands of u'_j which are of the form x or $\epsilon(d').x$. Otherwise, if u'_j has an action prefixing or left-merge operator with an argument containing x among its variables, the action depth of $\sigma'(u'_j)$ would be at least $n+3$, which is larger than the action depth of $p'_{da} \underline{\leftrightarrow}\, a.\Phi_n + \sigma(t'')_d$. Hence, $u'_j \underline{\leftrightarrow}\, x + u'' + \sum_{j' \in J'} \epsilon(d_{j'}).x$ for some term u'' such that $x \notin vars(u'')$. Thus, $q'_d \underline{\leftrightarrow}\, a.\ (\epsilon(d).a.\Phi_n + \sigma'(u'') + \sum_{j' \in J'} \epsilon((d_{j'} + d)).\sigma'(x))$ and $q'_{ad} \equiv \epsilon(d).a.\Phi_n + \sigma'(u'') + \sum_{j' \in J'} \epsilon(d_{j'} + d).\sigma'(x)$. It should hold that $q'_{ad} \underline{\leftrightarrow}\, a.\Phi_n + \sigma(t'')_d$; but $a.\Phi_n + \sigma(t'')_d \stackrel{a}{\rightarrow} \Phi_n$ and a matching a-transition of q'_{ad} can only be due to $\sigma'(u'')$, which does not contain x and thus is the same as $\sigma(u'')$. It holds that $adepth(\sigma(u'')) < adepth(\sigma(u_j)) \leq adepth(\sigma(u)) \leq n + 2 = adepth(a.\Phi_n + \sigma(t'')_d)$. Therefore, any a-derivative of q'_{da} will have action depth of at most n. Hence, it cannot hold that $q'_{da} \underline{\leftrightarrow}\, a.\Phi_n + \sigma(t'')_d$, contradicting our assumption.

$u_j \equiv u'_j \lfloor\!\lfloor\, u''_j$ By one of our assumptions $\sigma(u'_j)$ and $\sigma(u''_j)$ are not bisimilar to 0. Therefore, $\sigma'(u'_j)$ and $\sigma'(u''_j)$ are not bisimilar to 0, either. By our assumption, $\sigma'(u'_j) \lfloor\!\lfloor\, \sigma'(u''_j) \stackrel{\epsilon(d)}{\rightarrow} \stackrel{a}{\rightarrow} \sigma'(u'_j)_{da} \,||\, \sigma'(u''_j)_d \,\underline{\leftrightarrow}\, p'_{da}$. Recall that $p'_{da} \underline{\leftrightarrow} a.\Phi_n + \sigma(t'')_d$ where $\sigma(t'') \stackrel{d}{\rightarrow} \sigma(t'')_d$ and $\sigma(t'') + \Phi_n \underline{\leftrightarrow} \Phi_n$. It follows from the latter bisimilarity that $\sigma(t'')_d + \Phi_n \,\underline{\leftrightarrow}\, \Phi_n$. We claim that $a.\Phi_n + \sigma(t'')_d$ is prime and hence, $\sigma'(u'_j)_{da} \,\underline{\leftrightarrow}\, 0$ and $\sigma'(u''_j) \,\underline{\leftrightarrow}\, p'_{da}$.

To prove the above claim assume towards a contradiction that $r \,||\, s \,\underline{\leftrightarrow}\, a.\Phi_n + \sigma(t'')_d$ for r and s not bisimilar to 0. We distinguish the following cases based on the behavior of $\sigma(t'')_d$.

i. Assume that $\sigma(t'')_d \,\underline{\leftrightarrow}\, 0$. It follows that $r \,||\, s \,\underline{\leftrightarrow}\, a.\Phi_n$. However, this is impossible since $a.\Phi_n$ is prime (Lemma 17, item 3).

ii. Assume that $\sigma(t'')_d \stackrel{\epsilon(e)}{\rightarrow} \sigma(t'')_{d+e} \stackrel{a}{\rightarrow} \sigma(t'')_{(d+e)a} \,\underline{\leftrightarrow}\, \phi_i$ for some $i \leq n$. Then, without loss of generality, $r \,||\, s \stackrel{\epsilon(e)}{\rightarrow} r_e \,||\, s_e \stackrel{a}{\rightarrow} r' \,||\, s_e \,\underline{\leftrightarrow}\, \phi_i$ for some r' such that $r \stackrel{\epsilon(e)}{\rightarrow} r_e \stackrel{a}{\rightarrow} r'$. It follows from primality of ϕ_i that $r' \,\underline{\leftrightarrow}\, 0$ and $s_e \,\underline{\leftrightarrow}\, \phi_i$. It also holds that $a.\Phi_n + \sigma(t'')_d \stackrel{a}{\rightarrow} \Phi_n$. Thus, $r \,||\, s$ should be able to mimic this transition; the transition cannot be due to r because then $s \,\underline{\leftrightarrow}\, \Phi_n$ (since Φ_n is also prime), which contradicts $s_e \,\underline{\leftrightarrow}\, \phi_i$. Hence, $r \,||\, s \stackrel{a}{\rightarrow} r \,||\, s' \,\underline{\leftrightarrow}\, \Phi_n$. It follows from primality of Φ_n that $r \,\underline{\leftrightarrow}\, \Phi_n$ and $r_e \,\underline{\leftrightarrow}\, \Phi_n$. Thus, using congruence of $\underline{\leftrightarrow}$ with respect to $||$, we have that $r_e \,||\, s_e \,\underline{\leftrightarrow}\, \Phi_n \,||\, \phi_i$. Since the action depth of $a.\Phi_n + \sigma(t'')_{d+e}$ is $n+2$, we infer that $\phi_i \,\underline{\leftrightarrow}\, a$ and $i = 1$. But even then, $r_e \,||\, s_e \stackrel{a}{\rightarrow} v \,\underline{\leftrightarrow}\, \phi_n \,||\, a$, which cannot be mimicked by

$a.\Phi_n + \sigma(t'')_{d+e}$ (since $\sigma(t'')_{d+e}$ does not have the sufficient action depth and the only a-transition afforded by $a.\Phi_n$ results in Φ_n).

Thus, we conclude that $a.\Phi_n + \sigma(t'')_d$ is prime and hence, $\sigma'(u'_j)_{da} \underline{\leftrightarrow} 0$ and $\sigma'(u''_j) \underline{\leftrightarrow} a.\Phi_n + \sigma(t'')_d$. We claim that since d is smaller than all delays mentioned in p and q and $\sigma'(u'_j) \stackrel{\epsilon(d)}{\to} \stackrel{a}{\to} \sigma'(u'_j)_{da} \underline{\leftrightarrow} 0$, then $\sigma(u'_j) \stackrel{a}{\to} \sigma(u'_j)_a$ for some $\sigma(u'_j) \underline{\leftrightarrow} 0$. From this claim (whose proof is given next), it follows that $\sigma(u_j) \equiv \sigma(u'_j) \lfloor\lfloor \sigma(u''_j) \stackrel{a}{\to} \sigma(u'_j)_a \,||\, \sigma(u''_j) \underline{\leftrightarrow} \sigma(u''_j)$. On the other hand, $q \equiv \sigma(u) \underline{\leftrightarrow} a \lfloor\lfloor \Phi_n$ and thus, $\sigma(u''_j) \underline{\leftrightarrow} \Phi_n$. Hence, the action depth of $\sigma(u'_j)$ is 1 and therefore $\sigma(u'_j) \underline{\leftrightarrow} a$. To summarize, we have proved then that $\sigma(u_j) \equiv \sigma(u'_j) \lfloor\lfloor \sigma(u''_j)$, $\sigma(u'_j) \underline{\leftrightarrow} a$ and $\sigma(u''_j) \underline{\leftrightarrow} \Phi_n$, which was to be shown. Thus, it only remains to prove the following lemma.

Lemma 18. *Assume that d is smaller than each delay in $\sigma(u)$ and let r be a process that is not bisimilar to 0. Define σ' to map x to $\epsilon(d).a.r$ and all other variables y to $\sigma(y)$. Assume that $\sigma'(u) \stackrel{\epsilon(d)}{\to} \sigma'(u)_d \stackrel{a}{\to} \sigma'(u)_{da} \underline{\leftrightarrow} 0$. Then, $\sigma(u) \stackrel{a}{\to} \sigma(u)_a$ for some $\sigma(u)_a \underline{\leftrightarrow} 0$.*

Proof. By an induction on the structure of u. For brevity, we only give the proof for the case $u \equiv y$; the proofs for other cases are similar.

Assume that $u \equiv y$. First of all observe that y cannot be the same as x. Indeed $\sigma(x) \equiv \epsilon(d).a.r \stackrel{\epsilon(d)}{\to} a.r \stackrel{a}{\to} r$ and it does not hold that $r \underline{\leftrightarrow} 0$ by one of the provisos in the lemma. Thus, $\sigma'(y) \equiv \sigma(y) \stackrel{\epsilon(d)}{\to} \stackrel{a}{\to} q'$ and $q' \underline{\leftrightarrow} 0$. We proceed with an induction on the structure of $\sigma'(y) \equiv \sigma(y)$.

$\sigma(y) \equiv a.q'$ Then the lemma follows since $\sigma(y) \stackrel{a}{\to} q'$.

$\sigma(y) \equiv \epsilon(e).q'$ Impossible since then $\sigma(y)$ would not afford an a-transition after a time delay of length d because $d < e$.

$\sigma(y) \equiv q_0 + q_1$ Assume without loss of generality that $q_0 \stackrel{\epsilon(d)}{\to} \stackrel{a}{\to} q'$. Time delay d is smaller than the delays mentioned in $\sigma(y)$ and, hence, in q_0. It follows from the induction hypothesis that $q_0 \stackrel{a}{\to} q''$ for some $q'' \underline{\leftrightarrow} 0$ and therefore $\sigma(y) \equiv q_0 + q_1 \stackrel{a}{\to} q'' \underline{\leftrightarrow} 0$.

$\sigma(y) \equiv q_0 \lfloor\lfloor q_1$ Then, $q_0 \stackrel{\epsilon(d)}{\to} \stackrel{a}{\to} q'_0 \underline{\leftrightarrow} 0$ and $q' \equiv q'_0 \,||\, q_{1d} \underline{\leftrightarrow} 0$ where $q_1 \stackrel{\epsilon(d)}{\to} q_{1d}$; hence, $q_{1d} \underline{\leftrightarrow} q_1 \underline{\leftrightarrow} 0$. It follows from the induction hypothesis that $q_0 \stackrel{a}{\to} q''_0$ for some $q''_0 \underline{\leftrightarrow} 0$ and thus, $\sigma(y) \equiv q_0 \lfloor\lfloor q_1 \stackrel{a}{\to} q''_0 \,||\, q_1 \underline{\leftrightarrow} 0$.

6 Conclusions

In this paper, we studied the equational theory of TCCS as proposed by Wang Yi in [12]. We improved upon the Gap Theorem of [5] and proved that, even in the presence of a single basic action, parallelism in TCCS cannot be resolved

in general. Furthermore we showed that TCCS, in its single- and two-sorted presentations, as well as its extensions with the untimed or the timed left-merge operator, does not afford a finite axiomatization.

It is an open question whether there exists a binary operator that, when added to TCCS, can give timed bisimilarity a finite basis. (A similar question is still open for untimed process algebras, i.e., whether there exists a single binary operator that can axiomatize communication and concurrency; the answer in both cases is expected to be negative.) Towards achieving this goal, in the extended version of this paper, we prove that adding two different variants of the timed left-merge operator $\lfloor\!\lfloor_0$ and $\lfloor\!\lfloor_1$ with the following semantics does not lead to a finite axiomatization for bisimilarity. (The leftmost rule below applies to both $\lfloor\!\lfloor_0$ and $\lfloor\!\lfloor_1$.)

$$\frac{x_0 \xrightarrow{a} y_0}{x_0 \lfloor\!\lfloor_{0,1}\, x_1 \xrightarrow{a} y_0 \,||\, x_1} \qquad \frac{x_0 \xrightarrow{\epsilon(d)} y_0}{x_0 \lfloor\!\lfloor_0\, x_1 \xrightarrow{\epsilon(d)} y_0 \,||\, x_1} \qquad \frac{x_0 \xrightarrow{\epsilon(d)} y_0 \quad x_1 \xrightarrow{\epsilon(d)} y_1}{x_0 \lfloor\!\lfloor_1\, x_1 \xrightarrow{\epsilon(d)} y_0 \,||\, y_1}$$

In the case of two-sorted TCCS, our proofs make use of the fact that the time domain has 0 as a cluster point. It remains open whether discrete-time TCCS (or its extension with (timed) left-merge) is finitely axiomatizable modulo bisimilarity.

References

1. Aceto, L., Jeffrey, A.: A complete axiomatization of timed bisimulation for a class of timed regular behaviours. TCS 152(2), 251–268 (1995)
2. Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: Paterson, M.S. (ed.) Automata, Languages and Programming. LNCS, vol. 443, pp. 322–335. Springer, Heidelberg (1990)
3. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. I&C 60(1-3), 109–137 (1984)
4. Baeten, J.C.M., Middelburg, C.A.: Process algebra with timing. Springer, Heidelberg (2002)
5. Godskesen, J.C., Larsen, K.G.: Real-time calculi and expansion theorems. In: Shyamasundar, R.K. (ed.) Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 652, pp. 302–315. Springer, Heidelberg (1992)
6. Jeffrey, A., Schneider, S., Vaandrager, F.W.: A comparison of additivity axioms in timed transition systems. Report CS-R9366, CWI, Amsterdam (1993)
7. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)
8. Milner, R., Moller, F.: Unique decomposition of processes. TCS 107(2), 357–363 (1993)
9. Moller, F.: Axioms for Concurrency. Ph.D. Thesis, University of Edinburgh (1989)
10. Moller, F., Tofts, C.M.N.: A temporal calculus of communicating systems. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 401–415. Springer, Heidelberg (1990)

11. Moller, F.: The importance of the left merge operator in process algebras. In: Paterson, M.S. (ed.) Automata, Languages and Programming. LNCS, vol. 443, pp. 752–764. Springer, Heidelberg (1990)
12. Yi, W.: Real-time behaviour of asynchronous agents. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 502–520. Springer, Heidelberg (1990)
13. Yi, W.: CCS + time = an interleaving model for real time systems. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) Automata, Languages and Programming. LNCS, vol. 510, pp. 217–228. Springer, Heidelberg (1991)
14. Yi, W.: A calculus of real time systems. PhD thesis, Chalmers University of Technology (1991)

Conceptual Data Modeling with Constraints in Maude

Scott Alexander

scottxyz@usa.com

Abstract. *Conceptual data modeling* (CDM) for relational databases can declare *constraints* on both computed and stored relations, and abstracts from entity-relationship (E/R) modeling by not distinguishing between *entities* and *attributes*. To provide a formal semantics, better interoperability, and arbitrary constraints, we can map CDM to a wide-spectrum algebraic specification language such as Maude. A case study is presented using a *functional* module to represent a conceptual data model and its constraints, and a *system* module to obtain a constraint-enforcing interpreter allowing concurrent edits to the database state.

1 Motivation

Relational databases allow *declarative constraints* (such as unique indexes and non-null fields) only for (stored) tables but not for (computed) views ("queries"), requiring instead *procedural* constructs such as triggers for constraints on queries. This has led to a proliferation of relational database modeling languages (many lacking a formal semantics), described by some as a "methodological jungle" [1]. Entity-relationship (E/R) modeling (*e.g.*, [2]) may allow declaring constraints on both tables and queries equally, but may still be too concrete and non-modular (in the "extensibility" sense [3]) for earlier, more "conceptual" modeling stages, as it forces premature distinctions between *entities* and *attributes*.

Conceptual data modeling (CDM) methods such as ORM, PSM, NIAM, and FCO-IM [4,5,6,7,8,9] map either an entity or an attribute to an *(object) type*, and capture the notion of "an attribute in an entity" as a *(fact) type* having a sequence of two *role arrows*, one to the ("attribute") object type and one to the ("entity") object type. CDM diagrams are more modularly extensible than E/R diagrams, because to extend an E/R diagram an attribute may have to be changed to an entity, a distinction lacking in CDM. (Figure 1 shows an example.) A "relational mapping" procedure (*e.g.*, `Rmap` [5,6]) maps a CDM model to an E/R model, essentially mapping CDM arrows sharing a common target (resp. source) type and belonging to several "mandatory" constraints (resp. to one "uniqueness" constraint) to *attributes* of a single *entity* — which is mapped to E/R from the arrows' shared target (resp. source) CDM type.

CDM diagrams cannot easily depict more-complex constraints, often lack a formal semantics, and must interoperate with other systems, so mapping CDM with constraints to a wide-spectrum multiparadigm executable algebraic specification language would be useful. Adapting techniques used for UML+OCL

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 96–110, 2007.
© Springer-Verlag Berlin Heidelberg 2007

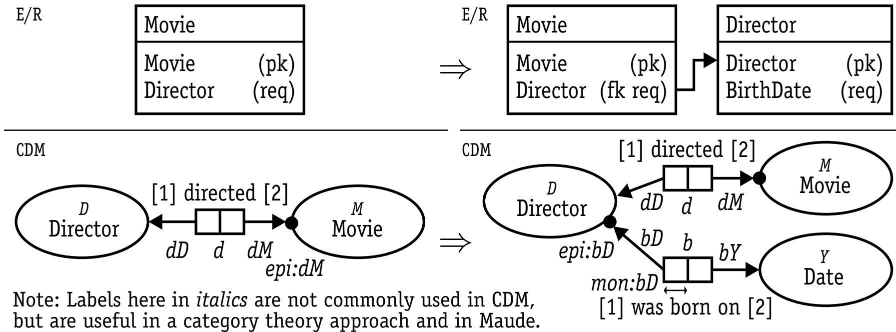


Fig. 1. Adding the Director's BirthDate in the *E/R diagrams* (upper row) requires changing `Director` from an entity to an attribute, but in the *CDM diagrams* (lower row) no existing diagram elements need to be changed

diagrams [10] and RM-ODP specifications [11,12], we can use Maude [13] to represent a conceptual data model or *type graph*, and populations or instances of its *type model* (*i.e.*, database *states*), using a multiset configuration defined in a *property specification* expressed as a *functional module* which provides operators to check constraints — and then use Maude's *rewriting semantics* [14] to obtain an interpreter defined in a *system specification* expressed as a *system module* which enforces the constraints while allowing concurrent edits to the database state. "Big-step" [3] rewrites will group several rewrite steps (possibly invalid individually) into a single (collectively valid) rewrite step (simulating `Rmap`).

2 Conceptual Data Modeling with Constraints

Any CDM type can also be a *subtype* of one or more other types, depicted in diagrams as a heavy arrow from (sub)type to (super)type. Multiple inheritance is allowed. *Subtype arrows* (labeled `(spec)` here) which *specialize* their target (super)type(s) are distinguished from subtype arrows (labeled `(gen)` here, and depicted using dotted heavy arrows here) which *generalize* their source (sub)type(s). CDM can also define a *multiset* (collection, bag) type using a *power* `(pow)` role and an associated *element* `(elt)` role.

A CDM *constraint* can be declared on one or more role or subtype arrows, or object or fact types. Figure 1 declares two *mandatory* (*total*) constraints: • *`epi:dM`* (resp. • *`epi:bD`*) requiring that arrow `dM` (resp. `bD`) must be epic, or surjective, *i.e.*, there must be *at least one* `dM` (resp. `bD`) arrow instance for every `M` (resp. `D`) object instance. It also declares a *uniqueness* constraint ↔ *`mon:bD`* requiring that arrow `bD` must be monic, or injective, *i.e.*, there may be *at most one* `bD` arrow instance for every `D` object instance. In general, a CDM mandatory (resp. uniqueness) constraint can apply to a set or *cotuple* (resp. a sequence or *tuple*) of arrows *having a common target (resp. source)* — or else *"extensible to" one* by traversing the type graph composing "forward" along pushouts

such as subtype inclusions (resp. "backward" along pullbacks such as joins), in which case the • (resp. a `U` or `P`) is circled and connected by dotted lines to the arrows defining the constraint. For an example, see the *external (primary) uniqueness* constraint `mon:o*i@oC=iC` in Fig. 2. A *disjoint* constraint on a set of subtype arrows (depicted by a ⊗ symbol connected with a dotted line to each arrow) requires that their pullback must be empty, *i.e.*, their source types must not share any instances. A *set-comparison* constraint between *compatible* role sequences, *i.e.*, where corresponding roles have the same target (depicted using either (i) a circled ⊆ or (ii) a ⊗ symbol on top of a a dotted-line arrow going from the first sequence to the second — or (iii) just a double-headed dotted-line arrow between the sequences), requires that the first role sequence population must be either (i) a subset of, (ii) mutually exclusive with, or (iii) equal to the second. A *ring* constraint on two role arrows having the same target requires that the role sequence population (viewed as a binary relation) must be *e.g.* antisymmetric, asymmetric, acyclic, irreflexive, intransitive, or symmetric, and is depicted using a small superscripted circle followed by the abbreviated property name, *e.g.*, °`acyc` or °`irr`. A *cardinality* constraint on a role (sequence), generalizing mandatory and uniqueness constraints and recorded using inequalities or a range `N..M`, requires that each target instance (product) must have from `N` to `M` corresponding arrow instances. A *value* constraint on an object (recorded using inequalities, sets, and ranges) limits the object's allowable instance values. Certain more-complex constraints, difficult to depict *graphically*, must be recorded *textually* ("off-diagram"), and some CDM tools may omit mapping them to E/R.

3 Rewriting Logic and Maude

Maude [13] is a high-level language and a high-performance interpreter and compiler in the OBJ [15] algebraic specification family that supports membership equational logic [16] and rewriting logic [17] specification and programming.

Membership equational logic (MEL) is a Horn logic whose atomic sentences are equalities $t = t'$, and membership assertions of the form $t : S$ stating that a term t has sort S. Such a logic extends order-sorted equational logic, and supports sorts, subsort relations, subsort polymorphic overloading of operators, and the definition of partial functions with equationally defined domains.

Rewriting logic (RWL) is a logic of change that can naturally deal with state and with highly nondeterministic concurrent computations. In rewriting logic, the state space of a distributed system is specified as an algebraic data type in terms of an equational specification (Σ, E), where Σ is a signature of sorts (types) and operations, and E is a set of (conditional) equational axioms. The dynamics of a system in rewriting logic is then specified by a set R of labeled rewrite rules of the form $t \rightarrow t'$, where t and t' are Σ-terms. These rules describe the local, concurrent transitions possible in the system; *i.e.*, when a part of the system state fits the pattern t then it can change to a new local state fitting pattern t'. Rules may be conditional, in which case the guards act as blocking pre-conditions, in the sense that a conditional rule can only be fired if the condition is satisfied.

4 Expressing a Conceptual Data Model in Maude

4.1 Fact and Object Types and Instances, and Constraint Satisfaction

CDM fact types can be modeled as *multirelations* [18] — or, more abstractly, as *spans* [19,20] with labeled legs, avoiding the need to order a fact type's roles. In a categorical approach [1], a conceptual data model is a *type graph* where a *node* is a fact type or an object type, an *edge* is a subtype inclusion or a role arrow, and the subtype subgraph is cycle-free and commutes. A *type model* (capturing the notion of a population or state) is a graph homomorphism from a type graph to an instance category with finite sums and products and disjoint sums.

Using the more-concrete multirelations approach [19,1], we can represent a type graph and its type model in Maude using a multiset configuration (of sort `PreCdm`) built from terms representing object and fact instances:

```
fmod PRECDM is
  sorts Obj Fact PreCdm .
  subsorts Obj Fact < PreCdm .
  op nil : -> PreCdm .
  op _;_ : PreCdm PreCdm -> PreCdm
    [assoc comm id: nil prec 121 format(d s ni d)] .
endfm
```

CDM object and fact types will declared to be subsorts of `Obj` and `Fact`. Functional module `CDM`, which imports `PRECDM`, provides an extensible mechanism for checking constraints, using MEL to define a "partial coercion" operator `{_}` returning a term of *sort* `Cdm` (resp. of *kind* `[Cdm]`) if its argument satisfies (resp. does not satisfy) arbitrary conditions which we can specify later:

```
fmod CDM is extending PRECDM .
  sorts Cdm MTruth .
  ops tt ff : -> MTruth .
  op {_} : [PreCdm] -> [Cdm] [format(d n+++i s--- d)] .
endfm
```

We can later define constraints using partial operators taking a term of kind `[PreCdm]` and expressing satisfaction by returning a term of kind `[MTruth]` or returning the term `tt` of sort `MTruth`. Value constraints can be defined using just constants or membership axioms. A term of sort `PreCdm` will be a *valid* instance or population of the conceptual data model iff it reduces to a term of sort `Cdm` using partial operator `{_}` (*i.e.*, iff all constraints are satisfied).

Using pattern-matching and term-rewriting, Maude functional modules can define and check *arbitrary* constraints (*i.e.*, those expressible in MEL). These clearly include all *graphical* (*i.e.*, finite-set-theoretic) as well as *textual* (*e.g.*, order-sorted algebraic) constraints expressible in CDM.

4.2 Case Study: Expressing a CDM Diagram with Constraints in Maude

CDM diagram `UNI` in Fig. 2 models a university, including various types of persons plus departments, courses, prerequisites, dates, quarters, and grades.

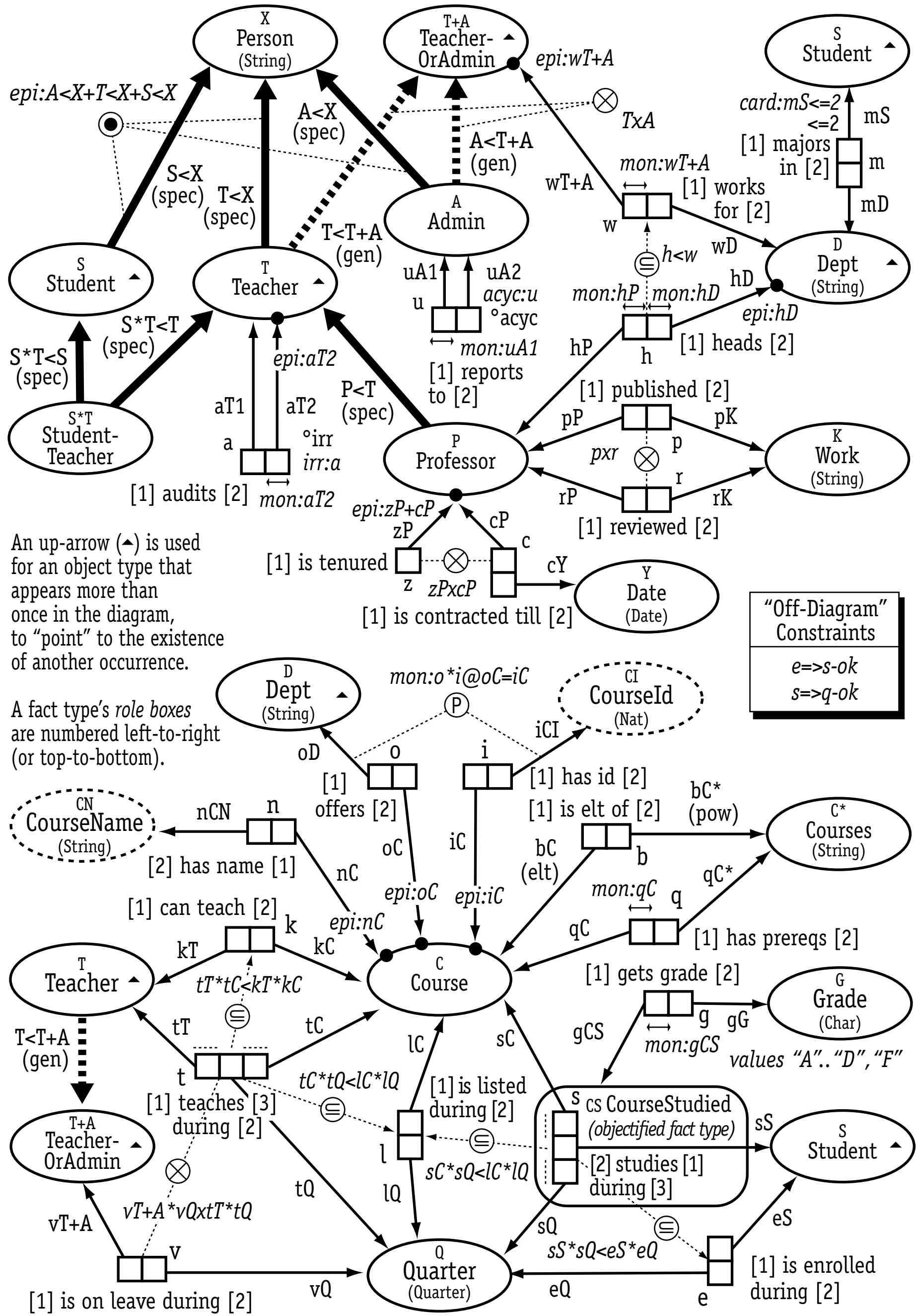


Fig. 2. Case study: conceptual data model UNI, modeling a university

Constraints (in `italics`) on role arrows require that: every `TeacherOrAdmin` must work for at least (*epi:wT+A*) and at most (*mon:wT+A*) one `Dept`, every `Dept` must be headed by exactly one (*epi:hD*, *mon:hD*) `Professor`, and a `Professor` may head at most one (*mon:hP*) `Dept` — in which case they must also work for that `Dept` (*h<w*). A `Student` may major in up to two `Depts` (*card:mS<=2*). A `Professor` must be either (*epi:zP+cP*) tenured, or contracted till a `Date`, but not both (*zPxcP*). Every `Teacher` must be audited by exactly one (*epi:aT2*, *mon:aT2*) other (*irr:a*) `Teacher`. An `Admin` may report to at most one (*mon:uA1*) `Admin`, and the reporting relation must be acyclic (*acyc:u*). The same `Professor` is not allowed to have both reviewed and published the same `Work` (*pxr*). Subtype arrows and constraints require that: every `Person` must be either a `Student`, a `Teacher` or an `Admin` (*epi:A<X+T<X+S<X*) and no `Person` may be both a `Teacher` and an `Admin` (*TxA*). Every `StudentTeacher` is both a `Student` and a `Teacher` (multiple inheritance), and every `TeacherOrAdmin` is either a `Teacher` or an `Admin` (generalization).

A `Teacher` may teach a `Course` during a `Quarter` only if: the `Course` is listed for that `Quarter` (*tC*tQ<lC*lQ*), they are not on leave during that `Quarter` (*vT+A*vQxtT*tQ*), and they "can teach" that `Course` (*tT*tC<kT*kC*). A `Student` may study a `Course` during a `Quarter` only if: the `Course` is listed for that `Quarter` (*sC*sQ<lC*lQ*), they are enrolled during that `Quarter` (*sS*sQ<eS*eQ*), and they have studied (and gotten `Grade "A".."D"` in) all the prerequisites for that `Course` (*s=>q-ok*, off-diagram). Every `Course` must have a `CourseName` (*epi:nC*). For every `Course` instance, the combination of the `Dept` (that offers it) and its `CourseId` must exist (*epi:oC*, *epi:iC*) and be unique (*mon:o*i@oC=iC*). A `Course` may have as prerequisites at most one (*mon:qC*) *set* of `Courses` (built using `(pow)` arrow `bC*` and `(elt)` arrow `bC`). Every `CourseStudied` may get at most one `Grade` (*mon:gCS*). Also, if a `Student` is enrolled during a `Quarter`, they must take one to four `Courses` during that `Quarter` (*e=>s-ok*, off-diagram).

In Maude functional module `fmod UNI` we map the CDM object types and fact types to sorts (all subsorts of `Obj` or `Fact`) and appropriate constructors; the subtype subgraph to additional subsorts; and the value constraint for `Grade` to a membership axiom.

```
fmod UNI is pr PRECDM . pr STRING .
 sorts Dept Person Admin Professor Student Teacher StudentTeacher
   TeacherOrAdmin Date Quarter Work Course Courses .
 subsorts Course Dept Person Quarter Courses Work Date < Obj .
 subsorts Admin Teacher Student < Person .
 subsorts Professor < Teacher .
 subsorts StudentTeacher < Student Teacher .
 subsorts Teacher Admin < TeacherOrAdmin .
 op person  : String -> Person [ctor] .
 op dept    : String -> Dept [ctor] .
 op work    : String -> Work [ctor] .
 op date    : NzNat NzNat NzNat ~> Date [ctor] .
 op course  : Dept Nat -> Course [ctor] .
 op quarter : NzNat NzNat ~> Quarter [ctor] .
 op courseSet : String -> Courses [ctor] .
```

```
  vars Y M D Q : NzNat .  var G : Char .
 cmb date( Y , M , D ) : Date if Y >= 1600 /\ M <= 12
  /\ M == 9 or M == 4 or M == 6 or M == 11 implies D <= 30
  /\ M =/= 9 and M =/= 4 and M =/= 6 and M =/= 11 and M =/= 2
       implies D <= 31
  /\ M == 2 and (Y rem 4 == 0) implies D <= 29
  /\ M == 2 and (Y rem 4 > 0 or Y rem 100 == 0 and Y rem 400 > 0)
        implies D <= 28 . --- 1900, 1800, 1700 are not leap years
 cmb quarter( Y , Q ) : Quarter if  Y >= 1600 /\ Q <= 4 .

  sorts    a b c e g h i k l m n o p q r s t u v w z .
  subsorts a b c e g h i k l m n o p q r s t u v w z < Fact .
  op _audits_ : Teacher Teacher -> a [ctor] .
  op _is`elt`of_ : Course Courses -> b [ctor] .
  op _is`contracted`till_ : Professor Date -> c [ctor] .
  op _is`enrolled`during_ : Student Quarter -> e [ctor] .
  op _gets`grade_ : s Char ~> g [ctor] . --- s is "objectified"
 cmb s1:s gets grade G : g if "A" <= G and G <= "D" or G == "F" .
  op _heads_ : Professor Dept -> h [ctor] .
  op _has`id_ : Course Nat -> i [ctor] .
  op _can`teach_ : Teacher Course -> k [ctor] .
  op _is`listed`during_ : Course Quarter -> l [ctor] .
  op _majors`in_ : Student Dept -> m [ctor] .
  op _has`name_ : Course String -> n [ctor] .
  op _offers_ : Dept Course -> o [ctor] .
  op _published_ : Professor Work -> p [ctor] .
  op _has`prereqs_ : Course Courses -> q [ctor] .
  op _reviewed_ : Professor Work -> r [ctor] .
  op _studies_during_ : Student Course Quarter -> s [ctor] .
  op _teaches_during_ : Teacher Course Quarter -> t [ctor] .
  op _reports`to_ : Admin Admin -> u [ctor] .
  op _is`on`leave`during_ : TeacherOrAdmin Quarter -> v [ctor] .
  op _works`for_ : TeacherOrAdmin Dept -> w [ctor] .
  op _is`tenured : Professor -> z [ctor] .
 endfm
```

Maude's sort hierarchy checking ensures that the subtype subgraph is cycle-free and commutes. Maude sorts mapped from CDM types at the source of a `(spec)` subtype arrow or at the target of a `(gen)` subtype arrow do not need constructors, as such types use the *reference scheme* [4,5] of the type at the other end of the arrow; instead, terms which are (proper) subsorts of `Person` arise via membership axioms, and a term of sort `TeacherOrAdmin` arises whenever a term of sort `Teacher` or `Admin` is created. *Objectified* (or "nested") [4,5] CDM fact type `CourseStudied` does not need its own sort or constructor in Maude; we simply enclose the *Student studies Course during Quarter* fact instance (of sort `s`) inside parentheses in any term constructed from `op _gets`grade_`.

Maude's mixfix syntax for object and fact instances echoes natural language, a central goal of CDM methods (with names like NIAM, "natural language information analysis method," or FCO-IM, "fully communication-oriented information modeling"), recognizing that communication with *domain experts* about the *universe of discourse* in a natural, precise language is essential for successful data modeling. "Syntactic sugar" enables alternate fact type *readings* [5], *e.g.*:

```
op _is‘offered‘by_ : Course Dept -> o . --- no [ctor] attrib
eq C:Course is offered by D:Dept = D:Dept offers C:Course .
```

In `fmod UNI.CDM` we declare all (explicit) constraints shown in Fig. 2 and define the equation(s) for a sample *mandatory*, *uniqueness*, and *subset* constraint, and for the three *mutual-exclusion* set-comparison constraints, the *uniqueness* constraint on a *join*, the *(maximum) cardinality* constraint, the *mandatory* constraint on *two* arrows, and the *irreflexive* constraint. We use a refutational style exploiting MEL, pattern-matching and Maude's `otherwise` (`owise`) attribute (which allows its equation to be applied only if all other equations for the same top term having more-specific left-hand sides fail to match) to define "negative predicates" which attempt to find a counterexample, *i.e.*, a *violation* of the constraint. Only if such a counterexample is *found* does the term reduce to a "true" value (`tt`) of *sort* `MTruth`. If *no* counterexample is found, a value (other than `tt`) of *kind* `[MTruth]` results, meaning the negative predicate is *false* (*i.e.*, the constraint is *satisfied*). Negative predicates are prefixed with a ∼ symbol as a reminder. Iff none of the negative predicates (all tested in membership axiom `cmb { M? } : Cdm`) equals `tt`, then the `PreCdm`(-term) is a valid `Cdm`(-term).

```
fmod UNI.CDM is protecting UNI . extending CDM .
 ops _~epi:wT+A _~mon:wT+A _~h<w         _~zPxcP       _~e=>s-ok
     _~epi:hD   _~mon:hD   _~tC*tQ<lC*lQ _~pxr         _~s=>q-ok
     _~epi:aT2  _~mon:aT2  _~tT*tC<kT*kC _~vT+A*vQxtT*tQ _~def:D
     _~epi:oC   _~mon:gCS  _~sC*sQ<lC*lQ _~mon:o*i@oC=iC --- ...
     _~epi:iC   _~mon:hP   _~sS*sQ<eS*eQ _~epi:zP+cP    _~irr:a
     _~epi:nC   _~mon:qC   _~mon:uA1     _~card:mS<=2   _~acyc:u
 :  [PreCdm] -> [MTruth] .

 var M? : PreCdm .    var X : Person .  var Q Q1 Q2 : Quarter .
 var N N1 N2 : Nat .  var S : Student . var C C1 C2 : Course .
 var D D1 D2 : Dept . var T : Teacher . var C* C*' : Courses .
 var Y : Date .       var A A1 A2 : Admin .
 var K : Work .       var P P1 P2 : Professor .
 var G : Char .       var T+A : TeacherOrAdmin .

 eq F:Fact ; F:Fact ; M? = F:Fact ; M? . --- remove dupe facts
 eq O:Obj  ; O:Obj  ; M? = O:Obj  ; M? . --- remove dupe objects

 eq (T+A ; T+A works for D ; M?) ~epi:wT+A = M? ~epi:wT+A .
 eq (T+A ;                  M?) ~epi:wT+A = tt [owise] .
 --- similar equation-pairs for all other mandatory constraints:
 --- ops _~epi:hD _~epi:aT2 _~epi:oC _~epi:iC _~epi:nC
 eq (T+A works for D1 ; T+A works for D2 ; M?) ~mon:wT+A = tt .
 --- similar equations for all other uniqueness constraints:
 --- ops _~mon:hD _~mon:aT2 _~mon:gCS _~mon:hP _~mon:qC _~mon:uA1
 eq (P heads D ; P works for D ; M?) ~h<w = M? ~h<w .
 eq (P heads D ;                 M?) ~h<w = tt [owise] .
 --- similar equation-pairs for all other subset constraints:
 --- ops _~tC*tQ<lC*lQ _~tT*tC<kT*kC _~sC*sQ<lC*lQ _~sS*sQ<eS*eQ

 eq (P ; P is tenured ; P is contracted till Y ; M?) ~zPxcP = tt .
 eq (P published K ; P reviewed K ; M?) ~pxr = tt .
```

```
  eq (T is on leave during Q ; T teaches C during Q ; M?)
      ~vT+A*vQxtT*tQ = tt .
  eq (S majors in D ; S majors in D1 ; S majors in D2 ;
      M? ; S ) ~card:mS<=2 == tt .
  eq (D1 offers C ; C has id N1 ;
      D2 offers C ; C has id N2 ; M?) ~mon:o*i@oC=iC = tt .
  eq (P ; P is tenured          ; M?) ~epi:zP+cP = M? ~epi:zP+cP .
  eq (P ; P is contracted till Y ; M?) ~epi:zP+cP = M? ~epi:zP+cP .
  eq (P ;                         M?) ~epi:zP+cP = tt [owise] .
  eq (T audits T ; M?) ~irr:a = tt .

 cmb { M? } : Cdm if M? ~def:D =/= tt  --- ...
  /\ M? ~epi:wT+A =/= tt  --- ...
  /\ M? ~mon:wT+A =/= tt  --- ...
  /\ M? ~h<w  --- ...
  /\ M? ~zPxcP =/= tt             /\ M? ~card:mS<=2 =/= tt
  /\ M? ~pxr =/= tt               /\ M? ~mon:o*i@oC=iC =/= tt
  /\ M? ~vT+A*vQxtT*tQ =/= tt     /\ M? ~epi:zP+cP  =/= tt
  /\ M? ~irr:a =/= tt             /\ M? ~e=>s-ok =/= tt
  /\ M? ~acyc:u =/= tt            /\ M? ~s=>q-ok =/= tt  .
 endfm
```

(Acyclic) ring constraint *acyc:u* uses auxiliary operators `_$occurs1:u_` and `_$occurs2:u_` to check whether an *Admin* instance occurs as the first or second argument in an *Admin reports to Admin* fact instance. The three equations for `_~acyc:u` use structural induction on facts of sort `u`, removing any which aren't "linked" at *both* ends into the binary relation (and so can't be part of a cycle):

```
  ops _$occurs1:u_ _$occurs2:u_ : [Admin] [PreCdm] -> [MTruth] .
  eq  A1 $occurs1:u ( A1 reports to A2 ; M? ) = tt .
  eq  A2 $occurs2:u ( A1 reports to A2 ; M? ) = tt .
 ceq  ( M? ; A1 reports to A2 ) ~acyc:u = M? ~acyc:u
  if  ( A2 $occurs1:u M? =/= tt or A1 $occurs2:u M? =/= tt )
  /\  A1 =/= A2 .
  eq  (M? ; U:u ) ~acyc:u = tt [owise] .
  eq  M? ~acyc:u = ff [owise] .
```

"Off-diagram" constraint *s=>q-ok* (requiring that a `Student` may study a `Course` during a `Quarter` only if they have gotten a non-failing `Grade` for each of that `Course`'s prerequisite `Courses`) can be efficiently refuted using just two equations with the `[owise]` attribute, using structural induction on `b`-terms:

```
 ceq ( M? ; C2 has prereqs C* ; C1 is elt of C* ;
      ( (S studies C1 during Q1) gets graded G ) ;
         S studies C2 during Q2 ) ~s=>q-ok
  =  ( M? ; C2 has prereqs C* ;
       ( (S studies C1 during Q1) gets graded G ) ;
          S studies C2 during Q2 ) ~s=>q-ok       if G =/= "F" .
  eq ( M? ; C2 has prereqs C* ; C1 is elt of C* ;
       S studies C2 during Q ) ~s=>q-ok = tt [owise] .
```

To refute "off-diagram" constraint *e=>s-ok* we first define an auxiliary predicate `>maxCourses` (resp. `<minCourses`) which returns `tt` iff a given `Student`

studies more (resp. fewer) than a given number of `Course`s during a given `Quarter`, and then use structural induction on `e`-terms:

```
ops >maxCourses <minCourses :
    [PreCdm] [Student] [Quarter] [Nat] -> [MTruth] .
eq >maxCourses( (M? ; S studies C during Q) , S , Q , (s N) )
=  >maxCourses(  M? , S , Q , N ) .
eq >maxCourses( (M? ; S studies C during Q) , S , Q , 0 ) = tt .
eq <minCourses( (M? ; S studies C during Q) , S , Q , (s N) )
=  <minCourses(  M? , S , Q , N ) .
eq <minCourses(  M? , S , Q , (s N) ) = tt [owise] .
ceq ( M? ; S is enrolled during Q ) ~e=>s-ok = tt
 if >maxCourses(  M? , S , Q , 4 ) == tt
 or <minCourses(  M? , S , Q , 1 ) == tt or M? ~e=>s-ok == tt .
```

For each sort (*e.g.*, `Dept`), we also need a negative predicate (*e.g.*, `_~def:D`) with two equations for each fact type constructor that uses it (*e.g.*, `_heads_`, `_works‘for_`, `_offers_`, `_majors‘in_`), reducing to `tt` iff the fact constructor is ever called on an undefined object instance:

```
eq  ( D ; P heads D ; M? ) ~def:D = ( D ; M? ) ~def:D .
eq  (     P heads D ; M? ) ~def:D = tt [owise] .
```

Most CDM diagrams assume a uniqueness constraint on *each* fact type spanning *all* its roles, handled by the first equation in `UNI.CDM`, removing dupes.

5 Getting a Constraint-Enforcing Interpreter "for free" Using Maude's Rewriting Semantics

Maude *functional* modules of the form `fmod` (Σ,E) `endfm` specify *equational* theories and provide a functional style of programming, using (conditional) *equations* oriented from left to right as simplification rules to reach a fully reduced or canonical form (assuming the equations are terminating and confluent). Maude *system* modules on the other hand are of the form `mod` (Σ,E,R) `endm` where (Σ,E) is a membership equational theory, and R is a set of labeled (conditional) rewrite *rules* of the form $r : t \rightarrow t'$ specifying a *rewrite* theory, which may be nonconfluent, nonterminating, and nondeterministic. System modules can naturally model many types of concurrent systems [14].

By exploiting the *rewriting semantics* of Maude system modules, we can write a *system specification* which imports the (functional) *property specification* of a conceptual data model and its constraints and specifies a system allowing concurrent edits to the data model's *state* while enforcing its constraints. In this way Maude's rewriting semantics yields an interpreter "for free" for the specific CDM "language" defined in the property specification.

Using code from [3], we represent requested edits plus the model's state as a pair or *configuration* whose first component is a *program* (containing unprocessed "messages" requesting to insert or delete objects or facts) and whose second component is a *record* (containing the "actual" objects and facts in the state):

```
fmod CONF is pr PROGRAM . pr RECORD .
 sort Conf .
 op <_,_> : Program Record -> Conf [ctor format(n d n s++ n d)] .
endfm
```

The sort `Program` (resp. `Record`) has a "record" structure composed of one or more fields each constructed using an index paired with a component, specified by the modules below, imported by `CONF`. Record *inheritance* makes `Programs` and `Records` modular (extensible by adding more fields). A `Program` or `Record` with more fields is a special case of one having a subset of those fields.

```
fmod PROGRAM is
 sorts PIndex PComponent PField PreProgram Program PTruth .
 subsorts PField < PreProgram .
 op null : -> PreProgram [ctor] .
 op _,_ : PreProgram PreProgram -> PreProgram
   [ctor assoc comm id: null] .
 op {_} : [PreProgram] -> [Program] [ctor format(s s++ --s s)] .
 op _:_ : [PIndex] [PComponent] -> [PField] [ctor] .
 op tt : -> PTruth .
 op duplicated : [PreProgram] -> [PTruth] .
 var I : PIndex . var C C' : PComponent . var P? : PreProgram .
 eq duplicated( ( I : C ) , ( I : C' ) , P? ) = tt .
 cmb { P? } : Program if duplicated( P? ) =/= tt .
endfm

fmod RECORD is pr PROGRAM
 * ( sort PIndex to RIndex ,
     sort PComponent to RComponent ,
     sort PField to RField ,
     sort PreProgram to PreRecord ,
     sort Program to Record ,
     sort PTruth to RTruth ) .
endfm
```

We will use the same syntax for *actual* objects and facts, as well as *messages* requesting to insert or delete objects and facts. Objects and facts in the `ins` (resp. `del`) field of the `Program` component of a configuration will be interpreted as *insert* (resp. *delete*) *messages*, and in the `db` field in the `Record` component as *actual* objects and facts in the current database state. This is done by making sort `PreCdm` a subsort of program component sort `PComponent`, and sort `Cdm` a subsort of record component sort `RComponent`:

```
fmod CDM.PROGRAM is pr PROGRAM . pr PRECDM .
 subsort PreCdm < PComponent .
 ops ins del : -> PIndex .
 var M : PreCdm . mb (ins : M) : PField . mb (del : M) : PField .
endfm

fmod CDM.RECORD is pr RECORD . pr CDM .
 subsort Cdm < RComponent .
 op db : -> RIndex [format (d s)] .
 var M : Cdm . mb (db : M) : RField .
endfm
```

An insert or delete message should result in the insertion or deletion of an actual object or fact only if allowed by the constraints. Because they may contain constraint-violating messages, the components in the `ins` and `del` fields contain terms of sort `PreCdm` rather than sort `Cdm`. Messages which would violate the constraints are not rewritten. Only the `db` field contains a term of sort `Cdm` representing the actual objects and facts in the current, valid database state.

More subtly, there may be messages in the `ins` and `del` fields which, if executed *individually* (in isolation from other messages), would violate the model's constraints, but which would satisfy them if executed *collectively* — *i.e.*, simultaneously with other messages. For example, to execute a message inserting a *Dept*, we also need exactly one *Professor* to head it. The *Professor heads Dept* fact exists iff the *Dept* object exists, so the messages inserting the *Dept* object and the *Professor heads Dept* fact must execute simultaneously (or not at all).

Maude system modules provide *concurrent* or simultaneous rewriting semantics, so several concurrent state transitions can be performed simultaneously. To allow concurrent insertion and deletion of valid sets of objects and facts while enforcing the model's constraints, we extend system module `RCONF` [3]:

```
mod RCONF is extending CONF .
 op {_,_} : [Program] [Record] -> [Conf] [ctor] .
 op [_,_] : [Program] [Record] -> [Conf] [ctor] .
 vars P P' : Program . vars R R' : Record .
 crl [step] :  < P , R >  =>  < P' , R' >
  if           { P , R }  =>  [ P' , R' ] .
endm
```

by adding two rules labeled `[ins]` and `[del]` in system module `CDM.RCONF`:

```
mod CDM.RCONF is inc RCONF . ex CDM.PROGRAM . ex CDM.RECORD .
 var PM PM' RM : PreCdm .
 var P : PreProgram . var R : PreRecord .
 crl [ins] :
     { {(ins : (PM ; PM')) , P} , {(db : {      RM }) , R} }
  => [ {(ins :       PM' ) , P} , {(db : { PM ; RM }) , R} ]
  if { PM ; RM } : Cdm /\ PM =/= nil .
 crl [del] :
     { {(del : (PM ; PM')) , P} , {(db : { PM ; RM }) , R} }
  => [ {(del :       PM' ) , P} , {(db : {      RM }) , R} ]
  if {      RM } : Cdm /\ PM =/= nil .
endm
```

While Maude *functional* module `CDM` provided the basis for a *property specification* describing what a given conceptual data model *is* (including any constraints that must hold), Maude *system* module `CDM.RCONF` exploits Maude's rewriting semantics to provide a *system specification* [14] describing what a conceptual data model *does* (*i.e.*, it allows concurrent edits to the database state while enforcing its constraints). This rewriting logic interpreter for conceptual data models with constraints is an example of a formal analysis and simulation tool obtained "for free" using Maude's rewriting semantics.

We now define a test population in `fmod UNI-TEST`, then define an interpreter in `mod UNI.RCONF` which just imports `UNI.CDM` and `CDM.RCONF`, and test it using system module `mod UNI.RCONF-TEST` which imports `UNI.RCONF` and `UNI-TEST`:

```
fmod UNI-TEST is pr UNI.CDM .
 mb person("Alice") : Teacher . mb person("Bob") : Professor .
 op uni1 : -> PreCdm .
 eq uni1 =
   person("Alice") ; person("Bob") ; dept("Math") ;
   person("Alice") works for dept("Math") ;
   person("Bob") works for dept("Math") ;
   person("Bob") heads dept("Math") ;
   person("Bob") is tenured ;
   person("Alice") audits person("Bob") ;
   person("Bob") audits person("Alice") ;
   dept("Math") offers course( dept("Math") , 101 ) ;
   course( dept("Math") , 101 ) has name "Calculus" .
endfm

mod UNI.RCONF is pr UNI.CDM . pr CDM.RCONF . endm
mod UNI.RCONF-TEST is pr UNI.RCONF . pr UNI-TEST . endm
```

The instances in `uni1`, interpreted as insert messages, are valid if collectively inserted into *e.g.* an empty database, so the first rewrite below should terminate with all instances "transferred" from the `ins` field to the `db` field in the fully rewritten term, leaving an empty `ins` field. The second rewrite should also terminate, but the `dept("English")` *insert message* should remain "unconsumed" in the `ins` field, because rewriting this message would violate the constraints.

```
rewrite < { ins : uni1 } , { db : {nil} } > .
rewrite < { ins : (uni1 ; dept("English")) } , { db : {nil} } > .
```

6 Discussion and Related Work

This work attempts to contribute to ongoing efforts to provide formal semantics and tool support for current programming practice, and is similar to work done using Maude to provide an algebraic semantics for UML+OCL class and object diagrams [10] and to specify the ODP information and enterprise viewpoints [11,12]. CDM diagrams (without constraints) can be regarded as a subset of UML class diagrams or of the ODP information viewpoint. The UML+OCL Maude effort has resulted in a formal tool, the ITP/OCL inductive theorem prover, which can be used to provide automatic generation and validation of object diagrams with respect to OCL constraints [10]. Unlike the approach in the current paper which represents a conceptual data model using a multiset configuration at Maude's level 0, the UML+OCL Maude effort uses an `FModule` at Maude's metalevel (level 1) to represent a UML diagram.

Tools using CDM or "extended" E/R methods to model relational databases include: VisioModeler [4] (commercial, formerly produced by InfoModelers, now part of Microsoft's Visio); Infagon [21] (open-source, uses FCO-IM); ERW [22]

(freeware, based on a bicategorical definition of multirelations); LISA-D [7,23] (developed at Nijmegen, based on category theory [1]); and the ontology editor ICOM [24] (freeware, Java/CORBA, EER, uses *description logic* to provide a provably complete inference mechanism for constraint consistency checking).

7 Conclusions and Further Work

This work has presented a method using Maude functional modules to represent a conceptual data model and its constraints, and using Maude system modules to obtain an interpreter providing operational semantics allowing constraint-enforcing concurrent edits to a conceptual data model's state.

It should be fairly straightforward to provide support for *update* messages in addition to insert and delete messages, as well as CDM *derived fact types* [4,5]. Since `Rmap` [5,6] is deterministic, it could be implemented functionally, instead of relying on Maude's rewriting semantics to search for collectively valid sets of insert and delete messages. This would provide a *denotational semantics* translating a conceptual data model to an entity-relationship model, in addition to the *operational semantics* currently provided. The resulting E/R model could be represented using an *object-oriented module* in (Full) Maude.

Other important tasks include schema transformation and optimization [25] and constraint consistency checking, which could be approached using reflection at Maude's metalevel (level 1). Level 1 could also be used to generate the repetitive (level 0) equations for commonly used constraint classes.

References

1. Lippe, E., ter Hofstede, A.: A Category Theory Approach to Conceptual Data Modeling. In: RAIRO Theoretical Informatics and Applications. Faculty of Mathematics and Informatics, vol. 30, pp. 31–79. University of Nijmegen, Nijmegen, The Netherlands (1996)
2. Chen, P.: The Entity-Relationship Model: Toward a Unified View of Data. ACM Trans DB Sys. 1(1), 9–36 (1976)
3. Meseguer, J., Braga, C.: Modular Rewriting Semantics of Programming Languages. In: Rattray, C., Maharaj, S., Shankland, C. (eds.) AMAST 2004. LNCS, vol. 3116, pp. 364–378. Springer, Heidelberg (2004)
4. Halpin, T.: Object-Role Modeling (ORM/NIAM). In: Bernus, P., Mertins, K., Schmidt, G. (eds.) Handbook on Architectures of Information Systems, ch. 4, Springer, Berlin (1998)
5. Halpin, T.: Conceptual Schema and Relational Database Design, 2nd edn. Prentice-Hall, Sydney, Australia (1995)
6. Paulussen, G.: AIS1: Halpin hfdst 10. Relational Mapping: Implementing a Conceptual Schema. From FORML-Guide ch. 6 t/m BLZ 104 (2003), http://www.cs.ru.nl/G.Paulussen/AIS1/Sheets/SheetsWeek05Dinsdag.pdf
7. ter Hofstede, A., Proper, H., van der Weide, T.: Formal definition of a conceptual language for the description and manipulation of information models. Information Systems 18, 489–523 (1993)

8. Bakema, G., Zwart, J., van der Lek, H.: Fully Communication-Oriented NIAM. In: Nijssen, G., Sharp, J. (eds.) NIAM-ISDM 1994 Conf. Working Papers, Albuquerque, NM, USA, pp. L1–35 (1994)
9. Nijssen, G., Halpin, T.: Conceptual Schema and Relational Database Design: A Fact-Oriented Approach. Prentice-Hall, Sydney, Australia (1989)
10. Clavel, M., Egea, M.: An Algebraic Semantics for UML+OCL Class Diagrams. Universidad Complutense de Madrid, Spain, (2006) Available on the web at http://maude.sip.ucm.es/~marina/pubs/fase06.pdf
11. Durán, F., Vallecillo, A.: Writing ODP Information Specifications in Maude. Technical Report ITI-2001-10, Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Málaga, Spain (2001) Available at http://www.lcc.uma.es/~av/Publicaciones/01/ITI-2001-10.pdf
12. Durán, F., Vallecillo, A.: Writing ODP Enterprise Specifications in Maude. In: Cordeiro, J., Kilov, H., (eds.) Proc. of WOODPECKER'01, Setubal, Portugal, pp. 55–68 (2001) An extended version is available as technical report at http://www.lcc.uma.es/~av/Publicaciones/01/ITI-2001-8.pdf
13. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: Maude Manual (Version 2.2). SRI International, Menlo Park, CA, USA (2005)
14. Meseguer, J.: Software Specification and Verification in Rewriting Logic. In: Models, Algebras, and Logic of Engineering Software, NATO Advanced Study Institute, pp. 133–193. IOS Press, Amsterdam (2003)
15. Goguen, J., Winkler, T., Meseguer, J., Futatsugi, K., Jouannaud, J.P.: Introducing OBJ. In: Goguen, J., Malcolm, G. (eds.) Software Engineering with OBJ: Algebraic Specification in Action, Kluwer Academic Publishers, Dordrecht (2000)
16. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (ed.) WADT 1997. LNCS, vol. 1376, pp. 18–61. Springer, Heidelberg (1998)
17. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Comput. Sci. 96, 73–155 (1992)
18. Vigna, S.: Multirelational semantics for extended entity-relationship schemata with applications. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 35–49. Springer, Heidelberg (2002)
19. Bruni, R., Gaducci, F.: Some algebraic laws for spans (and their connections with multi-relations). In: Kahl, W., Parnas, D., Schmidt, G. (eds.) Relational Methods in Software. Electronic Notes in Theoretical Computer Science, vol. 44(3), Elsevier, Amsterdam (2001)
20. Barbosa, L.: A Brief Introduction to Bicategories. Technical Report DI-PURe-03:12:10, Departamento de Informática da Universidade do Minho, Campus de Gualtar, Braga, Portugal (2003)
21. Mattic Software (Infagon 5.0) available on the web at http://www.infagon.com
22. Vigna, S.: ERW: Entities and relationships on the web. In: Poster Proc. of 11th International World Wide Web Conference, Honolulu, USA (2002)
23. van der Weide, T.: Domain Modeling: The systematic construction of an ontology. In: The DaVinci Series of Lecture Notes: The Art & Craft of Information Systems Engineering, Radboud University Nijmegen, Nijmegen, The Netherlands (2005)
24. Franconi, E., Ng, G.: ICOM Intelligent Conceptual Modelling Tool, Version 1.1 Manual (Draft) http://www.cs.man.ac.uk/~franconi/icom/
25. Halpin, T., Proper, H.: Database Schema Transformation & Optimization. In: Papazoglou, M.M.P. (ed.) ER 1995 and OOER 1995. LNCS, vol. 1021, Springer, Heidelberg (1995)

Datatypes in Memory

David Aspinall[1] and Piotr Hoffman[2]

[1] LFCS, School of Informatics, University of Edinburgh, U.K.
[2] Institute of Informatics, Warsaw University, Poland

Abstract. Besides functional correctness, specifications must describe other properties of permissible implementations. We want to use simple algebraic techniques to specify *resource usage* alongside functional behaviour. In this paper we examine the space behaviour of datatypes, which depends on the representation of values in memory. In particular, it varies according to how much values are allowed to overlap, and how much they must be kept apart to ensure correctness for destructive space-reusing operations.

We introduce a mechanism for specifying datatypes represented in a memory, with operations that may be destructive to varying degrees. We start from an abstract model notion for data-in-memory and then show how to specify the observable behaviour of models. The method is demonstrated by specifications of lists-in-memory and pointers; with a suitable definition of implementation, we show that lists-in-memory may be implemented by pointers. We then present a method for proving implementations correct and show that it is sound and, under certain assumptions, complete.

1 Introduction

This paper is part of an investigation into using simple algebraic techniques to write specifications of *resource usage* alongside functional correctness, where resources are quantitative measures such as time, space, power, and the like. Resource usage is of course a relative notion, and depends on the computation mechanism of an underlying machine as well as the representation of data on that machine. We would like to write specifications which are as abstract as possible with respect to these low-level details, but which nonetheless are able to distinguish usefully between different algorithms and representations which are not distinguished by classical algebraic specifications.

We start off here by considering *memory* as the prototypical resource, and consider the behaviour of datatype operations which are implemented in memory. Many standard algorithms make use of shared mutable data structures; these algorithms have quite different resource usage behaviour compared with functional versions that copy data instead. For mutating algorithms to work correctly, the layout in memory of the data structures must satisfy certain conditions; for example, some parts of these structures may occupy the same memory cells, some may not. We provide a mechanism to specify layout constraints, which enable

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 111–125, 2007.
© Springer-Verlag Berlin Heidelberg 2007

or prevent mutating algorithms, and so restrict the class of models of our specifications to ones which have certain resource behaviours.

We specify layout constraints by using *preservation* and *disjointness* predicates which restrict the use of sharing in implementations. Intuitively, a memory-altering operation *preserves* some data object in memory if after executing it the object is still available in the new memory. Otherwise, the object is destroyed and the operation considered *destructive*. If two objects are *disjoint* (separate) in the memory, then destructive operations on the first object cannot affect the second. Motivating examples follow in the body of the paper.

A key insight is that we need only be concerned by the *behavioural*, or observable, consequences of a given data layout, not by the layout itself in a concrete model. Both preservation and disjointness have behavioural characterisations, and it turns out that for standard heap models these abstract characterisations coincide with the natural, naive notions. We get sensible results for other memory models as well, which allow comparison between functional and imperative implementations within the same logical framework, for example.

As well as mechanisms for specifying data structures, we define a notion of *implementation*. Using a simple form of program, we show how to implement one data structure in terms of another. We then give some basic ideas for proving implementations correct by a form of equational reasoning. We show that this method is sound, and, with restrictions, complete.

Contributions and related work. As far as we are aware, this is the first explicit attempt to study an approach to datatype space usage using algebraic specification methods. There has been a wealth of recent activity in program logics for pointer implementations datatypes in concrete memory models and type theories or analyses for shape and layout description (to mention only a few, e.g. [1,2,3,4,5]). Notably, Bunched Implications, BI, provides an abstract model theory for resources, as well as a substructural logic for describing models [6]. Although we also aim at an abstract approach, we intentionally work from first principles within the algebraic framework, rather than try to recast lines of work based on different semantic foundations. More comments on related work are in the conclusions.

Outline. The structure of the rest of this paper is as follows. In Sect. 2 we introduce the abstract algebraic framework and two canonical example algebras. In Sect. 3 we define the central behavioural equivalence relation which we use as a basis for both specification and reasoning. The relation can express preservation of data at the same time as behaviour of operations. We apply this to a specification of lists in Sect. 4, which we write in a specially defined behavioural version of conditional equational logic. Sect. 5 then shows how to use the behavioural approach to define a natural *disjointness relation* which is also useful in specifications. Sect. 6 gives a specification for pointers and in Sect. 7 we define a notion of implementation and show how pointers may be used to implement lists; we sketch how this may be proved formally and prove that our approach is sound and, in certain cases, complete. Sect. 8 concludes and gives some comparison with the related work.

2 Memory Signatures and Algebras

Definition 1. *A* memory signature *consists of the following components:*

- *disjoint sets of* abstract sorts *and* memory sorts*,*
- *a set of* abstract operations *of the form* $f : s_1 \times \cdots \times s_n \to t_1 \times \cdots \times t_k$ *where* $n, k \geq 0$ *and* $s_1, \ldots, s_n, t_1, \ldots, t_k$ *are abstract sorts,*
- *a set of* memory operations *of the form* $f : \mu \times (s_1 \times \cdots \times s_n) \to \mu \times (t_1 \times \cdots \times t_k)$ *where* μ *is a special symbol representing the memory and where* $n, k \geq 0$ *and* $s_1, \ldots, s_n, t_1, \ldots, t_k$ *are arbitrary sorts.*

The idea here is that objects of abstract sorts are directly observable, whereas objects of memory sorts can only be interpreted in the context of a memory via the memory operations. Abstract operations have a purely auxiliary function and are used in specifications. A memory operation is a form of "machine instruction", representing the actual steps of computation of the considered machine. The machine is modelled by a *memory algebra* over the signature.

Definition 2. *A memory algebra A consists of the following components:*
- *a non-empty set* $A[\mu]$ *of* memories*,*
- *for any sort* s*, a set* $A[s]$ *of* objects of type s*,*
- *for any operation* f*, a partial function* $A[f]$ *of appropriate type,*
- *for any memory sort* s*, a* validity predicate $A_V[s] \subseteq A[\mu] \times A[s]$.

If $(m, o) \in A_V[s]$ *we say* o is valid in m *and write* $o \in m$*. Validity in a memory is extended pointwise to tuples of objects of arbitrary sort, considering objects of abstract sort to be valid in any memory. A memory algebra must ensure that* memory operations preserve validity*, i.e., whenever* $\alpha \in m$ *and* $A[f](m, \alpha)$ *is defined and equal to* (m', β)*, then* $\beta \in m'$*.*

The partiality of memory operations is intended to represent errors or non-termination, but not out-of-memory exceptions. Although our approach is designed to deal with out-of-memory conditions, in this paper we assume that they do not occur. Out-of-memory exceptions can be included at the cost of some extra complexity, by adding another form of undefinedness so that non-termination and lack of memory can be distinguished.

Validity allows us to model the destruction of data. Any memory operation $f(m, \alpha)$ must produce a memory m' and output β valid in m', but we do not require the input α to remain valid in the new memory m'. Destructive operations, such as disposing a pointer, can destroy their own arguments.

We illustrate these definitions with concrete examples. Consider the memory signature with abstract sort `bool` and operations `t, f : bool` and with memory sort `list` and the following memory operations:

$\mathtt{nil} : \mu \to \mu \times \mathtt{list}$	$\mathtt{isnil} : \mu \times \mathtt{list} \to \mu \times \mathtt{bool}$
$\mathtt{cons} : \mu \times \mathtt{bool} \times \mathtt{list} \to \mu \times \mathtt{list}$	$\mathtt{hd} : \mu \times \mathtt{list} \to \mu \times \mathtt{bool}$
$\mathtt{tl} : \mu \times \mathtt{list} \to \mu \times \mathtt{list}$	$\mathtt{delete} : \mu \times \mathtt{list} \to \mu$

Define a memory algebra A over the above signature as follows. Let the abstract, boolean components be defined as usual, and let the memories all be sequences of pairs of natural numbers, which we treat as addresses: $A[\mu] \subseteq \mathbb{N} \to \mathbb{N}^2$. In other words, a memory is an infinite address space with two addresses, the first representing a boolean, stored at any location. If a is an address and m is a memory, then the *a-sequence in m* is the sequence $\{a_i\}_{i\in\mathbb{N}}$ defined by $a_0 = a$ and $a_{i+1} = \pi_2(m(a_i))$. Now define $A[\mu]$ to contain all $m \in \mathbb{N} \to \mathbb{N}^2$ such that the 0-sequence in m does not contain any repetitions. The 0-sequence is called the *free list*. Addresses in this sequence are called *free addresses*. Note that 0 is always free. Finally, let $A[\texttt{list}] = \mathbb{N}$ and define a list a to be valid in a memory m if in the a-sequence in m a free address occurs somewhere, and if the first such address is 0.

Now the memory operations are defined as follows on valid arguments:

- $\texttt{nil}(m) = (m, 0)$, and $\texttt{isnil}(m, a)$ is true if $a = 0$ and false otherwise;
- $\texttt{cons}(m, b, a) = (m', a')$, where m' is m with some free a' removed from the free list, with $m'(a') = (0, a)$ if b is false, and $m'(a') = (1, a)$ if b is true;
- $\texttt{hd}(m, a) = (m, \pi_1(m(a)))$ if $a \neq 0$; and $\texttt{tl}(m, a) = (m, \pi_2(m(a)))$ if $a \neq 0$;
- $\texttt{delete}(m, a) = m'$, where m' is m with all addresses from the a-sequence in m added to the free list.

Here, the free list at the 0-sequence is treated as a pool of memory for allocation and deallocation. In all cases not covered, the memory operations are undefined. We call this model of lists the *pointer model*.

Another model of lists is the algebra B with the same boolean component as A, but with $B[\mu] = \{*\}$ a singleton set and $B[\texttt{list}]$ the set of all finite sequences of booleans. Then all the list operations work just as regular list operations, except that they additionally return $*$ as the new memory. In particular, $\texttt{delete}(m, l) = m$ for all lists l. This model of lists is called the *algebraic model*.

3 Behavioural Equivalence

We now define a notion of behavioural equivalence for values in memory algebras. We do this by conceiving a memory algebra as a machine which contains a memory and a finite number of variables. The variables may keep data which is directly observable (of an abstract sort), or data that may only be interpreted using the memory (of a memory sort). The machine computes by applying a memory operation to the existing memory and data, thereby obtaining a new memory and new data. Two states of a machine should be considered equivalent if no sequence of steps of the machine leads to any difference in observable data.

Formally, a *state* of a memory algebra is a pair (m, γ), where m is a memory and $\gamma = (\gamma_1 : s_1, \ldots, \gamma_n : s_n)$ is a tuple of objects valid in m. Then n is the *length* of the state, and $(s_1, \ldots, s_n)$ is the *type* of the state. If $F : \{1, \ldots, N\} \to \{1, \ldots, n\}$ is any function, with n being the length of a state (m, γ) and N arbitrary, then the composition $(m, F(\gamma))$ defined by $F(\gamma)_i = \gamma_{F(i)}$ for $1 \leq i \leq N$ is a state as well. This is simply a rearrangement of the state (m, γ), possibly reordering, removing and duplicating objects. For any state (m, γ) and memory m', let $\gamma|_{m'}$

be the tuple obtained by removing from γ any objects not valid in m'. Of course, $(m', \gamma|_{m'})$ is a state. Finally, let "+" denote concatenation of tuples.

Definition 3. *The* behavioural equivalence $\sim$ *in a memory algebra A is the greatest relation on states of equal type such that if $(m_1, \gamma_1) \sim (m_2, \gamma_2)$ then:*

1. *if v_1, v_2 are abstract values in corresponding positions in γ_1, γ_2, then $v_1 = v_2$.*
2. *if v is abstract, then $(m_1, \gamma_1 + (v)) \sim (m_2, \gamma_2 + (v))$.*
3. *if γ_1 and γ_2 have length n and $F : \{1, \ldots, N\} \to \{1, \ldots, n\}$ is any function, then $(m_1, F(\gamma_1)) \sim (m_2, F(\gamma_2))$.*
4. *if $\gamma_1 = \eta_1 + \delta_1$ and $\gamma_2 = \eta_2 + \delta_2$, η_1 and η_2 have the same length and f is an appropriately typed memory operation, then $A[f](m_1, \eta_1)$ is defined iff $A[f](m_2, \eta_2)$ is. In this case, let $A[f](m_1, \eta_1) = (m_1', \eta_1')$ and $A[f](m_2, \eta_2) = (m_2', \eta_2')$; it is required that for all indices i, $(\gamma_1)_i$ is valid in m_1' iff $(\gamma_2)_i$ is valid in m_2' and $(m_1', \eta_1' + \gamma_1|_{m_1'}) \sim (m_2', \eta_2' + \gamma_2|_{m_2'})$.*

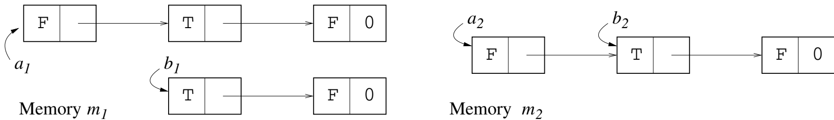
The above relation is well-defined and is an equivalence. Intuitively:

1. abstract sorts are observable.
2. one may at any moment add arbitrary variables of abstract sort.
3. one may rearrange the variables.
4. one may apply memory operations to valid objects, but any objects invalidated in doing so are removed from the state; both undefinedness and destruction of data are observable.

Because invalid variables are removed from the state in clause 4, we forbid a computation that holds a "dangling" pointer which later becomes valid again. This doesn't imply that some form of on-the-fly garbage collection is involved; it just means that programs cannot assume anything about invalidated objects, and specifications cannot express such assumptions.

In the algebraic model of lists there cannot be any interaction between two lists; we have $(m_1, \gamma_1) \sim (m_2, \gamma_2)$ iff $(m_1, (\gamma_1)_i) \sim (m_2, (\gamma_2)_i)$ for all $1 \le i \le n$, where n is the length of γ_1 and γ_2. In this case $\sim$ is simply the identity relation, because any pair of non-equal lists is differentiated by an appropriate number of `tl` operations and then a `hd` or `isnil` operation. This is true for any "non-destructive" model, even if non-equal but equivalent lists exist in it.

Things are different in the pointer model, where we have a destructive operation and there can be overlaps between lists. Consider the two memories:



Here $(m_1, (a_1, b_1)) \sim (m_2, (a_2, b_2))$ doesn't hold, although $(m_1, (a_1)) \sim (m_2, (a_2))$ and $(m_1, (b_1)) \sim (m_2, (b_2))$ both hold. This is because the lists in m_2 interfere in a way that can be observed by performing a `delete` operation.

For any state $\zeta = (m, (a_1 : \texttt{list}, \ldots, a_n : \texttt{list}))$ in the pointer model, let Φ_ζ take any pair $1 \le i, j \le n$ to the list of booleans which is kept in the maximal

common part of a_i and a_j in m (in particular, $\Phi_\zeta(i,i)$ is the list of booleans corresponding to the list a_i in m). Then $\sim$ is the kernel of Φ, that is, for any tuples ζ_1 and ζ_2 of length n of lists we have $\zeta_1 \sim \zeta_2$ iff $\Phi_{\zeta_1} = \Phi_{\zeta_2}$.

4 A Specification of Boolean Lists

We can specify memory algebras in two parts: a specification of the abstract part, and a specification of the memory part. We suppose that the abstract part (i.e., the booleans) is already specified, and concentrate on the memory part.

We use axioms of a very simple form, similar to conditional equational logic. Of course, this does not mean that more complex logics cannot be used in our approach. Formulae are given by the following grammar:

$$\begin{aligned}\phi ::= {} & (m,\alpha) \sim (m,\alpha) \mid f(m,\alpha) = \bot \mid f(m,\alpha) \neq \bot \mid \\ & x \in m \mid \forall m \cdot \phi \mid \forall x \cdot \phi \mid \phi \wedge \phi \mid \\ & x \in m \implies \phi \mid f(m,\alpha) \to (m,\alpha) \implies \phi\end{aligned}$$

Here m ranges over variables binding memories, x over other variables, and α over tuples of variables. Variable typing is assumed but left implicit. The variables x may be bound either to objects of a single sort or to finite tuples of objects; such variables will usually be named γ, δ, etc. So the quantification $\forall\gamma \cdot \gamma \in m \implies \phi$ says that for all tuples γ of objects, if all these objects are valid in m, then ϕ holds; we abbreviate this by writing $\forall\gamma \in m \cdot \phi$. The formula $f(m, x_1, \ldots, x_n) \to (m', y_1, \ldots, y_k) \implies \phi$ is true if whenever $f(m, x_1, \ldots, x_n)$ is defined, then after binding the result to m' and $y_1, \ldots, y_k$, the formula ϕ holds. If a variable y_i is not used in ϕ, we may write $_$ instead of y_i.

As syntactic sugar we use equality, $x = y$, if x and y are of an abstract sort. This can be expressed using the relation $\sim$ as $\forall m \cdot (m,x) \sim (m,y)$. For memory sorts, our axioms make more use of the behavioural equivalence: we write $m \lesssim m'$ as a shorthand for the *non-destructiveness assertion* $\forall\gamma \in m \cdot (m,\gamma) \sim (m',\gamma)$, which says that all objects of m are preserved (observationally) in m'. To specify additionally that some object a in m behaves equivalently to a' in m' we write $(m,a) \lesssim (m',a')$ as a shorthand for $\forall\gamma \in m \cdot (m,a,\gamma) \sim (m',a',\gamma))$. This generalises in the obvious way to a tuple of objects.

The specification of boolean lists begins with the following three axioms:

$$\forall m \cdot \mathtt{nil}(m) \neq \bot \ \wedge \ (\mathtt{nil}(m) \to (m', _) \implies m \lesssim m') \tag{1}$$

$$\forall m \forall l \in m \cdot \mathtt{isnil}(m,l) \neq \bot \ \wedge \ (\mathtt{isnil}(m,l) \to (m', _) \implies m \lesssim m') \tag{2}$$

$$\forall m \forall b \forall l \in m \cdot \mathtt{cons}(m,b,l) \neq \bot \ \wedge \ (\mathtt{cons}(m,b,l) \to (m', _) \implies m \lesssim m') \tag{3}$$

These say that `nil`, `isnil` and `cons` are always defined on valid arguments, and that they are non-destructive. This does not mean that preexisting objects can't be changed at all: that can happen, so long as the result is behaviourally equivalent to the original. Next we specify the behaviour of `isnil`, `hd` and `tl`:

$$\forall m \cdot \mathtt{nil}(m) \to (m', l) \implies \mathtt{isnil}(m', l) \to (_, b) \implies b = \mathtt{t} \tag{4}$$

$$\forall m \forall b \forall l \in m \cdot \texttt{cons}(m,b,l) \to (m',l') \implies \texttt{isnil}(m',l') \to (_,b) \implies b = \texttt{f} \tag{5}$$

$$\forall m \forall b \forall l \in m \cdot \texttt{cons}(m,b,l) \to (m',l') \implies \texttt{hd}(m',l') \neq \bot \wedge \\ (\texttt{hd}(m',l') \to (m'',b') \implies (m',b) \lesssim (m'',b')) \tag{6}$$

$$\forall m \forall b \forall l \in m \cdot \texttt{cons}(m,b,l) \to (m',l') \implies \texttt{tl}(m',l') \neq \bot \wedge \\ (\texttt{tl}(m',l') \to (m'',l'') \implies (m',l) \lesssim (m'',l'')) \tag{7}$$

Note that axiom (6) says not only that the correct boolean value is returned, but also that `hd` is non-destructive. Axiom (7) is even stronger: `tl` is non-destructive and the produced tail must fully share with the original tail.

We can give alternative axioms for `hd` and `tl` that specify different amounts of destructiveness. If instead of axiom (6) we wrote:

$$\forall m \forall b \forall l \in m \cdot \texttt{cons}(m,b,l) \to (m',l') \implies \texttt{hd}(m',l') \neq \bot \wedge \\ (\texttt{hd}(m',l') \to (m'',b') \implies (m,b) \lesssim (m'',b'))$$

then we would obtain lists in which `hd` is allowed (though not forced) to destroy or modify the head of the list. Similarly, we could allow `tl` to destroy or modify the head of the list when computing the tail. Yet more possibilities exist, e.g., one could allow `tl` to fully destroy the old list. This would need an axiom somewhat similar to the (forthcoming) axioms for `delete`, plus an axiom of the form:

$$\forall m \forall b \forall l \in m \cdot \texttt{cons}(m,b,l) \to (m',l') \implies \texttt{tl}(m',l') \to (m'',l'') \implies \\ (m,l) \sim (m'',l'')$$

As for `delete`, the task would be easy with a *disjointness predicate* $o_1 \bot_m o_2$ stating that two given objects are disjoint in a memory m (with respect to a set of operations, see next section). Separation here means that manipulating o_1 cannot have an effect on o_2 and vice versa. Using this we could write:

$$\forall m \forall l \in m \cdot \texttt{delete}(m,l) \neq \bot$$

$$\forall m_0 \cdot \texttt{nil}(m_0) \to (m,l) \implies \forall \gamma \in m_0 \cdot \gamma \bot_m l$$

The second formula states that a nil list and any further manipulation of it (e.g. by `cons` and then `delete`), cannot have any effect on preexisting objects. The separation predicate is introduced in the next section. However, without it we can specify `delete` by the two axioms:

$$\forall m_0 \cdot \texttt{nil}(m_0) \to (m,l) \implies \texttt{delete}(m,l) \neq \bot \wedge \\ (\texttt{delete}(m,l) \to m' \implies m_0 \lesssim m') \tag{8}$$

$$\forall m \forall b \forall l \in m \cdot \texttt{cons}(m,b,l) \to (m',l') \implies \texttt{delete}(m,l) \to m_0'' \implies \\ \texttt{delete}(m',l') \neq \bot \ \wedge \ (\texttt{delete}(m',l') \to m'' \implies m_0'' \lesssim m'') \tag{9}$$

Axiom (8) says that deleting the nil list does not destroy preexisting objects. Axiom (9) says that if we add an element and then delete the list, then objects

that wouldn't have been destroyed if we deleted the list without adding the element will be left intact. Thus, axiom (9) allows us to show that deleting a longer list is like deleting a nil list, and axiom (8) shows that deleting a nil list does not destruct unrelated objects. Together, they guarantee that preexisting objects will be retained. Clearly these axioms do not force the model to be a destructive one; `delete` may be a dummy operation, as in the algebraic list model. But this would change if we introduced methods for counting the amount of used memory, for example; then we could specify that `delete` decreases the amount of memory used.

Our axiomatization of lists of booleans can be easily extended to an axiomatization of lists in which both booleans and other lists may be stored. In effect, this would be an axiomatization of directed acyclic graphs (dags) — a datatype in which sharing is really essential.

One could argue that our axioms are apparently rather complex for such a simple datatype as lists. However, many of these axioms have a regular form (e.g., idioms for non-destructiveness) and we could use further shorthands. More importantly, we would claim that there is a range of non-equivalent specifications of lists-in-memory, usefully describing different degrees of destructiveness, so the axioms need to be complex enough to capture these differences.

5 Specifying Disjointness

A notion of disjointness is useful in specifying memory operations. The previous section demonstrated this for the `delete` operation. Another example is a `copy` operation, which should produce a new and disjoint copy of a given list. Using our predicate for disjointness, this is captured by:

$$\forall m \forall l \in m \cdot \mathtt{copy}(m,l) \neq \bot \ \wedge (\mathtt{copy}(m,l) \to (m', _) \implies m \lesssim m')$$
$$\forall m \forall l \in m \cdot \mathtt{copy}(m,l) \to (m', l') \implies (m,l) \sim (m',l') \wedge \forall \gamma \in m \cdot l' \perp_{m'} \gamma$$

In a concrete model such as the pointer model, disjointness has a clear meaning. Pleasingly, it turns out that disjointness may be defined in an abstract, behavioural manner for any memory algebra. This abstract notion, when applied to pointer models of datatypes, yields the expected form of separation, and when applied to other, e.g., functional models, also gives very natural results.

The disjointness predicate has the form $\gamma \perp_m \delta$, where γ and δ are tuples of of objects valid in m. Formally, we add the following new formula:

$$\phi ::= \ldots \mid \alpha \perp_m \alpha$$

We define the interpretation of disjointness coinductively using non-interference: two objects may be treated as disjoint if manipulations on one of them cannot affect the other, and vice versa. In particular, the disjointness of two objects is relative to the operations one may use on them.

Definition 4. *Let $\mathcal{F}$ be a set of memory operations from a memory signature.* Disjointness with respect to $\mathcal{F}$ *is the greatest memory-indexed family of symmetric relations on valid tuples of objects and such that if $\gamma \perp_m \delta$, then:*

- *if v is an object of abstract sort, then $\gamma + (v)\perp_m \delta$,*
- *if γ is of length n, $F : \{1, \ldots, N\} \to \{1, \ldots, n\}$ is any function and $f \in \mathcal{F}$ is of the appropriate type, and if $A[f](m, F(\gamma)) = (m', \gamma')$, then: (i) δ is valid in m', (ii) $(m', \delta) \sim (m, \delta)$, and (iii) $(\gamma|_{m'} + \gamma')\perp_{m'}\delta$.*

Definition 4 imposes the *frame condition* that manipulation of objects cannot affect other objects which are disjoint in the same memory. However, because the disjointness notion is relativised to a particular memory, it is possible for an operation to destructively combine two objects from one memory, and, in the new memory, for those objects to be no longer disjoint, or to become invalid. An example of the second case is the familiar destructive append, which invalidates the first object; an example of the first case is `smash_tails`(l_1, l_2) which coalesces two disjoint objects l_1 and l_2 so that they share the longest possible suffix.

Disjointness is anti-monotone with respect to the set of operations $\mathcal{F}$, and for $\mathcal{F} = \emptyset$ all valid tuples are disjoint in all memories. When it is left implicit, we take all memory operations to be in $\mathcal{F}$. In the pointer model of lists, disjointness w.r.t. all the operations is just real disjointness of two lists in memory. In particular, any list is disjoint with the nil list 0. In the algebraic model of lists, meanwhile, the disjointness relation is the total relation — we even have $l\perp_m l$ for any list l. All this is not surprising. But if, in the pointer model, we consider disjointness w.r.t. operations other than `delete`, then we also get the total relation. This is true even when the "real" lists overlap in memory — the overlap cannot have consequence and so is ignored. One could claim that a model deserves the name "functional" just in case its disjointness relation is total.

At the other end of the spectrum, suppose we add `zero_memory` $: \mu \to \mu$ to our signature, and implement it in the pointer model as a constant function which takes any memory m to a memory in which all addresses are on the free list. If we consider disjointness w.r.t. the set $\mathcal{F} = \{$`zero_memory`$\}$, then only pairs of sequences of nil lists and booleans are disjoint. We don't have $0\perp_m l$, where 0 is the empty list and l is non-empty, since zeroing memory on the left side invalidates l. If we now set $\mathcal{F}$ to contain all of the operations, then even $()\perp_m()$ no longer holds, since it is possible to first produce a non-empty list and then invalidate it, as suggested above.

6 A Specification of Pointers

Now we specify a datatype of pointers. This is a purely "imperative" datatype, with no functional aspects. In the next section we show how to use it to implement datatypes such as lists. The signature has a sort `pointer` and operations:

`0` $: \mu \to \mu \times$ `pointer`
`isO` $: \mu \times$ `pointer` $\to \mu \times$ `bool`
`new` $: \mu \to \mu \times$ `pointer`
`dispose` $: \mu \times$ `pointer` $\to \mu$
$\texttt{set}_1, \texttt{set}_2 : \mu \times \texttt{pointer} \times \texttt{pointer} \to \mu$
$\texttt{setbool}_1, \texttt{setbool}_2 : \mu \times \texttt{pointer} \times \texttt{bool} \to \mu$
$\texttt{val}_1, \texttt{val}_2 : \mu \times \texttt{pointer} \to \mu \times \texttt{pointer}$
$\texttt{valbool}_1, \texttt{valbool}_2 : \mu \times \texttt{pointer} \to \mu \times \texttt{bool}$

The operation `0` returns a special, constant pointer, and `is0` tests identity of this pointer. In any pointer two values may be stored and retrieved, each being either another pointer or a boolean. Pointers may be created and disposed of.

The axioms specifying pointers are as follows:

$$\forall m \cdot \mathtt{new}(m) \neq \bot \;\wedge\; (\mathtt{new}(m) \to (m', _) \implies m \lesssim m') \tag{10}$$

$$\forall m \forall p \in m \cdot \mathtt{val}_i(m,p) \neq \bot \;\wedge\; (\mathtt{val}_i(m,p) \to (m', _) \implies m \lesssim m') \tag{11}$$

$$\forall m_0 \cdot \mathtt{0}(m_0) \to (m,p) \implies \mathtt{is0}(m,p) \to (_, b) \implies b = \mathtt{t} \tag{12}$$

$$\forall m_0 \cdot \mathtt{new}(m_0) \to (m,p) \implies \mathtt{is0}(m,p) \to (_, b) \implies b = \mathtt{f} \tag{13}$$

These are similar to ones given earlier for list operations. An axiom analogous to (10) is required for `0` and ones analogous to (11) for `is0` and for `valbool`.

$$\begin{aligned} \forall m_0 \cdot \mathtt{new}(m_0) \to (m,p) \implies \forall q \in m \cdot \mathtt{set}_i(m,p,q) \neq \bot \wedge \\ (\mathtt{set}_i(m,p,q) \to m' \implies m_0 \lesssim m') \quad (14) \\ \forall m_0 \cdot \mathtt{new}(m_0) \to (m,p) \implies \mathtt{dispose}(m,p) \neq \bot \wedge \\ (\mathtt{dispose}(m,p) \to m' \implies m_0 \lesssim m') \quad (15) \\ \forall m_1 \cdot \mathtt{new}(m_1) \to (m_2,p) \implies \forall q \in m_2 \cdot \mathtt{set}_i(m_2,p,q) \to m_3 \implies \\ \mathtt{val}_i(m_3,p) \to (m_4,q') \implies (m_3,q) \lesssim (m_4,q') \quad (16) \end{aligned}$$

Axiom (14) says that storing in a pointer does not modify objects that existed before the pointer was created; a similar axiom is needed for `setbool`. Axiom (15) says that disposing a pointer does not affect objects that existed before its creation. Axiom (16) and a similar axiom for the boolean case guarantee that storing an object and then loading it gives the same (or a behaviourally equivalent) object back. Behavioural equivalence for pointers is crucial: it means that an implementation may keep internal information (e.g., needed for garbage collection) in pointers, as long as the outside world cannot access it.

We can define a model for pointers by altering the model of lists as follows:

– memories must keep both addresses 0 and 1 on the free list (they are used to denote booleans), and to keep 2 off the free list, initialised to $(0,0)$ (it will be used as the `0` pointer),
– the set of valid pointers in m is the greatest set of addresses not on the free list and such that if a is valid and $m(a) = (a_1, a_2)$, then $a_i = 0$, $a_i = 1$ or a_i is valid in m (for $i = 1, 2$),
– `0` returns the address 2; `is0` returns true called on 2, false otherwise,
– `new` allocates a new pointer and sets its fields to 0,
– $\mathtt{val}_i$ and $\mathtt{set}_i$ just return and set the appropriate fields under the given address; the boolean versions do the same, with 0 as false and 1 as true,
– `dispose` adds the given address to the free list.

One could also imagine a “lazy” model, in which `dispose` would defer its work, adding addresses to the free list later on when other operations are called.

7 Implementations

So far we haven't said how memory specifications may be *implemented*, that is, how one can define memory algebras. We now show how this can be done, and show how the correctness of implementations may be proved. We do not consider how to construct implementations "ex nihilo", but rather how to implement one datatype making use of other datatypes. This approach is reasonable, because one may assume that simple datatypes are given as built-in.

Let Δ be a memory signature containing the abstract sort `bool`. *Programs* over Δ are expressions P_λ of the following form:

$$P_\lambda ::= \lambda(\alpha) \cdot P$$
$$P \ ::= P; P \mid x \to x \mid f(\alpha) \to (\alpha) \mid \text{if } e \text{ then } P \mid \text{return}(\alpha) \mid \text{self}(\alpha)$$

Here, α denotes tuples of variables, implicitly typed by sorts from Δ, e denotes boolean expressions built using abstract operations and variables of abstract sorts, and f denotes memory operations from Δ. No variable is allowed to appear on the right hand side of the binding "$\to$" more than once. The instruction "self(α)" recursively calls the program being run, while "return(α)" terminates the computation (all recursive calls). It is required that the last instruction in any program is either a self, or a return. Type-soundness is also enforced, i.e., if a memory operation f is invoked with arguments being variables of sorts $s_1, \ldots, s_n$ and results being variables of sorts $t_1, \ldots, t_k$, then we have $f : \mu \times (s_1 \times \cdots \times s_n) \to \mu \times (t_1 \times \cdots \times t_k)$ in Δ. A program has *type* $s_1 \times \cdots \times s_n \to t_1 \times \cdots \times t_k$ if it binds, under the λ, variables of sorts $s_1, \ldots, s_n$, if it passes variables of such types via self, and if it invokes return with variables of type $t_1, \ldots, t_k$.

An *implementation* of a memory signature Σ by Δ is a map I taking any memory sort in Σ to a memory sort in Δ, and any memory operation $f : \mu \times (s_1 \times \cdots \times s_n) \to \mu \times (t_1 \times \cdots \times t_k)$ to a program over the signature Δ of type $I(s_1) \times \cdots \times I(s_n) \to I(t_1) \times \cdots \times I(t_k)$.

For example, suppose we want to implement lists, as defined in Sect. 3, using pointers, as defined in the previous section. We can define this by:

$$
\begin{array}{ll}
\texttt{list} \ := \texttt{pointer} & \texttt{hd} \ := \lambda(p) \cdot \texttt{valbool}_1(p) \to b;\ \text{return}(b) \\
\texttt{nil} \ := \lambda() \cdot \texttt{0} \to p;\ \text{return}(p) & \texttt{tl} \ := \lambda(p) \cdot \texttt{val}_2(p) \to p';\ \text{return}(p') \\
\texttt{isnil} := \lambda(p) \cdot \texttt{is0}(p) \to b;\ \text{return}(b) & \texttt{delete} := \lambda(p) \cdot \texttt{is0}(p) \to b; \\
\texttt{cons} \ := \lambda(b,p) \cdot \texttt{new} \to p_1; & \quad \text{if } b \text{ then return}(); \\
\quad \texttt{setbool}_1(p_1, b) \to (); & \quad \texttt{val}_2(p) \to p'; \\
\quad \texttt{set}_2(p_1, p) \to (); & \quad \texttt{dispose}(p) \to (); \\
\quad \text{return}(p_1) & \quad \text{self}(p')
\end{array}
$$

We define the semantics of programs in the obvious way, with infinitely looping programs causing non-definedeness, $\perp$. The semantics of programs induces a semantics of implementations: for any implementation $I : \Sigma \to \Delta$ and any memory algebra B over Δ, the semantics of Δ-programs gives us a memory algebra $B|_I$ over Σ.

Proving that implementations are correct. An implementation of lists by pointers is *correct*, if assuming that the pointer axioms of Sect. 5 hold, then so do the

list axioms of Sect. 3. This will guarantee that if B satisfies the pointer axioms, then $B|_I$ satisfies the list axioms.

For any implementation $I : \Sigma \to \Delta$ one can define a set $\mathrm{Sen}(I)$ of formulas over an extended signature $\Delta \cup \Sigma$ which define the operations in Σ. For example, the definition of `nil` leads to the two formulae:

$$\begin{aligned}
&\forall m \cdot \mathtt{nil}(m) \to (m', p) \implies \mathtt{0}(m) \neq \bot \wedge \\
&\qquad (\mathtt{0}(m) \to (m'', p') \implies (m', p) \lesssim (m'', p')) \\
&\forall m \cdot \mathtt{0}(m) \to (m', p) \implies \mathtt{nil}(m) \neq \bot \wedge \\
&\qquad (\mathtt{nil}(m) \to (m'', p') \implies (m', p) \lesssim (m'', p'))
\end{aligned}$$

In a similar manner, definitions of other operations may be generated. Armed with these formulas and the axioms defining pointers, we may now attempt to prove the axioms defining lists. Consider, for example, axiom (6). Thanks to the definition of `cons` and `hd` in $\mathrm{Sen}(I)$, this is equivalent to:

$$\begin{aligned}
&\forall m \forall b \forall p \in m \cdot \mathtt{new}(m) \to (m_1, p') \implies \mathtt{setbool}_1(m_1, p', b) \to m_2 \implies \\
&\qquad \mathtt{set}_2(m_2, p', p) \to m' \implies \mathtt{valbool}_1(m', p') \neq \bot \wedge \\
&\qquad\qquad (\mathtt{valbool}_1(m', p') \to (m'', b') \implies (m', b) \lesssim (m'', b')
\end{aligned}$$

This is indeed a consequence of the pointer axioms. Next consider axiom (8). It is equivalent to the following two formulas, corresponding to two branches of the if in `delete`'s definition:

$$\forall m_0 \cdot \mathtt{0}(m_0) \to (m, p) \implies \mathtt{is0}(m, p) \neq \bot \;\wedge\; (\mathtt{is0}(m, p) \to (m', b) \implies b = \mathtt{t} \implies m_0 \lesssim m') \quad (17)$$

$$\begin{aligned}
&\forall m_0 \cdot \mathtt{0}(m_0) \to (m, p) \implies \mathtt{is0}(m, p) \neq \bot \;\wedge\; (\mathtt{is0}(m, p) \to (m_1, b) \implies \\
&\qquad b = \mathtt{f} \implies \mathtt{val}_2(m_1, p) \to (m_2, p') \implies \mathtt{dispose}(m_2, p) \to m_3 \implies \\
&\qquad\qquad \mathtt{delete}(m_3, p') \to m' \implies m_0 \lesssim m') \quad (18)
\end{aligned}$$

In this simple case, because the second branch of the if always holds, we can prove the second formula even without again using the definition of `delete`, which may be found in $\mathrm{Sen}(I)$. But in general, we may repeatedly use the definitions in $\mathrm{Sen}(I)$ — this is, for example, necessary when proving that axiom (9) holds.

It can be shown that the method presented above is indeed sound. The notation $\Phi \models_\Sigma \varphi$ below means that any Σ-algebra satisfying all the formulas from the set Φ satisfies φ as well. The notation $\Phi|_I \models_\Sigma \varphi$, where $I : \Sigma \to \Delta$ is an implementation, Φ is a set of Δ-formulas and φ is a Σ-formula means that for any Δ-algebra B satisfying all the formulas from Φ, the Σ-algebra $B|_I$ satisfies the formula φ. We say that I *is an identity* on a subsignature Σ_0 of Σ if it is an identity on sorts and if, for any symbol f in Σ_0, we have $I(f) = \lambda(\alpha) \cdot f(\alpha) \to (\beta); \mathrm{return}(\beta)$.

Theorem 1. *If $I : \Sigma \to \Delta$ is an implementation which is an identity on $\Sigma \cap \Delta$, Φ is a set of Δ-formulas and φ is a Σ-formula, and if $\Phi \cup \mathrm{Sen}(I) \models_{\Sigma \cup \Delta} \varphi$ then $\Phi|_I \models_\Sigma \varphi$.*

Proof (sketch). Assume B satisfies $B \models_{\Delta} \Phi$. Let B' be the union (amalgamation) of B and $B|_I$, i.e., an algebra over $\Sigma \cup \Delta$. This union may be formed, since B and $B|_I$ coincide on $\Sigma \cap \Delta$. All observations in B' are also observations in B, because operations in $B|_I$ are defined in terms of operations of B. Therefore B and B' satisfy the same Δ-formulas; in particular, $B' \models_{\Sigma \cup \Delta} \Phi$. By construction of $\mathrm{Sen}(I)$ we also have $B' \models_{\Sigma \cup \Delta} \mathrm{Sen}(I)$. By assumption we then have $B' \models_{\Sigma \cup \Delta} \varphi$. It can be shown by induction on φ that this implies $B' \models_{\Sigma} \varphi$, since there are no more observations over Σ than over $\Sigma \cup \Delta$. □

This theorem provides a sound method of proving implementations correct. In general, this method is not complete. This is because the relations $\sim$ and $\perp_m$ over the signature Σ are coarser than the same relations over $\Sigma \cup \Delta$, where more operations exist. Consider, e.g., sets implemented by lists, with repetitions allowed in the representations. Then two lists differing only in the number of repetitions will be considered equivalent over Σ (i.e., with respect to the set operations). But over $\Sigma \cup \Delta$ (i.e., with respect to the list operations) they are not equivalent any more.

A second source of incompleteness is non-termination: the defining sentences in $\mathrm{Sen}(I)$ don't force non-terminating memory operations to actually return $\perp$. To circumvent this problem, we consider, for a set of Δ-formulas Φ, only implementations $I : \Sigma \to \Delta$ that are *total* w.r.t. Φ, that is, implementations such that in $B|_I$ memory operations are total for any algebra B s.t. $B \models_{\Delta} \Phi$.

Save for the above phenomena, the presented proof method is complete. In other words, for total implementations, completeness is guaranteed if all observations in $\Sigma \cup \Delta$ may be conducted in Δ as well:

Theorem 2. *If $I : \Sigma \to \Delta$ is an implementation, Σ contains Δ and I is an identity on Δ, then for any set Φ of Δ-formulas such that I is total w.r.t. Φ, and for any Σ-formula φ we have $\Phi \cup \mathrm{Sen}(I) \models_{\Sigma} \varphi$ iff $\Phi|_I \models_{\Sigma} \varphi$.*

Proof. By the previous theorem and since I is an identity on $\Sigma \cap \Delta = \Delta$, only the "if" direction needs to be shown. Assume $B \models_{\Sigma} \Phi \cup \mathrm{Sen}(I)$ and let B_0 be the restriction of B to Δ. Then $B_0|_I \models_{\Sigma} \Phi$, because $B_0|_I$ and B coincide on Δ and all operations in Σ are defined by I in terms of operations from Δ, so no new observations exist in $B_0|_I$. Thus $B_0|_I \models_{\Sigma} \varphi$. We also have $B_0 \models_{\Delta} \Phi$, since $B \models_{\Sigma} \Phi$ and B_0 is a restriction of B. Since I is total w.r.t. Φ, this implies that in $B_0|_I$ all operations are total. At the same time, B and $B_0|_I$ coincide on Δ and $B \models_{\Sigma} \mathrm{Sen}(I)$. But there can be only one total algebra which coincides on Δ with B and satisfies $\mathrm{Sen}(I)$, and so $B = B_0|_I$. Hence, $B \models_{\Sigma} \varphi$, as required. □

Another way to ensure completeness is to allow various versions of the relations $\sim$ and $\perp_m$ to appear in axioms. If we tag these relations by the appropriate legal sets of observations (in the form of programs), and if by $I(\varphi)$ we denote the formula with appropriate tagging, then we would get the equivalence$\Phi \cup \mathrm{Sen}(I) \models_{\Sigma \cup \Delta} I(\varphi)$ *iff* $\Phi|_I \models_{\Sigma} \varphi$. The obvious downside of using $I(\varphi)$ is that we have to deal with a more complex logic. But consider a logic obtained by allowing the relations $\sim$ and $\perp_m$ to appear as *premises* in implications in the

formulas: in this case the proposed proof method ceases to be sound, because the new observations work both ways, harming both completeness and soundness. If we use the translation $I(\varphi)$, soundness and completeness are preserved.

8 Conclusions

We introduced an algebraic scheme for specifying and proving correct pointer programs using their observable behaviour. The mechanism is based on first-order and behavioural principles, and so could be adopted within existing algebraic specification frameworks. Further investigations are warranted, including the study of a suitable proof system and extensions of the language. One extension would be the aforementioned generalised equivalence and separation relations; another would be a more expressive logic where coinductive predicates such as disjointness are definable directly.

While we aimed at being more abstract than existing work based on concrete memory models, it is clear that we could be more abstract still. Recasting our work in a higher-order setting would allow us to make comparisons with related work in programming language semantics based on monads and coalgebras (e.g. [7,8]), as would studying model-theoretic foundations. It is natural to want to specify parts of our models to be generated inductively and require an initial subalgebra interpretation, while the observational relations have coinductive characterisations related to final coalgebras.

Finally, we would like to revisit our starting point of specifying interaction between resources and their consumption in general. Although our focus was on models of memory in this paper, there is nothing special about "memories" in our approach that forces them to have this interpretation. Getting closer to real machines, we might add stack operations to our memory signature. To describe space usage and consider limited memories, we can add a `size` function on memories together with an *out-of-memory* exception. Imagining a different interpretation entirely, we could conceive of the sort μ to denote a database of statistical data in tables; computations on this data may interfere when data sources are combined but not independent.

Related work. As mentioned at the start, there is much current activity in studying pointer programs, their logics and correctness proofs, as well as more abstract notions of resource. Space precludes a survey; we mention only a few connections. First, program logics designed for managing aliasing (e.g., Reynolds' Separation Logic [1], Honda et al's process algebra inspired approach [2]) aim to simplify proofs of pointer manipulating programs, particularly for better modularity. Our notion of machine generalises the concrete model considered by Separation Logic, but our low-level language of assertions differs, in particular making the global heap explicit. Nonetheless our equivalence notion allows both strong assertions like $m \lesssim m'$ which amount to *global* frame conditions, as well as *local* equivalences such as $(m, a) \sim (m, a')$ which, conceptually, are restricted to reachable heap portions.

Elsewhere, other authors have found equivalence relations like ours useful. For example, Calcagno and O'Hearn [9] pointed out a need for an observational approach in Separation Logic in the presence of garbage collection, because otherwise assertions in the logic can distinguish programs which ought to be considered identical. Benton [10,5] studies correctness proofs for program analyses and transformations using a relational Hoare logic, noting that in general desirable program equivalences are context-sensitive.

In the algebraic domain, perhaps surprisingly, nobody seems to have begun from the same simple definitions as we gave in Sect. 2. But there are certainly a number of rich mechanisms for treating state in dynamic systems, for example, Hidden Algebra [11] and SB-CASL [12], as well as work on notions of behavioural equivalence between algebras and proof mechanisms (see e.g., [13]). We hope that one of our contributions in this work is to open a way to bring together this strand of work in algebraic specification with the recent work in program logics.

Acknowledgements. This work was supported by the British Council; DA was also supported by the EC project MOBIUS (IST-15905), PH by the EC project SENSORIA (IST-16004). We're grateful for feedback from referees and colleagues.

References

1. Reynolds, J.: Separation logic: A logic for shared mutable data structures. In: LICS 2002, pp. 55–74 (2002)
2. Honda, K., Yoshida, N., Berger, M.: An observationally complete program logic for imperative higher-order frame rules. LICS'05, pp. 270–279 (2005)
3. Aspinall, D., Beringer, L., Hofmann, M., Loidl, H.W., Momigliano, A.: A program logic for resources. Theoretical Computer Science (Accepted) (2007)
4. Petersen, L., Harper, R., Crary, K., Pfenning, F.: A type theory for memory allocation and data layout. POPL'03, pp. 172–184 (2003)
5. Benton, N., Kennedy, A., Hofmann, M., Beringer, L.: Reading, writing and relations. In: Kobayashi, N. (ed.) APLAS 2006. LNCS, vol. 4279, pp. 114–130. Springer, Heidelberg (2006)
6. Pym, D., O'Hearn, P., Yang, H.: Possible worlds and resources: The semantics of BI. Theoretical Computer Science 315(1), 257–305 (2004)
7. Jacobs, B., Poll, E.: Coalgebras and monads in the semantics of Java. TCS 291(3), 329–349 (2003)
8. Schröder, L., Mossakowski, T.: Monad-independent dynamic logic in HasCasl. J. Log. Comput. 14(4), 571–619 (2004)
9. Calcagno, C., O'Hearn, P., Bornat, R.: Program logic and equivalence in the presence of garbage collection. TCS 298(3), 557–581 (2003)
10. Benton, N.: Simple relational correctness proofs for static analyses and program transformations. POPL'04, pp. 14–25 (2004)
11. Goguen, J., Malcolm, G.: A hidden agenda. TCS 245(1), 55–101 (2000)
12. Baumeister, H., Zamulin, A.: State-based extension of CASL. In: Grieskamp, W., Santen, T., Stoddart, B. (eds.) IFM 2000. LNCS, vol. 1945, pp. 3–24. Springer, Heidelberg (2000)
13. Hennicker, R., Bidoit, M.: Observational logic. In: Haeberer, A.M. (ed.) AMAST 1998. LNCS, vol. 1548, pp. 263–277. Springer, Heidelberg (1999)

Bisimilarity and Behaviour-Preserving Reconfigurations of Open Petri Nets*

Paolo Baldan[1], Andrea Corradini[2], Hartmut Ehrig[3],
Reiko Heckel[4], and Barbara König[5]

[1] Dipartimento di Matematica Pura e Applicata, Università di Padova, Italy
[2] Dipartimento di Informatica, Università di Pisa, Italy
[3] Institut für Softwaretechnik und Theoretische Informatik,
Technische Universität Berlin, Germany
[4] Department of Computer Science, University of Leicester, UK
[5] Abteilung für Informatik und Angewandte Kognitionswissenschaft,
Universität Duisburg-Essen, Germany

Abstract. We propose a framework for the specification of behaviour-preserving reconfigurations of systems modelled as Petri nets. The framework is based on open nets, a mild generalisation of ordinary Place/Transition nets suited to model open systems which might interact with the surrounding environment and endowed with a colimit-based composition operation. We show that natural notions of (strong and weak) bisimilarity over open nets are congruences with respect to the composition operation. We also provide an up-to technique for facilitating bisimilarity proofs. The theory is used to identify suitable classes of reconfiguration rules (in the double-pushout approach to rewriting) whose application preserves the observational semantics of the net.

1 Introduction

Petri nets are a well-known model of concurrent and distributed systems, widely used both in theoretical and applicative areas [19]. In classical approaches, nets are intended to represent closed, completely specified systems evolving autonomously through the firing of transitions. Therefore, ordinary Petri nets do not support directly certain features that are needed to model *open* systems, namely systems which can interact with the surrounding environment or, in a different view, systems which are only partially specified.

Firstly, a large (possibly still open) system is typically built out of smaller open components. Syntactically, an open system is equipped with suitable interfaces, over which the interaction with the external environment can take place. Semantically, openness can be represented by defining the behaviour of a component as if it were embedded in general environments, determining any possible interaction over the interfaces.

* Research partially supported by the EU IST-2004-16004 SEnSOria, the MIUR Project ART, the DFG project SANDS and CRUI/DAAD Vigoni "Models based on Graph Transformation Systems: Analysis and Verification".

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 126–142, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Secondly, often the building components of an open system are not statically determined, but they can change during the evolution of the system, according to predefined reconfiguration rules triggered by internal or external solicitations.

In this paper we present a framework where open systems can be modelled as Petri nets. Observational semantics based on (weak) bisimulation are shown to be congruences with respect to the composition operation defined over Petri nets. Building on this, suitable reconfigurations of such systems can be specified as net rewritings, which preserve the behaviour of the system.

The framework is based on so-called *open nets*, a mild generalisation of ordinary Petri nets introduced in [2,3] to answer the first of the requirements above, i.e., the possibility of interacting with the environment and of composing a larger net out of smaller open components. An open net is an ordinary net with a distinguished set of places, designated as open, through which the net can interact with the surrounding environment. As a consequence of such interaction, tokens can be freely generated and removed in open places. In the mentioned papers open nets are endowed with a composition operation, characterised as a pushout in the corresponding category, suitable to model both interaction through open places and synchronisation of transitions.

In the first part of the paper, after having extended the existing theory for open nets to deal with *marked* nets, we introduce bisimulation-based observational equivalences for open nets. Following the intuition about reactive systems discussed in [12], such equivalences are based on the observation of the interactions between the given net and the surrounding environment. The framework treats uniformly *strong bisimilarity*, where every transition firing is observed, and *weak bisimilarity*, where a subset of unobservable transition labels is fixed and the firings of transitions carrying such labels are considered invisible. Bisimilarity is shown to be a congruence with respect to the composition operation over open nets. Interestingly enough, this holds also when the set of non-observable labels is not empty, i.e., for weak bisimilarity: some natural questions regarding the relation with weak bisimilarity in CCS are also addressed. In addition, we also define an up-to technique for facilitating bisimulation proofs.

Exploiting the results in the first part of the paper we introduce a framework for open net reconfigurations. The fact that open net components are combined by means of categorical colimits, suggests a setting for specifying net reconfigurations, based on double-pushout (DPO) rewriting [9]. Using the congruence result for bisimilarity we identify classes of transformation rules which ensure that reconfigurations of the system do not affect its observational behaviour.

A concluding section discusses some related work. A full version of the paper, with proofs and additional results, is available as [4].

2 Marked Open Nets

An *open net*, as introduced in [2,3], is an ordinary P/T Petri net with a distinguished set of *open places*, which represent the interface through which the environment can interact with the net. An open place can be an *input place*,

meaning that the environment can put tokens into it, or an *output place*, from which the environment can remove tokens, or both. In this section we introduce the basic notions for open nets as presented in [3], generalising them to nets with initial marking: this will be needed in the treatment of bisimilarity in Section 4.

Given a set X we write $\mathbf{2}^X$ for the powerset of X and $X^\oplus$ for the free commutative monoid over X. Moreover, given a function $h : X \to Y$ we denote by the same symbol $h : \mathbf{2}^X \to \mathbf{2}^Y$ its extension to sets, and by $h^\oplus : X^\oplus \to Y^\oplus$ its monoidal extension. Given a multiset $u \in X^\oplus$, with $u = \bigoplus_{x \in X} u_x \cdot x$, for $x \in X$ we will write $u(x)$ to denote the coefficient u_x. The symbol 0 denotes the empty multiset.

Definition 1 (multiset projection). *Given a function $f : X \to Y$ and a multiset $u \in Y^\oplus$ we denote by $(u \downarrow f)$ the projection of u along f, which is the multiset over X defined as $(u \downarrow f) = \bigoplus_{x \in X} u_{f(x)} \cdot x$.*

For instance, given $f : \{s_0, s_1, s_2\} \to \{s'_1, s'_2, s'_3\}$ such that $f(s_0) = f(s_1) = s'_1$ and $f(s_2) = s'_2$, we have $(2s'_1 \oplus s'_2 \oplus s'_3 \downarrow f) = 2s_0 \oplus 2s_1 \oplus s_2$. We will mainly work with injective functions, for which the projection operation satisfies some expected properties, such as $f^\oplus((u \downarrow f)) \leq u$ and $(f^\oplus((u \downarrow f)) \downarrow f) = (u \downarrow f)$.

We consider nets where transitions are labelled over a fixed set of labels Λ.

Definition 2 (P/T Petri net). *A* P/T Petri net *is a tuple $N = (S, T, \sigma, \tau, \lambda)$ where S is the set of places, T is the set of transitions, $\sigma, \tau : T \to S^\oplus$ are functions mapping each transition to its pre- and post-set and $\lambda : T \to \Lambda$ is a labelling function for transitions.*

In the sequel we will denote by ${}^\bullet(\cdot)$ and $(\cdot)^\bullet$ the monoidal extensions of the functions σ and τ to functions from $T^\oplus$ to $S^\oplus$. Moreover, given $s \in S$, the pre- and post-set of s are defined by ${}^\bullet s = \{t \in T : s \in t^\bullet\}$ and $s^\bullet = \{t \in T : s \in {}^\bullet t\}$.

Definition 3 (Petri net category). *Let N_0 and N_1 be Petri nets. A* Petri net morphism *$f : N_0 \to N_1$ is a pair of total functions $f = \langle f_T, f_S \rangle$ with $f_T : T_0 \to T_1$ and $f_S : S_0 \to S_1$, such that for all $t_0 \in T_0$, ${}^\bullet f_T(t_0) = f_S^\oplus({}^\bullet t_0)$, $f_T(t_0)^\bullet = f_S^\oplus(t_0{}^\bullet)$ and $\lambda_1(f_T(t_0)) = \lambda_0(t_0)$. The category of P/T Petri nets and Petri net morphisms is denoted by* **Net**.

We next introduce the notion of open net. As anticipated above, differently from [2,3], we work here with marked nets.

Definition 4 (open net). *An* open net *is a pair $Z = (N_Z, O_Z)$, where $N_Z = (S_Z, T_Z, \sigma_Z, \tau_Z, \lambda_Z)$ is a P/T Petri net and $O_Z = (O_Z^+, O_Z^-) \in \mathbf{2}^{S_Z} \times \mathbf{2}^{S_Z}$ are the sets of input and output open places of the net. A* marked *open net is a pair $(Z, \hat{u})$ where Z is an open net and $\hat{u} \in S_Z^\oplus$ is the initial marking.*

Hereafter, unless stated otherwise, all open nets will be implicitly assumed to be marked. An open net will be denoted simply by Z and the corresponding initial marking by $\hat{u}$. Subscripts carry over to the net components. The graphical representation for open nets is similar to that for standard nets. In addition, the

fact that a place is input or output open is represented by an ingoing or outgoing dangling arc, respectively. For instance, in net Z_1 of Fig. 1, place s is both input and output open, while s' is only output open.

The notion of enabledness for transitions is the usual one, but, besides the changes produced by the firing of the transitions of the net, we consider also the interaction with the environment which is modelled by events, denoted by $+_s$ and $-_s$, which produce or consume a token in an open place s.

Definition 5 (set of extended events). *Let Z be an open net. The* set of extended events *of Z, denoted by $\bar{T}_Z$ and ranged over by ϵ is defined as*

$$\bar{T}_Z = T_Z \cup \{+_s : s \in O_Z^+\} \cup \{-_s : s \in O_Z^-\}.$$

Defining ${}^\bullet{+_s} = 0$ and ${+_s}^\bullet = s$, and symmetrically, ${}^\bullet{-_s} = s$ and ${-_s}^\bullet = 0$, the notion of pre- and post-set extends to multisets of extended events.

Given a marking $u \in {O_Z^+}^\oplus$, we denote by $+_u$ the multiset $\bigoplus_{s \in S} u(s) \cdot +_s$. Similarly, $-_u = \bigoplus_{s \in S} u(s) \cdot -_s$ for $u \in {O_Z^-}^\oplus$.

Definition 6 (firings and steps). *Let Z be an open net. A* step *in Z consists of the execution of a multiset of (extended) events $A \in \bar{T}_Z^\oplus$, i.e., $u \oplus {}^\bullet A\ [A\rangle\ u \oplus A^\bullet$. A step is called a* firing *when it consists of a single event, i.e., $A = \epsilon \in \bar{T}_Z$.*

A firing can be (i) the execution of a transition $u \oplus {}^\bullet t\ [t\rangle\ u \oplus t^\bullet$, with $u \in S_Z^\oplus$, $t \in T_Z$; (ii) the creation of a token by the environment $u\ [+_s\rangle\ u \oplus s$, with $s \in O_Z^+$, $u \in S_Z^\oplus$; (iii) the deletion of a token by the environment $u \oplus s\ [-_s\rangle\ u$, with $u \in S_Z^\oplus$, $s \in O_Z^-$. A step is the firing of a multiset of transitions and interactions with the environment, of the kind $A \oplus -_w \oplus +_v$ for $A \in T_Z^\oplus, w \in {O_Z^-}^\oplus$ and $v \in {O_Z^+}^\oplus$.

Definition 7 (open net category). *An* open net morphism *$f : Z_1 \to Z_2$ is a Petri net morphism $f : N_{Z_1} \to N_{Z_2}$ such that, if we define $\mathsf{in}(f) = \{s \in S_1 : {}^\bullet f_S(s) - f_T({}^\bullet s) \neq \emptyset\}$ and $\mathsf{out}(f) = \{s \in S_1 : f_S(s)^\bullet - f_T(s^\bullet) \neq \emptyset\}$, then*

1. *(i) $f_S^{-1}(O_2^+) \cup \mathsf{in}(f) \subseteq O_1^+$ and (ii) $f_S^{-1}(O_2^-) \cup \mathsf{out}(f) \subseteq O_1^-$.*
2. *$\hat{u}_1 = (\hat{u}_2 \downarrow f_S)$ (*reflection of initial marking*).*

The morphism f is called an open net embedding *if both f_T and f_S are injective. We will denote by* **ONet** *the category of open nets and open net morphisms.*

Intuitively, an embedding $f : Z_1 \to Z_2$ "inserts" net Z_1 into a larger net Z_2, which might constrain the behaviour of Z_1. Conditions 1.(i) and 1.(ii) first require that open places are reflected and hence that places which are "internal" in Z_1 cannot be promoted to open places in Z_2. Furthermore, they ensure that the context in which Z_1 is inserted can interact with Z_1 only through the open places. In fact, if s is a place of Z_1 and its image $f_S(s)$ is in the post-set of a transition of Z_2 which is not in the image of Z_1, from the perspective of Z_1 the environment can generate tokens in s; in this case $s \in \mathsf{in}(f)$, and thus Condition 1.(i) requires s to be an input place. Condition 1.(ii) is analogous

for output places. Finally, condition 2 requires that the marking of Z_1 is the projection of the marking of Z_2: any place $s_1 \in S_1$ must carry the same number of tokens as its image $f(s_1) \in S_2$, i.e., $\hat{u}_1(s_1) = \hat{u}_2(f(s_1))$ for any $s_1 \in S_1$. All morphisms f_1, f_2, α_1 and α_2 in Fig. 1 are examples of open net embeddings (the mappings on places and transitions are those suggested by the shape and labelling of the nets).

It is worth observing that most of the constructions in the paper will be defined for open net embeddings, hence readers can limit their attention to embeddings if this helps the intuition. Still, on the formal side, working in a larger host category with more general morphisms is essential to obtain a characterisation of the composition operation in terms of pushouts. Specifically, non-injective open net morphisms are needed as mediating morphisms (recall, for example, that the category of sets with injective functions does not have all pushouts).

In the sequel, given an open net morphism $f = \langle f_S, f_T \rangle : Z_1 \to Z_2$, to lighten the notation we will omit the subscripts "S" and "T" in its place and transition components, writing $f(s)$ for $f_S(s)$ and $f(t)$ for $f_T(t)$. Moreover we will write $f^{\oplus} : \bar{T}_{Z_1}^{\oplus} \to \bar{T}_{Z_2}^{\oplus}$ to denote the monoidal function defined on the generators by $f^{\oplus}(t) = f(t)$ for $t \in T_{Z_1}$ and, for $x \in \{+, -\}$, $f^{\oplus}(x_s) = x_{f(s)}$, if $f(s) \in O_2^x$ and $f^{\oplus}(x_s)$ undefined, otherwise. Note that $f^{\oplus}$ can be partial since open places can be mapped to closed places.

Unlike most of the morphisms considered over Petri nets in the literature, open net morphisms are *not* simulations. Instead, since open net embeddings are designed to capture the idea of inserting a net into a larger one, they are expected to reflect the behaviour, in the sense that given an embedding $f : Z_0 \to Z_1$, the behaviour of Z_1 can be projected along f to the behaviour of Z_0.

To formalise reflection of the behaviour along open nets embeddings, we define the projection operation also over steps.

Definition 8 (projecting extended events). *Given an open net embedding $f : Z \to Z'$ and an extended event $\epsilon' \in \bar{T}_{Z'}$ we define the* projection of ϵ' along *f as follows:*

– *if $\epsilon' = t' \in T_{Z'}$ is a transition then*
$$(t' \Downarrow f) = \begin{cases} t & \text{if } t \in T_Z \text{ and } f(t) = t' \\ -_{(\bullet t' \downarrow f)} \oplus +_{(t' \bullet \downarrow f)} & \text{if } t' \notin f(T_Z) \end{cases}$$
– *if $\epsilon' = x_{s'}$, with $x \in \{+, -\}$, then $(x_{s'} \Downarrow f) = x_{(s' \downarrow f)}$.*

The projection operation over multisets of extended events $(_ \Downarrow f) : \bar{T}_{Z'}^{\oplus} \to \bar{T}_{Z}^{\oplus}$, is defined as the monoidal extension of the projection of firings.

In words, if we think of the embedding as an inclusion, given a transition t', the projection $(t' \Downarrow f)$ is the transition itself if t' is in Z. Otherwise, if t' is not in Z but it consumes or produces tokens in places of Z, the projection of t' contains the corresponding extended events, expressing the interactions over open places.

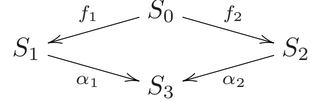
Lemma 1 (reflection of behaviour). *Let $f : Z \to Z'$ be an open net embedding. For every step $u'\ [A'\rangle\ v'$ in Z' there is a step $(u' \downarrow f)\ [(A' \Downarrow f)\rangle\ (v' \downarrow f)$ in Z, called the* projection *of the step $u'\ [A'\rangle\ v'$ over Z.*

3 Composing Open Nets

We introduce next a basic mechanism for composing open nets which is characterised as a pushout construction in the category of open nets. The case of unmarked nets was already discussed in [3]. Here we extend the theory to deal with marked open nets. Intuitively, two open nets Z_1 and Z_2 are composed by specifying a common subnet Z_0, and then by joining the two nets along Z_0.

Let us start with a technical definition which will be useful below.

Proposition 1 (composition of multisets). *Consider a pushout diagram in the category of sets as below, where all morphisms are injective. Given $u_1 \in S_1^{\oplus}$ and $u_2 \in S_2^{\oplus}$ such that $(u_1 \downarrow f_1) = (u_2 \downarrow f_2) = u_0$, there is a (unique) multiset $u_3 \in S_3^{\oplus}$ such that $(u_3 \downarrow \alpha_i) = u_i$, for $i \in \{1, 2\}$. Such a multiset u_3 will be denoted by $u_3 = u_1 \uplus_{u_0} u_2$.*



As in [2,3], two embeddings $f_1 : Z_0 \to Z_1$ and $f_2 : Z_0 \to Z_2$ are called *composable* if the places which are used as interface by f_1, i.e., the places $\mathsf{in}(f_1)$ and $\mathsf{out}(f_1)$, are mapped by f_2 to input and output open places of Z_2, respectively, and also the symmetric condition holds.

Definition 9 (composability). *Let $f_1 : Z_0 \to Z_1$, $f_2 : Z_0 \to Z_2$ be embeddings in* **ONet**. *We say that f_1 and f_2 are* composable *if 1. $f_2(\mathsf{in}(f_1)) \subseteq O^+_{Z_2}$ and $f_2(\mathsf{out}(f_1)) \subseteq O^-_{Z_2}$; and 2. $f_1(\mathsf{in}(f_2)) \subseteq O^+_{Z_1}$ and $f_1(\mathsf{out}(f_2)) \subseteq O^-_{Z_1}$.*

Composability is necessary and sufficient to ensure that the pushout of f_1 and f_2 can be computed in **Net** and then lifted to **ONet**.

Proposition 2 (pushouts in ONet). *Let $f_1 : Z_0 \to Z_1$, $f_2 : Z_0 \to Z_2$ be embeddings in* **ONet** *(see Fig. 2(a)). Compute the pushout of the corresponding diagram in category* **Net** *(componentwise on places and transitions) obtaining net N_{Z_3} and morphisms α_1 and α_2, and then take as open places, for $x \in \{+, -\}$,*

$$O^x_{Z_3} = \{s_3 \in S_3 : \alpha_1^{-1}(s_3) \subseteq O^x_{Z_1} \ \wedge \ \alpha_2^{-1}(s_3) \subseteq O^x_{Z_2}\}$$

and as marking $\hat{u}_3 = \hat{u}_1 \uplus_{\hat{u}_0} \hat{u}_2$, defined according to Proposition 1. Then $(\alpha_1, Z_3, \alpha_2)$ is the pushout in **ONet** *of f_1 and f_2 if and only if f_1 and f_2 are composable. In this case we write $Z_3 = Z_1 +_{f_1,f_2} Z_2$.*

As an example, the open net embeddings f_1 and f_2 in Fig. 1 are composable and Z_3 is the resulting pushout object.

We next analyse the behaviour of an open net Z_3 arising as the composition of two nets Z_1 and Z_2 along an interface Z_0. More specifically, we show that steps of the component nets Z_1 and Z_2 can be "composed" to give a step of Z_3 when they agree on the interface and satisfy suitable compatibility conditions.

Lemma 2 (composing steps). *Let $f_1 : Z_0 \to Z_1$ and $f_2 : Z_0 \to Z_2$ be composable embeddings in* **ONet** *and let $Z_3 = Z_1 +_{f_1,f_2} Z_2$ (see Fig. 2(a)). Let $u_1\ [A_1\rangle\ v_1$ and $u_2\ [A_2\rangle\ v_2$ be steps in Z_1 and Z_2, respectively, such that $(u_1 \downarrow f_1) = (u_2 \downarrow f_2) = u_0$ and $A_2 = f_2^{\oplus}((A_1 \Downarrow f_1))$.*

Then, $(v_1 \downarrow f_1) = v_0 = (v_2 \downarrow f_2)$ and, if we define $A_3 = \alpha_1^{\oplus}(A_1)$,

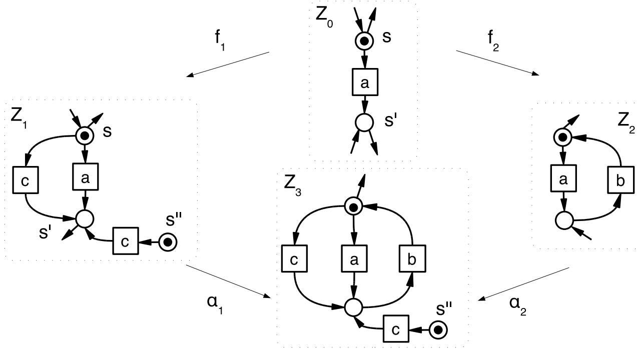


Fig. 1. An example of a pushout in **ONet**

$$u_1 \uplus_{u_0} u_2 \; [A_3\rangle \; v_1 \uplus_{v_0} v_2.$$

The above result can be used to get a compositionality result for steps, showing that the steps of Z_3 can be obtained by "composing" steps of the components Z_1 and Z_2 satisfying suitable compatibility requirements. However, this is outside the main focus of the paper and can be found in the full version [4].

4 Bisimilarity of Open Nets

We next study (strong and weak) bisimilarity for open nets, proving that it is a congruence with respect to the colimit-based composition of open nets.

First, we define the labelled transition system associated to an open net. Net transitions carry a label which is observed when they fire. Additionally, in the labelled transition system we also observe what happens at the open places. As discussed in the conclusions, this resembles the labelled transition system arising from the view of Petri nets as reactive systems in [14,20]. More precisely, given an open net Z, the corresponding labelled transition system has the markings of the net as states. Transitions are generated by the firings of Z and labelled over the set $\Lambda_Z = \Lambda \cup \{+_s : s \in O_Z^+\} \cup \{-_s : s \in O_Z^-\}$.

For notational convenience we extend the labelling function λ_Z to the set of extended events $\bar{T}_Z$, by defining $\lambda_Z(x) = x$ for $x \in \bar{T}_Z - T_Z$ (i.e., for $x = +_s$ or $x = -_s$ with $s \in S_Z$).

Definition 10 (lts for an open net). *The* labelled transition system *associated to an open net Z, denoted by $\mathsf{lts}(Z)$, is the pair $\langle S_Z^\oplus, \rightarrow_Z\rangle$, where states are markings $u_Z \in S_Z^\oplus$ and the transition relation $\rightarrow_Z \subseteq S_Z^\oplus \times \Lambda_Z \times S_Z^\oplus$ includes all transitions $u_Z \stackrel{\lambda_Z(x)}{\longrightarrow}_Z u'_Z$ such that there is a firing $u_Z \; [x\rangle \; u'_Z$ in Z.*

When observing the behaviour of a system, usually only a subset of events is considered observable. Here this is formalised by selecting a subset of labels representing internal firings, playing a role similar to τ-actions in process calculi,

and then considering a corresponding notion of weak bisimilarity. Let $\Lambda_\tau \subseteq \Lambda$ be a subset of *unobservable* labels, fixed for the rest of the paper. Given a Λ-labelled open net Z, for markings $v, v' \in S_Z^{\oplus}$ we write $v \overset{\tau}{\rightsquigarrow}_Z v'$ if $v \xrightarrow{\ell}_Z v'$ with $\ell \in \Lambda_\tau$, and $v \overset{\ell}{\rightsquigarrow}_Z v'$ if $v \xrightarrow{\ell}_Z v'$ with $\ell \in \Lambda_Z - \Lambda_\tau$. Then we define

- $v \overset{\tau}{\Longrightarrow}_Z v'$ when $v \overset{\tau}{\rightsquigarrow}{}^*_Z v'$.
- $v \overset{\ell}{\Longrightarrow}_Z v'$ when $v \overset{\tau}{\rightsquigarrow}{}^*_Z \overset{\ell}{\rightsquigarrow}_Z \overset{\tau}{\rightsquigarrow}{}^*_Z v' \qquad \ell \neq \tau$.

Weak bisimilarity is now defined in a standard way (but note that when the set of unobservable labels is empty, this actually corresponds to strong bisimilarity). Only, we need to specify for each open place of one net which is the corresponding open place in the other net. Given two open nets Z_1 and Z_2 a *correspondence* $\eta : O_1 \leftrightarrow O_2$ between Z_1 and Z_2 is a bijection $\eta : O_1^+ \cup O_1^- \to O_2^+ \cup O_2^-$ such that for $s_1 \in O_1$, $x \in \{+, -\}$ we have $s_1 \in O_1^x$ iff $\eta(s_1) \in O_2^x$.

Definition 11 ((weak) bisimilarity). *Let Z_1, Z_2 be open nets and $\eta : O_1 \leftrightarrow O_2$ be a correspondence between Z_1 and Z_2. A (weak) η-bisimulation over Z_1 and Z_2 is a relation over markings $\mathcal{R} \subseteq S_1^{\oplus} \times S_2^{\oplus}$ such that if $(u_1, u_2) \in \mathcal{R}$ then*

- *if $u_1 \overset{\ell}{\rightsquigarrow}_{Z_1} u_1'$, there exists u_2' such that $u_2 \overset{\eta(\ell)}{\Longrightarrow}_{Z_2} u_2'$ and $(u_1', u_2') \in \mathcal{R}$;*
- *the symmetric condition holds;*

where $\eta(+_s) = +_{\eta(s)}$, $\eta(-_s) = -_{\eta(s)}$, and $\eta(\ell) = \ell$ for any $\ell \in \Lambda \cup \{\tau\}$.

Two open nets Z_1 and Z_2 are (weakly) η-bisimilar, *denoted $Z_1 \approx_\eta Z_2$, if $\eta : O_1 \leftrightarrow O_2$ is a correspondence and there exists a (weak) η-bisimulation $\mathcal{R}$ over Z_1 and Z_2 such that $(\hat{u}_1, \hat{u}_2) \in \mathcal{R}$. We will say that Z_1 and Z_2 are (weakly) bisimilar, written $Z_1 \approx Z_2$, if $Z_1 \approx_\eta Z_2$ for some correspondence η.*

According to the following lemma, which is a corollary of Lemma 2, given *composable* embeddings $f_1 : Z_0 \to Z_1$ and $f_2 : Z_0 \to Z_2$, the firing of a transition in Z_2, projected along f_2 to Z_0 can then be simulated in Z_1.

Lemma 3. *Let Z_0, Z_1, Z_2 be open nets and let $f_i : Z_0 \to Z_i$ ($i \in \{1, 2\}$) be composable embeddings, as in Fig. 2(a). Furthermore, let $Z_3 = Z_1 +_{f_1, f_2} Z_2$. Assume that $u_2 \xrightarrow{\ell}_{Z_2} u_2'$ where $\ell \in \Lambda$, let $t \in T_2$ such that $\lambda_2(t) = \ell$ and $u_2\ [t\rangle\ u_2'$, let $u_0\ [A_0\rangle\ u_0'$ be its projection over Z_0 (hence $A_0 = (t \Downarrow f_2)$), and let $u_0 \xrightarrow{\ell_1}_{Z_0} \ldots \xrightarrow{\ell_n}_{Z_0} u_0'$ be any sequence of transitions in $\mathsf{lts}(Z_0)$ arising as a linearisation of such step in Z_0. Then for any $u_1 \in S_1^{\oplus}$ such that $(u_1 \downarrow f_1) = u_0$ we have that $u_1 \xrightarrow{\ell_1}_{Z_1} \ldots \xrightarrow{\ell_n}_{Z_1} u_1'$ and $u_1 \uplus_{u_0} u_2 \xrightarrow{\ell}_{Z_3} u_1' \uplus_{u_0'} u_2'$.*

Note that above, if transition t is in the image of Z_0, then the sequences of transitions in $\mathsf{lts}(Z_0)$ and $\mathsf{lts}(Z_1)$ are actually single firings. Otherwise, they are sequences of interactions over open places, possibly of length greater than one.

By exploiting this lemma we can prove that bisimilarity is a congruence with respect to the composition operation on open nets.

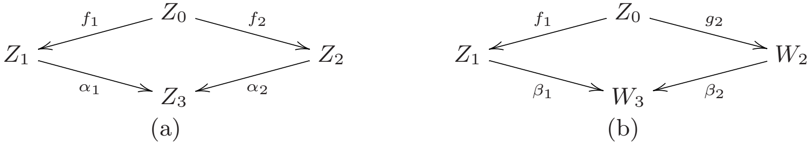


Fig. 2. Pushouts in **ONet**

Theorem 1 (bisimilarity is a congruence). *Let Z_0, Z_1, Z_2, W_2 be open nets. Let $Z_2 \approx_\eta W_2$, for some η. Consider the nets $Z_3 = Z_1 +_{f_1,f_2} Z_2$ and $W_3 = Z_1 +_{f_1,g_2} W_2$, as in Fig. 2 where f_1, f_2 and g_2 are embeddings, f_1 and f_2 are composable, and f_1 and g_2 are composable as well.*

If $g_2|_{O_0} = \eta \circ (f_2|_{O_0})$ (i.e., f_2 and g_2 are consistent with η on open places) then $Z_3 \approx_{\eta'} W_3$, where η' is defined as follows: for all $s \in O_{Z_3}$, $\eta'(s) = \beta_1(s')$ if $s = \alpha_1(s')$, and $\eta'(s) = \beta_2(\eta(s'))$ if $s = \alpha_2(s')$.

We next provide a kind of *up-to technique* for open net bisimilarity. Given an open net Z, let us define the *out-degree* of a place $s \in S$ as the maximum number of tokens that the firing of an extended event can remove from s, formally:

$$deg(s) = \max\left(\{({}^\bullet t)(s) : t \in T_Z\} \cup \{1 : s \in O_Z^-\}\right)$$

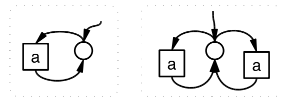
The idea, formalised in the notion of up-to bisimulation, is to allow tokens to be removed from open input places, when they exceed the out-degree of the place. More precisely, given a net Z and a marking $u \in S^\oplus$, let us say that a marking $v \in O_Z^{+\oplus}$ is *subtractable* from u if $\forall s \in O_Z^+.\ deg(s) \leq u(s) - v(s)$. Note that this implies that all transitions enabled in u are also enabled in $u \ominus v$.

Definition 12 (up-to bisimulation). *Let Z_1 and Z_2 be open nets, and let $\eta : O_1 \leftrightarrow O_2$ be a correspondence between Z_1 and Z_2. A relation $\mathcal{R} \subseteq S_1^\oplus \times S_2^\oplus$ between markings is called an* up-to η-bisimulation *if whenever $(u_1, u_2) \in \mathcal{R}$ then*

- *if $u_1 \overset{\ell}{\leadsto}_{Z_1} u_1'$, then there exist markings u_2' such that $u_2 \overset{\eta(\ell)}{\Longrightarrow}_{Z_2} u_2'$, and $v_1 \in O_1^{+\oplus}$ subtractable from u_1, with $(u_1' \ominus v_1, u_2' \ominus \eta^\oplus(v_1)) \in \mathcal{R}$;*
- *the symmetric condition holds.*

Proposition 3. *Let Z_1 and Z_2 be open nets, and let $\eta : O_1 \leftrightarrow O_2$ be a correspondence between Z_1 and Z_2. Let $\mathcal{R}$ be an up-to η-bisimulation. Then for any $(u_1, u_2) \in \mathcal{R}$ we have that $(Z_1, u_1) \approx_\eta (Z_2, u_2)$.*

As it often happens with up-to techniques, the above result might allow to show that two nets are bisimilar by exhibiting finite relations (while bisimulations are typically infinite). E.g., consider the open nets on the right, where label a is observable. Then a bisimulation would include at least the pairs $\{(k \cdot s, k \cdot s) : k \in \mathbb{N}\}$, where s is the only place. Instead, according to the definition above $\{(0, 0), (s, s)\}$ is an up-to bisimulation.



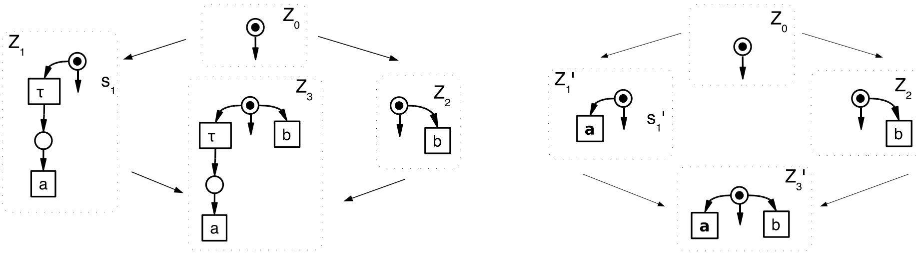


Fig. 3. Two pushouts of open nets for the comparison to CCS

Comparison to CCS. We now give some hints as to why weak bisimilarity is a congruence in the case of open nets, but not in CCS [16]. Remember that a classical counterexample for CCS is as follows: $p_1 = \tau.a.0 \approx a.0 = p_2$, but $q_1 = \tau.a.0 + b.0 \not\approx a.0 + b.0 = q_2$. The reason for the latter inequality is that q_1 can do a τ and become $a.0$, while q_2 cannot mimic this step.

Fig. 3 shows a similar situation of nondeterministic choice for open nets, where τ is the only unobservable label. However, note that here the two nets Z_1 (corresponding to $\tau.a.0$) and Z_1' (corresponding to $a.0$) are *not* weakly bisimilar. Whenever the τ-transition is fired in Z_1, resulting in the marking m_1, this can not be mimicked in Z_1' by staying idle, since then in Z_1' a transition with label $-_{s_1'}$ is possible, while a transition labelled $-_{s_1}$ is not possible for the net Z_1 with marking m_1. Also note that the places s_1 respectively s_1' must be output open in order to allow composition with the net Z_2.

Roughly, this means that for open nets we are always able to observe the first invisible action in an open component, which is reminiscent of the definition of observation congruence (denoted by $\approx^c$) in CCS: two processes p, q are called observation congruent if they are weakly bisimilar, with the additional constraint that whenever the first step of p is a τ-action, then it has to be answered by at least one τ-action of q (and vice versa). In both settings it is only the first τ-action that can be observed but not the subsequent ones.

5 Reconfigurations of Open Nets

The results in the previous sections are used here to design a framework where a system specified as a (possibly open) Petri net can be reconfigured dynamically by transformation rules, triggered by the state/shape of the system. The congruence result allows to characterise classes of reconfigurations which preserve the observational behaviour of the system.

The fact that the composition operation over open nets is defined in terms of a pushout construction suggests naturally a way of reconfiguring open nets by using the double-pushout approach to rewriting [9].

A *rewriting rule* over open nets consists of a pair of morphisms in **ONet**:

$$p = L_p \stackrel{l_p}{\leftarrow} K_p \stackrel{r_p}{\rightarrow} R_p$$

where L_p, K_p, R_p are open nets, called *left-hand side*, *interface* and *right-hand side* of the rule p, and l_p, r_p are open net embeddings. Furthermore, it is required that $(r_p \circ l_p^{-1})|_{O_{L_p}}$ is a correspondence between L_p and R_p, which we denote by $\eta_p : L_p \leftrightarrow R_p$. Intuitively, the rule specifies that, given a net Z, if the left-hand side L_p matches a subnet of Z then this can be reconfigured into Z' by replacing the occurrence of L_p with the right-hand side R_p, preserving the subnet K_p. Note that by requiring the existence of the correspondence η_p, we guarantee that the interface of the transformed net, consisting of the open places, is left untouched by the reconfiguration (a more general treatment can be found in [4]). A rewriting rule p is called *behaviour preserving* if its left- and right-hand sides are bisimilar: more precisely, if $L_p \approx_{\eta_p} R_p$.

Definition 13 (open net transformation). *Let p be a rewriting rule over open nets, let Z be an open net and let $m : L_p \to Z$ be a match, i.e., an open net embedding. We say that Z rewrites to Z' using p at match m, writing $Z \Rightarrow^{p,m} Z'$ or simply $Z \Rightarrow^{p} Z'$, if the diagram of Fig. 4(a) can be constructed in* **ONet***, where both squares are pushouts, and morphism n is composable with both l_p and r_p.*

We stress that we are interested in transformations where the two pushout squares are built from composable arrows (technically, this ensures that the transformation can be performed in **Net** and then "lifted" to **ONet**).

The next result is now an easy consequence of Theorem 1.

Theorem 2 (behaviour-preserving reconfigurations). *Let p be a behaviour-preserving open net rule. Given an open net Z and a match $m : L_p \to Z$, if $Z \Rightarrow^{p,m} Z'$ then $Z \approx Z'$.*

For instance, consider the double-pushout diagram in Fig. 4(b). It can be easily seen that the left- and right-hand sides of the applied rule are strongly bisimilar. Hence we can conclude that also Z and Z' are strongly bisimilar.

5.1 Applying Rules to Open Nets

As it is common in the categorical approaches to (graph) rewriting, the notion of open net transformation proposed in Definition 13 is rather "declarative" in style, because it requires the existence of two pushouts in category **ONet**, without stating how they can be constructed, and under which conditions. A more explicit description of the conditions under which a rule can be applied to an open net and of the way the resulting net can be constructed, is clearly necessary for practical purposes. Looking at Fig. 4(a), given a rule p and a match $m : L_p \to Z$, in order to build the open net transformation:

– The *pushout complement* of l_p and m must exist. The resulting arrows n and d must be such that l_p and n are composable. Additionally, there can be several pushout complements and in this case a canonical choice should be considered.

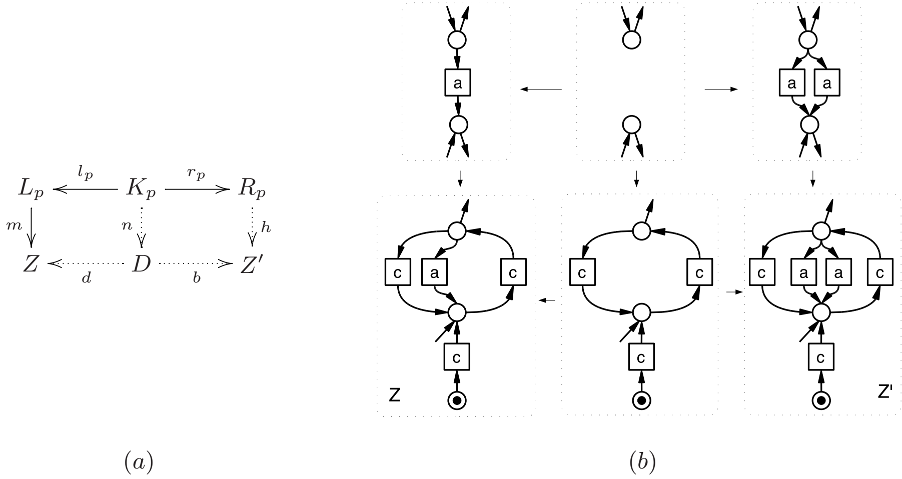


Fig. 4. Transforming open nets through DPO rewriting

– The resulting arrow n must be composable with r_p: then we know how to build Z' by Proposition 2.

Unfortunately, although a general theory of DPO rewriting has been developed recently in the framework of adhesive categories [11], we cannot exploit it here since the category of open nets falls outside the scope of the theory. Sufficient hypotheses under which the above conditions are satisfied are made explicit in the following lemma (more general conditions are considered in [4]).

Lemma 4 (existence of transformations in ONet). *Let p be an open net rewriting rule, let Z be an open net and let $m : L_p \to Z$ be a match such that:*

1. *for all $s \in L_p - l_p(K_p)$ we have $^\bullet m(s) \cup m(s)^\bullet \subseteq m(L_p - K_p)$;*
2. *for all $s \in K_p$, if $s \in \mathsf{in}(r_p) - \mathsf{in}(l_p)$ then $m(l_p(s)) \in O_Z^+$;*
3. *for all $s \in K_p$, if $s \in \mathsf{in}(l_p)$ then $l_p(s) \in O_L^+$ implies $m(l_p(s)) \in O_Z^+$;*

and the dual of the last two conditions, obtained by replacing $\mathsf{in}()$ by $\mathsf{out}()$ and $+$ by $-$, hold. Then, there exists a transformation $Z \Rightarrow^{p,m} Z'$.

The intuition underlying the conditions above is the following. Condition 1 is a typical *dangling condition*: it asserts that a place can be deleted only if all the transitions connected to this place are removed as well, otherwise the flow arcs of this transition would remain dangling. Technically, this condition ensures that the pushout complement exists and is unique in the underlying category **Net**. By condition 2, if $s \in \mathsf{in}(r_p) - \mathsf{in}(l_p)$, i.e., the rule p creates a new (ingoing) transition connected to place s, without replacing any old one, then the image of s in Z must be (input) open. Finally, condition 3 says that if $s \in \mathsf{in}(l_p)$, i.e., if some (ingoing) transitions are deleted from s then the image of s in Z must be (input) open if so is its image in L_p.

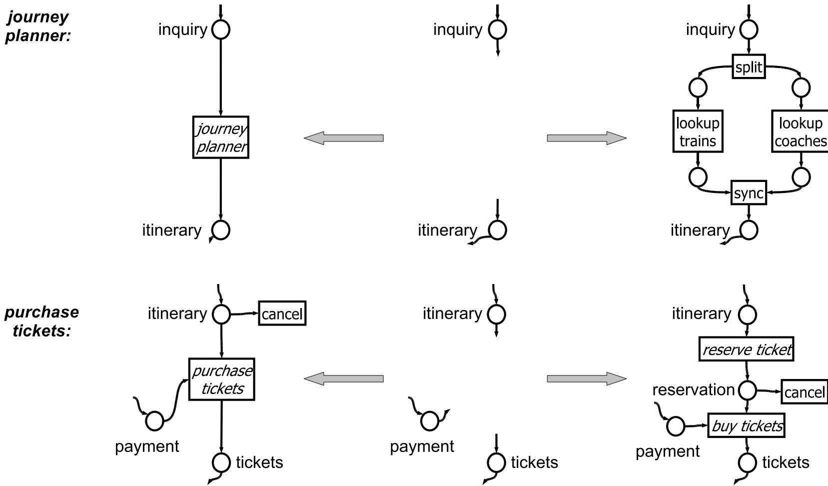


Fig. 5. Rules

Technically, conditions 2 and 3 (and their dual) ensure the existence of a *minimal* pushout complement D, i.e., a pushout complement which embeds into any other, which is the one that we choose to define the transformation; the conditions also guarantee the composability of n with both l_p and r_p. The net underlying the minimal pushout complement is $D = Z - m(L_p - l_p(K_p))$ (with set difference componentwise on places and transitions), and the open places of D are given by $O^x_D = d^{-1}(O^x_Z)$ for $x \in \{+, -\}$. The initial marking $\hat{u}_D$ is defined as $\hat{u}_D(s) = \hat{u}_Z(d(s))$ for any place $s \in S_D$.

As an example, consider again the DPO diagram in Fig. 4(b). It is not difficult to see that the rule and the match satisfy the conditions of Lemma 4. Hence we can complete the double-pushout construction transforming Z into Z', as depicted in the same figure.

5.2 Modeling Dynamic Reconfigurations of Services

Open nets allow us to specify a system as built out of smaller components. Then, its behaviour is captured by the firing behaviour of the open net. However, for highly dynamic systems, as mentioned in the introduction, it can be useful to have the possibility of specifying that, under suitable conditions, some structural changes or reconfigurations of the system can take place. For instance the invocation of a service could trigger a rule which provides an implementation of the required service.

The theory of open net reconfigurations can do the job. As an example, consider net N_0 in Fig. 6 which models the view of a traveller on the journey planning and ticket purchase services offered through a travel agency portal.

We distinguish *abstract transitions* representing services that should be provided elsewhere and *concrete transitions* representing local services and control

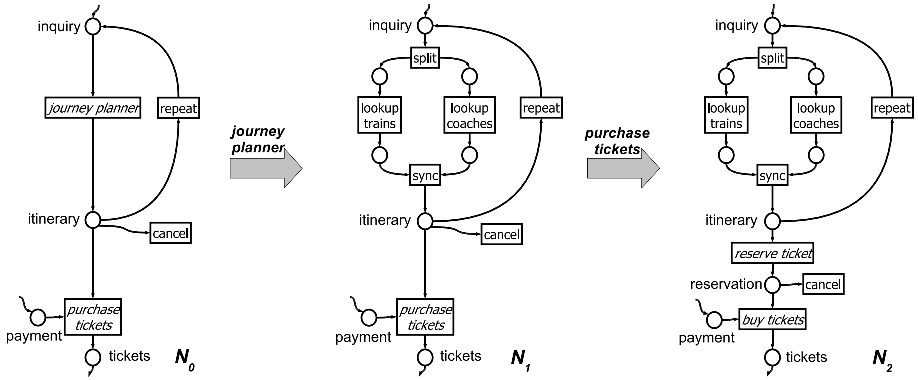


Fig. 6. Transformation of open nets representing a travel agent's portal

flow actions. The invocation of an external service can be seen at different levels of abstraction. From the point of view of the client process it is just the firing an abstract transition. At a lower level of abstraction, it is captured by a rule such as the one at the top of Fig. 5. An application of this rule, replacing the abstract transition by a new open net, models the discovery and binding of the concrete services required. The left- and right-hand sides of the rule are weakly bisimilar if we observe only the interactions at the open (interface) places, i.e., if we take $\Lambda_\tau = \Lambda$. This can be seen as a proof of the fact that the bound service meets the requirements: both in the abstract transition and in its concrete counterpart any inquiry will produce a corresponding itinerary.

The rule in the bottom of Fig. 5 represents a case where a simple pattern is replaced by a richer one. On the left we say that, given an itinerary, we can either purchase the required tickets or cancel the processes. On the right the transaction is refined, adding a prior reservation phase, while keeping the option to cancel. As above, the rule has weakly bisimilar left- and right-hand sides, ensuring that the visible effect of the abstract and concrete transitions at the interfaces is the same.

A possible sequence of transformations is shown in Fig. 6. By the above considerations, we are sure that the transformations do not change the observable behaviour of the system, a fact that can be interpreted as a proof of conformance of the provided service with respect to the abstract specification.

6 Conclusion and Related Work

Open nets, introduced in [2,3], are a reactive extension of standard Petri nets which allows to model systems interacting with an unspecified environment. Several other approaches to Petri net composition and reactivity have been proposed

in the literature (see, e.g., [6,17,10], to mention a few) and a detailed comparison with the open net model can be found in [3].

In this paper, firstly we have generalised the theory of open nets, including the characterisation of net composition using pushouts, to the case of marked nets. Next we have introduced the notions of strong and weak bisimilarity over open nets. Weak bisimilarity (and, as a particular case, also strong bisimilarity) is shown to be a congruence with respect to the colimit-based composition operation over open nets. To the best of our knowledge, this is the first time that a compositionality result is given for weak bisimilarity over Petri nets. Weak bisimilarity for Petri nets with a composition operation is studied for instance in [17], but it is not congruence, though a context closure allows one to get a congruence which is then characterised by means of a universal context. Our result about strong bisimilarity can be seen as a generalisation of those in [15,20], which essentially are developed for a special kind of open nets, arising by viewing them as bigraphical reactive systems or as reactive systems over a cospan category. In the resulting reactive Petri net model there is no distinction between open input and output places. Furthermore the composition operation used in these papers does not allow synchronisation of transitions. Similarities exist also with the problem studied in [7], where a reactive Petri net model which admits a compositional behavioural equivalence is exploited, in the framework of web-services, to provide a theoretical basis to service composition and discovery.

In the second part of the paper we have proposed a rewriting-based framework for Petri nets with reconfigurations. We have shown how our congruence results for the observational semantics can be used to identify classes of reconfigurations which do not alter the observational behaviour of the system. This is applied to a small case study of a workflow-like model of a travel agency.

The idea of using rewriting techniques for providing a reconfiguration mechanism for Petri nets has been already explored in the literature (see, e.g., reconfigurable nets of [1,13] and high-level replacement systems applied to Petri nets in [18]). In future work, besides analysing the relationships between these approaches and ours, we will continue to study the notion of reconfigurable open nets and describe in more detail how reconfigurations can be triggered by the net itself, for example by reaching certain markings or by firing certain transitions, following an intuition similar to that of dynamic nets [8].

Finally, it would be worth studying whether a formal duality can be established between our morphisms and standard simulation morphisms for Petri nets. Viewing our morphisms as inverses of (partial) simulation morphisms would allow to get a precise correspondence between our pushout-based composition and pullback-based synchronisation of Petri nets. Surely by simply taking Winskel's morphisms [22] this does not work (technically because when they are undefined on a transition they must be undefined on the corresponding pre- and post-set). However a duality result could be possibly obtained by considering suitable extensions of Winkel's morphisms, like those in [21,5].

References

1. Badouel, E., Llorens, M., Oliver, J.: Modeling concurrent systems: Reconfigurable nets. In: Proc. of PDPTA'03, vol. 4, pp. 1568–1574. CSREA Press (2003)
2. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional modeling of reactive systems using open nets. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 502–518. Springer, Heidelberg (2001)
3. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional semantics for open Petri nets based on deterministic processes. Mathematical Structures in Computer Science 15(1), 1–35 (2004)
4. Baldan, P., Corradini, A., Ehrig, H., Heckel, R., König, B.: Bisimilarity and behaviour-preserving reconfigurations of open Petri nets. Technical Report CS-2006-9, Computer Science Department, University Ca' Foscari of Venice (2006)
5. Bednarczyk, A.M., Borzyszkowski, M.A.: General morphisms of Petri nets. In: Wiedermann, J., van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 190–199. Springer, Heidelberg (1999)
6. Best, E., Devillers, R., Hall, J.G.: The Petri box calculus: a new causal algebra with multi-label communication. In: Rozenberg, G. (ed.) Advances in Petri Nets 1992. LNCS, vol. 609, pp. 21–69. Springer, Heidelberg (1992)
7. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: A behavioural congruence for web services. In: Proc. of FSEN'07. LNCS, Springer, Heidelberg (2007) (to appear)
8. Buscemi, M.G., Sassone, V.: High-level Petri nets as type theories in the join calculus. In: Honsell, F., Miculan, M. (eds.) ETAPS 2001 and FOSSACS 2001. LNCS, vol. 2030, Springer, Heidelberg (2001)
9. Ehrig, H.: Tutorial introduction to the algebraic approach of graph-grammars. In: Ehrig, H., Nagl, M., Rosenfeld, A., Rozenberg, G. (eds.) Graph-Grammars and Their Application to Computer Science. LNCS, vol. 291, pp. 3–14. Springer, Heidelberg (1987)
10. Kindler, E.: A compositional partial order semantics for Petri net components. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 235–252. Springer, Heidelberg (1997)
11. Lack, S., Sobociński, P.: Adhesive and quasiadhesive categories. RAIRO – Theoretical Informatics and Applications 39(3) (2005)
12. Leifer, J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 243–258. Springer, Heidelberg (2000)
13. Llorens, M., Oliver, J.: Introducing structural dynamic changes in Petri nets: Marked-controlled reconfigurable nets. In: Wang, F. (ed.) ATVA 2004. LNCS, vol. 3299, pp. 310–323. Springer, Heidelberg (2004)
14. Milner, R.: Bigraphical reactive systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 16–35. Springer, Heidelberg (2001)
15. Milner, R.: Bigraphs for Petri nets. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 686–701. Springer, Heidelberg (2004)
16. Milner, R.: In: Milner, R. (ed.) A Calculus of Communication Systems. LNCS, vol. 92, Springer, Heidelberg (1980)
17. Nielsen, M., Priese, L., Sassone, V.: Characterizing Behavioural Congruences for Petri Nets. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 175–189. Springer, Heidelberg (1995)

18. Padberg, J., Ehrig, H., Ribeiro, L.: High level replacement systems applied to algebraic high level net transformation systems. Mathematical Structures in Computer Science 5(2), 217–256 (1995)
19. Reisig, W.: Petri Nets: An Introduction. In: EATCS Monographs, Springer, Heidelberg (1985)
20. Sassone, V., Sobociński, P.: A congruence for Petri nets. In: Alternating Sequential-Parallel Processing. Electronic Notes in Computer Science, vol. 127(2), pp. 107–120. Elsevier, Amsterdam (1982)
21. Vogler, W.: Executions: A new partial-order semantics of Petri nets. Theoretical Computer Science 91(2), 205–238 (1991)
22. Winskel, G.: Event structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Advances in Petri Nets 1986. Proceedings of an Advanced Course, Bad Honnef. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987)

Free Modal Algebras: A Coalgebraic Perspective

N. Bezhanishvili* and A. Kurz**

Department of Computer Science, University of Leicester, United Kingdom

Abstract. In this paper we discuss a uniform method for constructing free modal and distributive modal algebras. This method draws on works by (Abramsky 2005) and (Ghilardi 1995). We revisit the theory of normal forms for modal logic and derive a normal form representation for positive modal logic. We also show that every finitely generated free modal and distributive modal algebra axiomatised by equations of rank 1 is a reduct of a temporal algebra.

1 Introduction

Modal logics play an important role in many areas of computer science. In recent years, the connection of modal logic and coalgebra received a lot of attention, see eg [30]. In particular, it has been recognised that modal logic is to coalgebras what equational logic is to algebras. The precise relationship between the logics and the coalgebras can be formulated using Stone duality [9]. From this perspective, algebras are the logical forms of coalgebras [1]; and the algebras that appear in this way give rise to modal logics.

In this paper we take the opposite view and ask how coalgebraic and categorical methods can elucidate traditional topics in modal logic. Algebraic methods and techniques proved to be very useful in investigations of modal logics, see eg [8,30]. Here we apply a mix of algebraic and coalgebraic (and categorical) techniques to shed some light on the construction of canonical models of modal logics. In principle, almost all properties of a given modal logic are enshrined in its free modal algebras or, dually and equivalently, in its canonical models [8]. Therefore, an understanding of the structure of the canonical model of a given modal logic can be the key for understanding the properties of this logic.

The general idea that we will discuss in this paper has appeared before in different contexts. Fine [16] used his canonical formulas for describing canonical models of modal logics and for deriving completeness results for these logics. Moss [25] revisited Fine's formulas to give a filtration type finite-model property proofs for various modal logics. Abramsky [2] constructed the canonical model of closed formulas of the basic modal logic as the final coalgebra for the Vietoris functor and Ghilardi [18,17] gave a similar description of canonical models of modal and intuitionistic logics to derive a normal form representation for these logics. For positive modal logic similar techniques were developed by Davey and Goldberg [13].

* Supported by the EPSRC grant EP/C014014/1 and by the Georgian National Science Foundation grant GNSF/ST06/3-003.

** Partially supported by EPSRC EP/C014014/1.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 143–157, 2007.
© Springer-Verlag Berlin Heidelberg 2007

The aim of this paper is to unify all these approaches and present a coherent method for constructing free modal and distributive modal algebras. Modal algebras are algebraic models of (classical) modal logic and distributive modal algebras are algebraic models of positive (negation-free) modal logic. We will show how to construct free algebras for a variety **V** equipped with an operator f. In case of modal algebras **V** is the variety of Boolean algebras and in case of distributive modal algebras **V** is the variety of distributive lattices. The main idea of the construction is the following: We start with the free **V**-algebra and step by step add freely to it the operator f. As a result we obtain a countable sequence of algebras whose direct limit is the desired free algebra.

We apply this general method to modal and distributive modal algebras. For distributive modal algebras these results appear to be new. In case of modal algebras this approach gives simple and coherent proofs of known results. We use the Stone duality for Boolean algebras and the Priestley duality for distributive lattices to describe the dual spaces of the finite approximants of the free algebras. The key for dualising these constructions lies in the coalgebraic representation of modal spaces as coalgebras for the Vietoris functor [22] and in the coalgebraic representation of modal Priestley spaces as coalgebras for the convex set functor [20,27]. This allows us to represent the canonical models of modal and positive modal logic as a limit of finite sets and posets, respectively. We also observe that the underlying Stone space of the canonical model of the basic modal logic is homeomorphic to the so-called Pelczynski space. This space appears to be one of the nine fixed points of the Vietoris functor on compact Hausdorff spaces with a countable basis [28,26].

As we will see below, this method directly applies to modal and positive modal logics that are axiomatised by the formulas of rank 1. We also indicate how to adjust our techniques to modal logics that are not axiomatised by formulas of rank 1. As an example we consider the ‘reflexive’ modal logic, that is, the modal logic axiomatised by the additional reflexivity axiom $\varphi \to \Diamond\varphi$, which is not of rank 1. This example also highlights how the Sahlqvist correspondence—an important technique of modal logic—can be applied to our method in order to describe canonical models of modal logics that are not axiomatised by formulas of rank 1.

In the end of the paper we revisit Fine’s normal forms for modal logic in a manner similar to Abramsky [2] and Ghilardi [18] and derive normal forms for positive modal logic. We also generalise Ghilardi’s result that every free modal algebra is a reduct of a temporal algebra to all varieties of modal and distributive modal algebras axiomatised by formulas of rank 1.

Other Related Work. Canonical models of modal logics have been investigated quite thoroughly. However, these investigations mostly concentrated on transitive modal logics; that is, modal logics with transitive Kripke frames. For an overview of these results we refer to [12, Section 8.6 and 8.7] (see also [7, Chapter 3] for similar results in the case of intuitionistic logic). The method of constructing canonical models for transitive modal logics is based on building the canonical model of a given logic layer by layer, that is, inductively on the depth of the canonical model. Although very useful, this method does not go through for non-transitive modal logics. For building free algebras for non-transitive modal logics one needs to use a different approach.

2 Dualities for Boolean Algebras and Distributive Lattices

In this section we briefly recall the Stone duality for Boolean algebras and the Priestley duality for distributive lattices.

2.1 Stone Duality for Boolean Algebras

A Stone space is a 0-dimensional (a topological space with a basis of clopens) compact Hausdorff space. For every Stone space X let $\mathsf{Clp}(X)$ denote the set of clopens (closed and open subsets) of X. We also let $\mathcal{P}(X)$ denote the powerset of X. The next theorem states the celebrated Stone representation theorem.

Theorem 2.1 (eg [19, 4.4], [14, 11.4]). *For every Boolean algebra B there is a Stone space X_B such that B is isomorphic to $(\mathsf{Clp}(X_B), \cup, \cap, -, \emptyset)$. If B is finite then X_B is finite and $\mathsf{Clp}(X_B) = \mathcal{P}(X_B)$.*

Proof. (Sketch) Let B be a Boolean algebra. Let $X_B :=$ the set of all maximal filters of B. For $a \in B$ let $\widehat{a} = \{x \in X_B : a \in x\}$. We declare $\{\widehat{a} : a \in B\}$ to be a basis for a topology on X_B. Then X_B becomes a Stone space and B is isomorphic to $\mathsf{Clp}(X_B)$.

Let $\mathbf{BA}$ denote the category of *Boolean algebras and Boolean homomorphisms*. Let also $\mathbf{Stone}$ denote the category of *Stone spaces and continuous maps*. The Stone representation theorem can be extended to corresponding categories.

Theorem 2.2 (see eg [19, 4.4]). $\mathbf{BA} \simeq \mathbf{Stone}^{op}$.

Proof. (Sketch) By Theorem 2.1 one only needs to deal with morphisms. Let $f : X \to Y$ be a continuous map. Then $f^{-1} : \mathsf{Clp}(Y) \to \mathsf{Clp}(X)$ is a Boolean homomorphism. Conversely, if $h : A \to B$ is a Boolean homomorphism, then the map $h^{-1} : X_B \to X_A$ is continuous. It is also easy to check that this correspondence is one to one.

Next we will discuss the duality between join preserving maps between Boolean algebras and special relations on corresponding Stone spaces. Let X and Y be Stone spaces. A relation $R \subseteq X \times Y$ is called *point closed* if $R[x] = \{y \in Y : xRy\}$ is a closed set for every $x \in X$. We say that R is a *clopen relation* if for every clopen $U \subseteq Y$ the set $\langle R \rangle U = \{x \in X : R[x] \cap U \neq \emptyset\}$ is a clopen subset of X.

Theorem 2.3 (see e.g., [8]). *There is a one-to-one correspondence between join preserving maps between Boolean algebras and point-closed and clopen relations on their dual Stone spaces. Moreover, on finite Stone spaces all relations are point-closed and clopen.*

Proof. (Sketch) (1) Let $h : A \to B$ be a join preserving map, that is, for all $a, b \in A$ we have $h(0) = 0$ and $h(a \vee b) = h(a) \vee h(b)$. Let X_A and X_B be the Stone spaces dual to A and B, respectively. We define $R_h \subseteq X_B \times X_A$ by

$$xR_hy \text{ iff } y \subseteq h^{-1}(x)$$

or, equivalently, xR_hy iff $(a \in y$ implies $ha \in x)$.[1] Conversely, if $R \subseteq X_B \times X_A$ is a point-closed and clopen relation, then $\langle R \rangle$ is the desired map $\mathsf{Clp}(X_A) \to \mathsf{Clp}(X_B)$.

Vietoris spaces and their duals, defined below, are central to our investigations.

Definition 2.4 (Functor V). *Let B be a Boolean algebra. Let $V(B)$ be the free Boolean algebra over the set $\{\Diamond a : a \in B\}$ modulo the equations, for all $a, b \in B$,*

$$(1) \quad \Diamond 0 = 0 \qquad\qquad (2) \quad \Diamond(a \vee b) = \Diamond a \vee \Diamond b$$

Thus, V is a functor on Boolean algebras. Now we define the dual to V on Stone spaces.

Definition 2.5 (Functor K). *For every Stone space X we let $K(X)$ be the set of all closed subsets of X equipped with a topology a subbasis of which is given by the sets*

$$\boxminus(U) = \{F \in K(X) : F \subseteq U\} \qquad\qquad \Diamond\!\!\!\Diamond(U) = \{F \in K(X) : F \cap U \neq \emptyset\}$$

where U ranges over clopen subsets of X.

The next theorem shows that the two definitions are dual to each other.

Theorem 2.6 ([19, Proposition 4.6]). *Let B a Boolean algebra and X its dual Stone space. Then the algebra $V(B)$ is dual to $K(X)$. If B is finite, $V(B)$ is dual to $\mathcal{P}(X)$.*

It follows from the definition of $V(B)$ that a map $\Diamond : B \to V(B)$ mapping each element $a \in B$ to $\Diamond a$ is join-preserving. The next proposition characterises the relation on $X \times K(X)$ which is dual to $\Diamond$. We just need to observe that $R_\Diamond$ defined as in the proof of Theorem 2.3 is $\in$.

Proposition 2.7. *Let $R_\Diamond \subseteq K(X_A) \times X_A$ be the relation corresponding to the join-preserving map $\Diamond : B \to V(B)$. Then for every $U \in K(X)$ and $x \in X_A$ we have $UR_\Diamond x$ iff $x \in U$.*

2.2 Priestley Duality for Distributive Lattices

We briefly review the duality between distributive lattices and Priestley spaces (Stone spaces with special partial orders). Recall that a subset U of an ordered set (X, R) is called an *upset* if for every $x, y \in X$ we have $x \in U$ and xRy imply $y \in U$. The complement of an upset is called a *downset*. A relation R on a Stone space X is said to satisfy *the Priestley separation axiom* if

$$\neg(xRy) \text{ implies there exists a clopen upset } U \text{ such that } x \in U \text{ and } y \notin U.$$

Definition 2.8. *A pair $\mathbb{X} = (X, R)$ is called a Priestley space if X is a Stone space and R a partial order satisfying the Priestley separation axiom.*

For every Priestley space $\mathbb{X} = (X, R)$ we let $\mathsf{ClpUp}(\mathbb{X})$ denote the set of all clopen upsets of $\mathbb{X}$. We also denote by $\mathsf{Up}(\mathbb{X})$ the set of all upsets of $\mathbb{X}$.

[1] Reading $c \in z$ as z satisfies c, we see that h acts here as a modal $\Diamond$.

Theorem 2.9 (see, e.g, [14, 11.23]). *For every distributive lattice D there is a Priestley space $\mathbb{X}_D$ such that D is isomorphic to $(\mathsf{ClpUp}(\mathbb{X}_D), \cup, \cap, \emptyset)$. If D is finite, then $\mathbb{X}_D$ is finite and $\mathsf{ClpUp}(\mathbb{X}_D) = \mathsf{Up}(\mathbb{X}_D)$.*

Let $\mathbf{DL}$ be the category of distributive lattices and lattice homomorphisms. Let also $\mathbf{Priest}$ denote the category of Priestley spaces and continuous order-preserving maps. We have the following analogue of Theorem 2.2.

Theorem 2.10 (see, e.g, [14, 11.30]). $\mathbf{DL} \simeq \mathbf{Priest}^{op}$.

Next we will briefly discuss the connection of meet and join preserving maps with Priestley relations. For a relation $R \subseteq X \times Y$ and $U \subseteq Y$ we let $[R]U = \{x \in X : R[x] \subseteq U\}$. Let $\mathbb{X} = (X, R)$ and $\mathbb{Y} = (Y, S)$ be Priestley spaces. A relation $Q \subseteq X \times Y$ is called *clopen increasing* (resp. *clopen decreasing*) if for every $x \in X$ the set $Q[x]$ is a closed upset of $\mathbb{Y}$ (resp. a closed downset of $\mathbb{Y}$) and for every clopen upset U of $\mathbb{Y}$ the set $[Q]U$ is a clopen upset of $\mathbb{X}$ (resp. $\langle Q \rangle U$ is a clopen upset of $\mathbb{X}$).

Theorem 2.11 (eg [11]). *There is a one-to-one correspondence between join preserving (resp. meet preserving) maps between distributive lattices and clopen increasing (resp. clopen decreasing) relations on their dual Priestley spaces. Moreover, on finite Priestley spaces a relation Q is clopen increasing (clopen decreasing) iff $Q[x]$ is an upset (resp. a downset) and $[Q]$ (resp. $\langle Q \rangle$) maps upsets to upsets.*

Vietoris construction for Priestley spaces and distributive lattices

Definition 2.12. *For every distributive lattice D let $V(D)$ denote the free distributive lattice over the set $\{\Diamond a : a \in D\} \cup \{\Box a : a \in D\}$ modulo the equations*

1. $\Diamond 0 = 0$, $\qquad \Box 1 = 1$,
2. $\Diamond(a \vee b) = \Diamond a \vee \Diamond b$, $\qquad \Box(a \wedge b) = \Box a \wedge \Box b$,
3. $\Box(a \vee b) \leq \Box a \vee \Diamond b$, $\qquad \Box a \wedge \Diamond b \leq \Diamond(a \wedge b)$.

Next we describe the dual construction of the Vietoris space for Priestley spaces [27]. Let $\mathbb{X} = (X, R)$ be a Priestley space. A set $F \subseteq X$ is called *convex* if for every $x, y, z \in X$ if $x, y \in F$ and xRz and zRy, then $z \in F$. For every Priestley space $\mathbb{X} = (X, R)$ let $\mathsf{Conv}(\mathbb{X})$ denote the set of all closed convex subsets of $\mathbb{X}$. We define a topology on $\mathsf{Conv}(\mathbb{X})$ a basis of which is given by the Boolean closure of the sets

$$\boxdot(U) = \{F \in \mathsf{Conv}(\mathbb{X}) : F \subseteq U\} \qquad \diamonddot(U) = \{F \in \mathsf{Conv}(\mathbb{X}) : F \cap U \neq \emptyset\}$$

where U ranges over clopen upsets of $\mathbb{X}$.[2] Moreover, for every $Y, Z \in \mathsf{Conv}(\mathbb{X})$ we define the so-called Egli-Milner order R^{EM} by

$$Y R^{EM} Z \text{ iff } Y \subseteq \langle R \rangle Z \text{ and } Z \subseteq \langle \check{R} \rangle Y.$$

[2] We note that this definition of topology on the set of closed and convex subsets of a Priestley space together with Theorem 2.13 below solves the problem raised in [27, Section 7.1] on how to define an analogue of the Vietoris topology on the set of closed and convex subsets of a Priestley space.

where $\check{R}$ is the converse of R. Then $(\mathsf{Conv}(\mathbb{X}), R^{EM})$ is a Priestley space. The next theorem, which is the Priestley space version of a theorem of Johnstone [20] (see also Palmigiano [27]), shows that the convex set construction on Priestley spaces is the dual to V.

Theorem 2.13. *Let D be a distributive lattice and $\mathbb{X} = (X, R)$ be its dual Priestley space. Then $(\mathsf{Conv}(\mathbb{X}), R^{EM})$ is the Priestley space dual to $V(D)$.*

As in the case of modal algebras, we have join-preserving and meet-preserving maps $\Diamond$ and $\Box$ from D to $V(D)$, mapping every element $a \in D$ to $\Diamond a$ and $\Box a$, respectively.

Proposition 2.14. *Let $R_\Diamond, R_\Box \subseteq \mathsf{Conv}(\mathbb{X}_\mathbb{A}) \times \mathbb{X}_\mathbb{A}$ be the relations corresponding to $\Diamond : D \to V(D)$ and $\Box : D \to V(D)$, respectively. Then $R_\Diamond = R_\Box$ and for every $U \in \mathsf{Conv}(\mathbb{X})$ and $x \in X$ we have $U R_\Diamond x$ iff $x \in U$.*

3 Modal Algebras and Distributive Modal Algebras

In this section we recall the definitions of modal and distributive modal algebras. We also look at the dual order-topological spaces of these algebras.

Modal algebras. A *modal algebra* (see e.g.[8, 5.2]) is a pair $(B, \Diamond_B)$ such that B is a Boolean algebra and $\Diamond_B : B \to B$ is a unary operator called a *modal operator* satisfying the equations of Definition 2.4. We also use a shorthand $\Box_B a = -\Diamond_B - a$, for every $a \in B$. Next we recall the representation theorem for modal algebras.

Definition 3.1 (see e.g., [8, Definition 5.65 and Proposition 5.83]). *A pair (X, R) is called a* modal space *if X is a Stone space and $R \subseteq X \times X$ is a point-closed and clopen relation.* [3]

Therefore, for every modal space, the algebra $(\mathsf{Clp}(X), \langle R \rangle)$ is a modal algebra. Moreover, every modal algebra can be represented in this way.

Theorem 3.2 (see, e.g., [8, Theorem 5.43]). *For every modal algebra $(B, \Diamond_B)$ there exists a modal space (X, R) such that $(B, \Diamond_B)$ is isomorphic to $(\mathsf{Clp}(X), \langle R \rangle)$.*

The modal space (X, R) is called the dual of $(B, \Diamond_B)$. Modal spaces can also be seen as coalgebras for the Vietoris functor K. In particular, every modal space can be represented as a Stone space X together with a continuous map $R : X \to K(X)$. The fact that R is well defined corresponds to $R[x]$ being closed, and R being continuous is equivalent to R being a clopen relation. For the details we refer to [22].

Distributive modal algebras. Lacking complements, one needs to represent both $\Diamond$ and $\Box$. A *distributive modal algebra* (see e.g., [11]) is a triple $(D, \Diamond_D, \Box_D)$ such that $\Diamond_D : D \to D$ and $\Box_D : D \to D$ are unary operations satisfying for every $a, b \in D$ the equations of Definition 2.12.

Definition 3.3. *A triple (X, R, Q) is called a* modal Priestley space *if $\mathbb{X} = (X, R)$ is a Priestley space and $Q \subseteq X \times X$ is a relation such that*

[3] Some authors also call modal spaces *descriptive frames*; see e.g., [8].

1. *$Q[x]$ is closed and convex for every $x \in X$; i.e., $Q[x] \in \mathsf{Conv}(\mathbb{X})$.*
2. *$\langle Q\rangle U \in \mathsf{ClpUp}(\mathbb{X})$ and $[Q]U \in \mathsf{ClpUp}(\mathbb{X})$ for every $U \in \mathsf{ClpUp}(\mathbb{X})$.*

Theorem 3.4. *(see [11]) For every distributive modal algebra $(D, \Diamond, \Box)$ there exists a modal Priestley space (X, R, Q) such that $(D, \Diamond, \Box)$ is isomorphic to $(\mathsf{ClpUp}(X), \langle R\rangle, [R])$.*

We mention here that the modal Priestley spaces can be seen as coalgebras for a functor Conv on Priestley spaces. In particular, every modal Priestley space can be represented as a Priestley space $\mathbb{X}$ together with an order-preserving and continuous map $Q : \mathbb{X} \to \mathsf{Conv}(\mathbb{X})$. That $Q[x]$ is closed and convex guarantees that Q is well defined. Q being order-preserving and continuous is equivalent to $\langle R\rangle$ and $[R]$ being well-defined maps on $\mathsf{ClpUp}(\mathbb{X})$. For the details we refer to [27].

We close this section by mentioning the connection between modal algebras and modal logic: a (positive) modal formula φ is a theorem of the basic modal logic **K** iff φ is valid in every (distributive) modal algebra.

4 Main Construction

As observed by Abramsky [2] and Ghilardi [18] the category of modal algebras is isomorphic to the category $Alg(V)$ of algebras for the functor V and, therefore, the free modal algebras can be obtained by a standard construction in category theory, the initial algebra sequence. Indeed, under fairly general circumstances [4], for a functor L on a category $\mathcal{C}$, the L-algebra L_ω free over $C \in \mathcal{C}$ is the colimit of the sequence $(L_n)_{n<\omega}$

$$L_0 \xrightarrow{e_0} L_1 \xrightarrow{e_1} L_2 \qquad \cdots \qquad L_\omega \tag{1}$$

where $L_0 = 0$ is the initial object of $\mathcal{C}$ and $L_{n+1} = (C + L)(L_n)$ and $e_{n+1} = (C + L)(e_n)$. Due to L_ω being a colimit, there is a canonical morphism $(C + L)(L_\omega) \to L_\omega$, the components of which provide the insertion of generators $C \to L_\omega$ and the L-algebra structure $L(L_\omega) \to L$. The same result can also be obtained from a slightly different sequence, the one used by [18], which is more convenient for our purposes

$$L_0 = C, \ \ L_{n+1} = (C + L)(L_n), \ \ e_0 : C \to C + L(C), \ \ e_{n+1} = (C + L)(e_n) \tag{2}$$

In this paper we are interested in the case where $\mathcal{C}$ is a variety[4] $\mathbf{V}$ and L encodes a signature that extends the signature $\Sigma_{\mathbf{V}}$ of $\mathbf{V}$ by additional operations Σ' and the equations $E_{\mathbf{V}}$ of $\mathbf{V}$ by additional equations E'. The terms in E' may use the operations built from the combined signature $\Sigma + \Sigma'$. We say that an equation in E' is of **rank 1** if every variable is under the scope of exactly one occurrence of an operation in Σ'. For example,$\Diamond p \to \Box p$ is of rank 1, but $p \to \Diamond p$ and $\Diamond\Diamond p \to \Diamond p$ are not. The precise relationship between L-algebras and algebras for an extended signature is studied in [23]. Roughly speaking, there is a one-to-one correspondence between functors $L : \mathbf{V} \to \mathbf{V}$ and extensions of $\mathbf{V}$ by operations Σ' and equations of rank 1 E'; under this

[4] For us, a variety is given by operations of *finite* arity and equations.

correspondence, $Alg(L)$ is isomorphic to the variety defined by operations $\Sigma_{\mathbf{V}} + \Sigma'$ and equations $E_{\mathbf{V}} + E'$. In other words, a variety is isomorphic to $Alg(L)$ iff it is axiomatized by equations of rank 1.

The basic construction we will describe is a variation of the sequence (2) which is both more special and more general. More special, because we take $\mathcal{C}$ to be a variety $\mathbf{V}$, more general because we consider sequences whose step-wise construction is not necessarily given by a functor as in (2), ie, for the time being, additional equations not of rank 1 are allowed.

Let $\mathbf{V}$ be a variety of algebras, let $\mathbf{V}_f$ be a variety obtained from $\mathbf{V}$ by expanding the signature of $\mathbf{V}$ by an operator f and let Ax be a set of axioms involving terms built from the operations of $\mathbf{V}$ and f. In other words, the algebras in $\mathbf{V}_f$ are the pairs (A, f), where $A \in \mathbf{V}$ and $f : A \to A$ is a map satisfying the axioms in Ax.[5] Further, we let $Eq(\mathbf{V})$ and $Eq(\mathbf{V}_f)$ be the equational theories of $\mathbf{V}$ and $\mathbf{V}_f$, respectively. For every $n \in \omega$ we will construct the n-generated free $\mathbf{V}_f$-algebra as a colimit of n-generated $\mathbf{V}$-algebras $A_0 \xrightarrow{i_0} A_1 \xrightarrow{i_1} \cdots$. A_0 is the n-generated free $\mathbf{V}$-algebra, and each A_{k+1} is obtained from A_k by freely adjoining to it the operator f. In other words, for each $k \in \omega$ the algebra A_k will be the algebra of all the non-$Eq(\mathbf{V}_f)$-equivalent terms of degree $\leq k$. Moreover, for each $k \in \omega$, there are two maps i_k and f_k between A_k and A_{k+1}. Since A_k is the algebra of all terms of degree $\leq k$ and A_{k+1} is the algebra of all terms of degree $\leq k + 1$, there is an embedding of A_k into A_{k+1}. The map i_k will be this embedding. Each term of degree m, for $m \leq k$ can be turned into a term of degree $m + 1 \leq k + 1$, by adjoining to it the operator f. The map f_k is exactly the map that adjoins f to each element of A_k. The operator $f_\omega : A_\omega \to A_\omega$ is obtained by lifting the maps $f_k : A_k \to A_{k+1}$ to A_ω.

The technical details are as follows. We fix a set $\mathrm{P} = \{p_1, \ldots, p_n\}$ of variables (or atomic propositions) in the language of $\mathbf{V}$. All the terms that we consider are build from P using the operations of $\mathbf{V}$ and f. For each $k \in \omega$, let S_k be the set of all terms in the language of $\mathbf{V}_f$ of degree $\leq k$, that is, of all terms that do not contain nestings of 'f' deeper than k. We say that an equation $s = t$, where s and t are terms, is deduced in $\mathbf{V}$ (resp. $\mathbf{V}_f$) from Γ and write $\Gamma \vdash_{\mathbf{V}} s = t$ (resp. $\Gamma \vdash_{\mathbf{V}_f} s = t$) if $s = t$ is deduced from Γ in the equational theory of $\mathbf{V}$ (resp. $\mathbf{V}_f$). Let $\equiv_{\mathbf{V}_f}$ be the relation on S_k defined by $s \equiv_{\mathbf{V}_f} t$ iff $\vdash_{\mathbf{V}_f} s = t$. Using the notation above we make the following

Definition 4.1. *The sequence $(T_k)_{k<\omega}$ is the sequence $(L_k)_{k<\omega}$, see (2), where C is the free $\mathbf{V}$-algebra over P and $L : \mathbf{V} \to \mathbf{V}$ maps A to the free algebra over $\{fa \mid a \in A\}$. The sequence $(A_k)_{k<\omega}$ is the quotient of $(T_k)_{k<\omega}$ by $\equiv_{\mathbf{V}_f}$.*

The algebra A_k is the algebra of (equivalence classes of) terms of degree $\leq k$, A_{k+1} is the algebra of terms of degree $\leq k + 1$ and the $i_k : A_k \to A_{k+1}$ obtained from quotienting the e_k of (2) are the obvious embeddings. Moreover, we define $f_k : A_k \to A_{k+1}$ to be the quotients of the maps $T_k \to L(T_k) \to T_0 + L(T_k)$ (insertion of generators followed by injection into a coproduct). Because of $i_{k+1} \circ f_k = f_{k+1} \circ i_k$, the f_k give rise, in the underlying category of sets, to a cocone over $(A_k)_{k<\omega}$, equipping A_ω with

[5] It is straightforward to replace f by a set of operations of finite arity. Here we consider only one unary operator to keep notation simple.

a $\mathbf{V}_f$-algebra structure $f_\omega : A_\omega \to A_\omega$. More concretely, f_k maps a term t of degree k to the term $f(t)$ of degree $k+1$; and, since each $a \in A_\omega$ comes from some A_k, we can write $f_\omega(a) = f_k(a)$ for some k.

Theorem 4.2. *The colimit of $(A_k)_{k<\omega}$ is the free n-generated $\mathbf{V}_f$-algebra.*

Note that if $\mathbf{V}$ is locally finite (ie the finitely generated algebras are finite), then each A_k is finite. Thus, if $\mathbf{V}$ is locally finite we can approximate every finitely generated free $\mathbf{V}_f$ algebra by finite n-generated $\mathbf{V}$ algebras. This is the case in our examples, where $\mathbf{V}$ is either the variety **BA** of Boolean algebras or the variety **DL** of distributive lattices.

The role of rank 1 axioms. The equational reasoning needed to determine whether two terms are identified in A_k may involve terms of degree larger than k. We therefore define $s \equiv^k_{\mathrm{AX}} t$ if, in the equational logic for the signature of $\mathbf{V}_f$, the equation $s = t$ has a proof from the axioms AX that only uses terms of degree $\leq k$.

Definition 4.3. *The sequence $(A'_k)_{k<\omega}$ is the quotient of $(T_k)_{k<\omega}$ by $(\equiv^k_{\mathrm{AX}})_{k<\omega}$.*

Theorem 4.4. *The colimit of $(A'_k)_{k<\omega}$ is the free n-generated $\mathbf{V}_f$-algebra.*

Note that the A_k are determined by the equational theory $Eq(\mathbf{V}_f)$ whereas the (A'_k) depend on the particular axiomatisation AX. Moreover, in contrast to the $i_k : A_k \to A_{k+1}$, the $i'_k : A'_k \to A'_{k+1}$ need not be injective. But if they are, one often can deduce desirable properties like decidability, normal forms, and others as shown by Ghilardi [18]. The following gives a sufficient condition. For a detailed definition of L below see [21, Section 4.1.3].

Theorem 4.5. *Let L be the functor on $\mathbf{V}$ where $L(A)$ is the free $\mathbf{V}$-algebra generated by $\{fa | a \in A\}$ modulo the axioms* AX*. If* AX *is of rank 1, the sequences $(A_k)_{k<\omega}$, $(A'_k)_{k<\omega}$ and $(L_k)_{k<\omega}$ (see (2)) coincide.*

In particular, we will exploit that for rank 1 axioms, the morphisms $i'_k : A'_k \to A'_{k+1}$ are injective.

5 Free Modal and Distributive Modal Algebras

We now combine Sections 2 - 4. For modal (distributive) algebras, the axioms AX of Section 4 are of rank 1 (see Definitions 2.4 and 2.12) and Theorem 4.5 applies.

Free modal algebras. Let B_0 be the n-generated free Boolean algebra, that is, B_0 is isomorphic to the powerset of a 2^n-element set (eg [19, 4.9]). Let X_0 be the dual of B_0. According to the construction discussed in the previous section we let $L = B_0 + V$ in (1), that is,

$$B_{k+1} = B_0 + V(B_k).$$

The maps $i_k, \Diamond_k : B_k \to B_{k+1}$ are as in Section 4. From Theorem 4.2 we obtain

Corollary 5.1. *The algebra $(B_\omega, \Diamond_\omega)$ obtained from the colimit of $(B_k)_{k<\omega}$ is the free modal algebra over B_0.*

Now we will look at the dual of $(B_\omega, \Diamond_\omega)$. Let X_0 be a 2^n element set (the dual of B_0) and (because of the duality of $\mathcal{P}$ and V (Theorem 2.6) and of $\times$ and $+$)

$$X_{k+1} = X_0 \times \mathcal{P}(X_k).$$

Theorem 5.2. *The sequence $(X_k)_{k<\omega}$ with maps $\pi_k : X_0 \times \mathcal{P}(X_k) \to X_k$ defined by*

$$\pi_k(x, A) = (x, \pi_{k-1}[A])$$

is dual to the sequence $(B_k)_{k<\omega}$ with maps $i_k : B_k \to B_{k+1}$. In particular, the π_k are surjective. Moreover, the relation $R_k \subseteq (X_0 \times \mathcal{P}(X_k)) \times X_k$ defined by

$$(x, A) R_k y \text{ iff } y \in A$$

is dual to $\Diamond_k : B_k \to B_{k+1}$ (see Theorem 2.3).

Remark 5.3. An element $x = (l, S) \in X_{k+1}$ can be understood as a tree with the root labelled by an element $l \in X_0$ and the children being the elements of $S \in \mathcal{P}(X_k)$. These trees have a rich history and have been studied, for example, by [3,2,6,18,5,31].

Corollary 5.4. *The modal space (X_ω, R_ω), where X_ω is the limit in* **Stone** *of the family $\{X_k\}_{k\in\omega}$ with the maps $\pi_{k+1} : X_{k+1} \to X_k$, and R_ω is defined by $(x_i)_{i\in\omega} R_\omega (y_i)_{i\in\omega}$ if $x_{k+1} R_k y_k$ for each $k \in \omega$ is (isomorphic to) to the dual of $(B_\omega, \Diamond_\omega)$.*

Remark 5.5. Note that (X_ω, R_ω) is isomorphic to the canonical model of the basic modal logic **K**; see [8, Section 5]. Therefore, a formula of modal logic is a theorem of **K** iff it is satisfiable in (X_ω, R_ω). Moreover, (X_ω, R_ω) is also the K-coalgebra cofree over X_0.

Free distributive modal algebras. Let D_0 be the free n-generated distributive lattice and $\mathbb{X}_0$ be its dual poset, that is, $\mathbb{X} = (\mathcal{P}(\mathbf{n}), \subseteq)$, where $\mathbf{n} = \{0, \ldots, n-1\}$ is an n-element set (eg [19, 4.8]). According to the construction discussed in the previous section we let $D_{k+1} = D_0 + V(D_k)$. where V is the Vietoris functor for distributive lattices. The maps $i_k, \Diamond_k : D_k \to D_{k+1}$ are as in Section 4. From Theorem 4.2 we obtain

Corollary 5.6. *The algebra $(D_\omega, \Diamond_\omega)$ obtained from the colimit of $(D_k)_{k<\omega}$ is the free modal distributive algebra over D_0.*

For the dual of $(D_\omega, \Diamond_\omega)$, Theorem 2.13 leads us to define $\mathbb{X}_{k+1} = \mathbb{X}_0 \times \mathsf{Conv}(\mathbb{X}_k)$.

Theorem 5.7. *The sequence $(\mathbb{X}_k)_{k<\omega}$ with $\pi_k : \mathbb{X}_0 \times \mathsf{Conv}(\mathbb{X}_k) \to \mathbb{X}_k$ defined by $\pi_k(x, A) = (x, \pi_{k-1}[A])$ is dual to the sequence $(D_k)_{k<\omega}$ with maps $i_k : D_k \to D_{k+1}$. In particular, the π_k are surjective. Moreover, the relation $Q_k \subseteq (\mathbb{X}_0 \times \mathsf{Conv}(\mathbb{X}_k)) \times \mathbb{X}_k$. defined by $(x, U) Q_k y$ iff $y \in U$ is dual to $\Diamond_k : D_k \to D_{k+1}$ (see Theorem 2.11).*

Corollary 5.8. *The modal Priestley space $(\mathbb{X}_\omega, Q_\omega)$, where $\mathbb{X}_\omega$ is the inverse limit in* **Priest** *of the family $\{\mathbb{X}_k\}_{k\in\omega}$ with the maps $\pi_{k+1} : \mathbb{X}_{k+1} \to \mathbb{X}_k$ and Q_ω is defined by $(x_i)_{i\in\omega} Q_\omega (y_i)_{i\in\omega}$ if $x_{k+1} Q_k y_k$ for each $k \in \omega$ is (isomorphic to) to the dual of $(D_\omega, \Diamond_\omega)$.*

Similar to the modal case the space $(\mathbb{X}_\omega, Q_\omega)$ is isomorphic to the canonical model of the basic positive modal logic and it is the final coalgebra for the functor $\mathbb{X}_0 \times \mathsf{Conv}$ on Priestley spaces.

6 Applications

Our first three applications are based on approximating the free algebras (and their duals) by the initial sequence of an appropriate functor as in (2). The last section, indicates how to go beyond rank 1 in a systematic way using Sahlqvist theory, but the details have to be left for future work.

6.1 Normal Forms

In this section we discuss normal forms for the elements of finitely generated free modal and distributive modal algebras. In logical terms this is equivalent to a normal form representation for the formulas of the corresponding language.

Definition 6.1. *Let* $\mathbf{V}$ *be a variety and* $A(n)$ *an* n*-generated free algebra of* $\mathbf{V}$*. We say that* $\mathbf{V}$ *admits a normal form representation if for every* $a \in A(n)$ *there exists a term* $t(a)$*, effectively computable from* a*, such that for every* $a, b \in A(n)$ *we have* $\vdash_{\mathbf{V}} a = b$ *iff* $t(a) = t(b)$*.*

We write $At(-)$ for the set of atoms of a Boolean algebra and, for $T \subseteq B_0$ and $S \subseteq At(B_k)$, let

$$\alpha_{S,T} := \bigwedge_{p \in T} p \wedge \bigwedge_{p \notin T} \neg p \wedge \bigwedge_{\varphi \in S} \Diamond\varphi \wedge \Box \bigvee_{\varphi \in S} \varphi$$

Lemma 6.2. $a \in B_{k+1}$ *is an atom iff* $a = \alpha_{S,T}$ *for some* $T \subseteq B_k$ *and* $S \subseteq At(B_k)$*.*

Similarly, for sets $T \subseteq D_0$ and $S \subseteq J(D_k)$, where $J(D_k)$ is the set of all join-irreducible elements of D_k we let

$$\beta_{S,T} := \bigwedge_{p \in T} p \wedge \bigwedge_{\varphi \in S} \Diamond\varphi \wedge \Box \bigvee_{\varphi \in S} \varphi$$

Lemma 6.3. $a \in D_{k+1}$ *is join-irreducible iff* $a = \beta_{S,T}$ *for some* $T \subseteq D_k$ *and* $S \subseteq J(D_k)$*.*

Corollary 6.4. *Basic modal logic and basic positive modal logic admit normal form representations.*

Proof. The result follows from above lemmas and the fact that every formula φ in n-variables can be seen as an element of the n-generated free algebra of the corresponding variety. As we showed above, for every element of the n-generated free modal or distributive modal algebra, there exists $k \in \omega$ such that φ belongs to B_k or D_k, respectively. Every element of a finite Boolean algebra (resp. finite distributive lattice) is a join of atoms (join-irreducible elements) that are below this element. Therefore, we obtain that $\varphi = \bigvee \alpha_{S,T}$ (resp. $\varphi = \bigvee \beta_{S,T}$).

Remark 6.5. The formulas $\alpha_{S,T}$ are the so-called Fine's canonical formulas [16,25]. Abramsky [2] and Ghilardi [18] derive these formulas in a way similar to ours.

6.2 Free Modal Algebras as Temporal Algebras

In this section we will give another corollary of our representations of free modal and distributive modal algebras. We will show that these algebras are reducts of temporal algebras. In case of modal logics this was first observed by Ghilardi [18].

Definition 6.6. *A modal algebra $(B, \Diamond)$ is a reduct of a temporal algebra if there exist $\Box_P : B \to B$ such that for every $a, b \in B$ we have $\Diamond a \leq b$ iff $a \leq \Box_P b$.*

A distributive modal algebra $(D, \Box, \Diamond)$ is a reduct of a temporal algebra if there exist $\Box_P, \Diamond_P : D \to D$ such that for every $a, b \in B$ we have $\Diamond a \leq b$ iff $a \leq \Box_P b$ and $\Diamond_P a \leq b$ iff $a \leq \Box b$.

Theorem 6.7. *Let $\mathbf{V}$ be a variety of modal or distributive modal algebras axiomatised by the formulas of rank 1. Then every finitely generated free $\mathbf{V}$-algebra is a reduct of a temporal algebra.*

Proof. (Sketch) We only look at the modal case. Let $(B_\omega, \Diamond_\omega)$ be the free $\mathbf{V}$-algebra. Then since each B_k is finite, the map $\Diamond_k : B_k \to B_{k+1}$ has a right adjoint $\Box_P^k : B_{k+1} \to B_k$, for every $k \in \omega$. Therefore, all we need to show is that the maps $\Box_P^k$ can be extended to the whole of B_ω. For this it is sufficient to prove that $i_{k-1}\Box_P^{k-1} = \Box_P^k i_k$. This equation holds if and only if for every $x \in X_k$, the equation $\pi_{k-1}^{-1} R_{k-1}[x] = R_k \pi_k^{-1}[x]$ holds. Checking that the latter equation is satisfied is easy and is based on the fact that for every $k \in \omega$ the maps π_k are surjective. We skip the details.

6.3 Pelczynski Compactification

In this section we characterise the space X_ω. We show that X_ω is homeomorphic to the so-called Pelczynski space.

Definition 6.8. *(see [28] and [26]) A space $\mathfrak{P}$ is called the* Pelczynski space *if $\mathfrak{P} = X_{iso} \cup X_{lim}$, where X_{iso} is a countable set of isolated points of $\mathfrak{P}$, X_{lim} is the set of limit points of $\mathfrak{P}$, the space X_{lim} is homeomorphic to the Cantor space $\mathfrak{C}$ and $\overline{X_{iso}} = \mathfrak{P}$.*

Theorem 6.9. *The underlying Stone space X_ω of the canonical model (X_w, R_w) is homeomorphic to the Pelczynski space $\mathfrak{P}$.*

Proof. (Sketch) The proof uses a result of Barr [6] that the set X_{iso} of isolated points of X_ω is dense in X_ω. We proceed by observing that the set X_{iso} is countable and that the set X_{lim} of the limit points of X_ω is uncountable and contains no isolated points in the topology induced from X_ω. Thus [15, 6.2.A.(c)], X_{lim} is homeomorphic to the Cantor space $\mathfrak{C}$ and therefore X_ω is homeomorphic to the Pelczynski space $\mathfrak{P}$.

Remark 6.10. In fact it is not a coincidence that the final coalgebra for the Vietoris functor is based on the Pelczynski space. We can prove that for every polynomial functor T on Stone spaces, the final coalgebra for T is finite or is homeomorphic to the Cantor space $\mathfrak{C}$, the Alexandroff compactification of a countable discrete space or to the Pelczynski space $\mathfrak{P}$.

6.4 Modal Logics Not Axiomatised by Rank 1 Axioms

In this section we indicate that our method can be extended to logics that are not axiomatised by axioms of rank 1. As a simple example we consider the logic **T** obtained from the basic modal logic **K** by adding to it the reflexivity axiom $p \to \Diamond p$ (see also Ghilardi [18, Section 5]). The Kripke frames for this logic are characterised by their accessibility relation being reflexive. Let $\mathbf{V_T}$ be the variety of modal algebras corresponding to $\mathbf{T}$. Since the reflexivity axiom is not of rank 1, in order to construct finitely generated free $\mathbf{V_T}$-algebra, we need to take quotients of the algebras B_k (Section 5). For every $k \in \omega$ we will quotient B_k by the relation $\equiv^k_{\mathrm{Ax}}$, $\mathrm{Ax} = \{p \to \Diamond p\}$, as in Definition 4.3. In other words, we define the sequence $(B'_k)_{k<\omega}$ by letting $B'_0 = B_0$ and $B'_{k+1} = B'_0 + V(B'_k)$ modulo $i_k a \to \Diamond_k a$, for $a \in B'_k$.

In dual terms, for every $k \in \omega$ we select a subset Y_k of X_k such that for every $U \subseteq Y_k$, we have $\pi_k^{-1}(U) \subseteq \langle R_k \rangle U$. This is equivalent to $\pi_k^{-1}(y) \subseteq \langle R \rangle \{y\}$ for every $y \in Y_k$. (The fact that we can move from sets to singletons is not a coincidence, it is a consequence of a more general fact that $\varphi \to \Diamond \varphi$ is a Sahlqvist formula [8, Section 3.6]). The latter condition is equivalent to $\pi_k(x, A) \in A$, for every $(x, A) \in Y_{k+1}$ and $k \in \omega$. Therefore, $Y_0 = X_0$ and for every $k \in \omega$, $Y_{k+1} = \{(x, A) : x \in Y_0, A \subseteq Y_k, \pi_k(x, A) \in A\}$. By induction on k we can also show that the restriction of π_k to Y_k is a surjection for every $k \in \omega$, which means that the quotients of i_k's are embeddings. Let $S_k = R_k \restriction Y_k$ and $\xi_k = \pi_k \restriction Y_k$, for each $k \in \omega$. We arrive at the following theorem.

Theorem 6.11. *The modal space $(\mathbb{Y}_\omega, S_\omega)$, where $\mathbb{Y}_\omega$ is the inverse limit in* **Stone** *of the family $\{\mathbb{Y}_k\}_{k<\omega}$ with the maps $\xi_{k+1} : \mathbb{Y}_{k+1} \to \mathbb{Y}_k$ and S_ω is defined by $(x_i)_{i<\omega} S_\omega (y_i)_{i<\omega}$ if $x_{k+1} S_k y_k$ for each $k < \omega$ is (isomorphic to) to the canonical model for the modal logic* $\mathbf{T}$.

This example suggests that a similar technique can be applied to other logics axiomatised by Sahlqvist formulas. Studying these questions in detail is one of the directions of future work.

7 Conclusions and Future Work

In this paper we presented a uniform method for constructing free algebras for algebras with operators axiomatised by equations of rank 1. We applied this general method to construct free modal algebras and free distributive modal algebras. We also recalled normal forms for modal logic and derived normal forms for positive modal logic. We list directions of further research.

One is to apply this construction to other non-classical logics for example intuitionistic logic, many-valued logics etc. More generally, one might be able to obtain results for varieties, in particular for locally-finite ones, that do arise from logic.

In the context of modal logic most of the important systems can not be axiomatised by the formulas of rank 1. Therefore, for describing free algebras for those systems, we need to adjust this method as indicated in Section 6.4. As adding axioms means to take quotients of the algebras A_k, it corresponds to taking subsets of the X_k on the dual side. To do this in a uniform way, one should look at Sahlqvist formulas.

Another interesting direction for further research is to spell out in detail the connection of this approach with the one of Moss [25]. It seems that Moss' filtration type technique has a direct representation in our construction. For various modal logics Moss constructs canonical models of formulas of finite modal degree. These models can be obtained from the models X_k by lifting in an appropriate way relations R_k between X_{k+1} and X_k to X_{k+1}.

The procedure to obtain normal forms should generalise to all logics of rank 1 (as long as the axioms are effectively given). This should be related to recent work of Schröder and Pattinson [29] on the complexity of rank 1 logics. Marx and Mikulás [24] also obtain complexity bounds for bi-modal logics by looking into algebras of terms of degree $\leq k$. Obtaining normal forms for logics that are not axiomatised by formulas of rank 1 is another interesting question.

We showed that the canonical model of the basic modal logic is based on the Pelczynski space. For other logics, however, such a characterisation does not exists. So a natural question is what are the underlying Stone spaces of canonical models of other modal logics. As we saw above the canonical model of the basic modal logic is a final coalgebra for the Vietoris functor. So an interesting question is whether the final coalgebra for every finite-set preserving functor is also based on the Pelczynski space.

All these questions hold also for positive modal logic and their variations considered in domain theory. But moreover, the recent work of Bruun and Gehrke [10], which connects ontologies with free distributive algebras with operators, adds another smack to this investigations: The axioms that [10] consider in their paper are of rank one.

Acknowledgements. We would like to thank Dimitri Pataraia, David Gabelaia, Mamuka Jibladze, Leo Esakia, Mai Gehrke and Hilary Priestley for many interesting discussions. We are also very grateful to the anonymous referees for valuable suggestions.

References

1. Abramsky, S.: Domain theory in logical form. Ann. Pure Appl. Logic 51, 1–77 (1991)
2. Abramsky, S.: A Cook's tour of the finitary non-well-founded sets. In We Will Show Them: Essays in Honour of Dov Gabbay. College Publications, 2005. Presented at BCTCS (1988)
3. Aczel, P.: Non-Well-Founded Sets. CSLI, Stanford (1988)
4. Adámek, J., Trnková, V.: Automata and Algebras in Categories. Kluwer Academic Publishers, Dordrecht (1990)
5. Baltag, A.: STS: A Structural Theory of Sets. PhD thesis, Indiana University (1998)
6. Barr, M.: Terminal coalgebras in well-founded set theory. Theoret. Comput. Sci. 114, 299–315 (1993)
7. Bezhanishvili, N.: Lattices of Intermediate and Cylindric Modal Logics. PhD thesis, ILLC, University of Amsterdam (2006)
8. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. CUP (2001)
9. Bonsangue, M., Kurz, A.: Duality for logics of transition systems. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, Springer, Heidelberg (2005)
10. Bruun, H., Gehrke, M.: Distributive lattice-structured ontologies, Draft (2006)
11. Celani, S., Jansana, R.: Priestley duality, a Sahlqvist theorem and a Goldblatt-Thomason theorem for positive modal logic. J. of the IGPL 7, 683–715 (1999)
12. Chagrov, A., Zakharyaschev, M.: Modal Logic. OUP (1997)

13. Davey, B., Goldberg, M.: The free p-algebra generated by a distributive lattice. Algebra Universalis 11, 90–100 (1980)
14. Davey, B., Priestley, H.: Introduction to Lattices and Order. CUP (1990)
15. Engelking, R.: General Topology. Heldermann Verlag (1989)
16. Fine, K.: Normal forms in modal logic. Notre Dame J. Formal Logic 16, 229–237 (1975)
17. Ghilardi, S.: Free Heyting algebras as bi-Heyting algebras. Math. Rep. Acad. Sci. Canada XVI 6, 240–244 (1992)
18. Ghilardi, S.: An algebraic theory of normal forms. Ann. Pure Appl. Logic 71, 189–245 (1995)
19. Johnstone, P.: Stone Spaces. CUP (1982)
20. Johnstone, P.: Vietoris locales and localic semilattices. In: Continuous lattices and their applications. Lecture Notes in Pure and Appl. Math. pp. 155–180. New York (1985)
21. Kupke, C.: Finitary Coalgebraic Logics. PhD thesis, ILLC, Amsterdam (2006)
22. Kupke, C., Kurz, A., Venema, Y.: Stone coalgebras. Theoret. Comput. Sci. 327, 109–134 (2004)
23. Kurz, A., Rosický, J.: Strongly complete logics for coalgebras. Submitted, electronically available
24. Marx, M., Mikulás, S.: An elementary construction for a non-elementary procedure. Studia Logica 72, 253–263 (2002)
25. Moss, L.: Finite models constructed from canonical formulas. Journal of Philosophical Logic (2007) (to appear)
26. Oka, S.: The topological types of hyperspaces of 0-dimensional compacta. Topology and its Applications 149, 227–237 (2005)
27. Palmigiano, A.: A coalgebraic view on positive modal logic. Theoret. Comput. Sci. 327, 175–195 (2004)
28. Pelczynski, A.: A remark on spaces 2^X for zero-dimensional X. Bull. Pol. Acad. Sci. 13, 85–89 (1965)
29. Schröder, L., Pattinson, D.: PSPACE bounds for rank 1 modal logics. In: LICS'06 (2006)
30. Venema, Y.: Algebras and coalgebras. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic, pp. 331–426. Elsevier, Amsterdam (2007)
31. Worrell, J.: On the final sequence of an finitary set functor. Theoret. Comput. Sci. 338, 184–199 (2005)

Coalgebraic Epistemic Update Without Change of Model*

Corina Cîrstea and Mehrnoosh Sadrzadeh

School of Electronics and Computer Science, University of Southampton
cc2,ms6@ecs.soton.ac.uk

Abstract. We present a coalgebraic semantics for reasoning about information update in multi-agent systems. The novelty is that we have one structure for both states and actions and thus our models do not involve the "change-of-model" phenomena that arise when using Kripke models. However, we prove that the usual models can be constructed from ours by categorical adjunction. The generality and abstraction of our coalgebraic model turns out to be extremely useful in proving preservation properties of update. In particular, we prove that positive knowledge is preserved and acquired as a result of epistemic update. We also prove common and nested knowledge properties of epistemic updates induced by specific epistemic actions such as public and private announcements, lying, and in particular unsafe actions of security protocols. Our model directly gives rise to a coalgebraic logic with both dynamic and epistemic modalities. We prove a soundness and completeness result for this logic, and illustrate the applicability of the logic by deriving knowledge properties of a simple security protocol.

1 Introduction

Modelling interactive multi-agent systems has a wide range of applications, e.g. in Artificial Intelligence, computer security and e-commerce. In such systems agents communicate and as a result their knowledge gets updated, and therefore one has to model the epistemics and dynamics of the system. The Kripke and algebraic models of these settings have been presented in [9,10,3,2,14]. The Kripke models have the advantage of being intuitive and concrete, while the algebraic setting benefits from high level features that result from mathematical abstraction.

In this paper we develop a coalgebraic semantics for dynamic epistemic systems, which combines the advantages of both the Kripke and the algebraic setting. Our model reasons about such systems in a uniform way, by treating both actions and agents as state transformers. Thus, we have only one structure that captures both dynamics and epistemics. This is contrary to the models of e.g. [10,3,1] that require subsequent "changes" to the epistemic structure to

* Research supported by EPSRC grant EP/D000033/1.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 158–172, 2007.
© Springer-Verlag Berlin Heidelberg 2007

model the dynamics. By "change" we mean either the *update product* between an epistemic Kripke structure and an action Kripke structure [3], or the *update functor* on the category of epistemic coalgebras [10,1]. In either case the epistemic structure is taken to be primitive and the dynamics is captured by operations on it. This brings us to the other novelty of our approach: we start our modelling task by fixing the epistemic actions, and then define the epistemic states based on these actions and on the agents participating in them. Again, this is contrary to the models of [10,3,1], which involve first fixing the epistemic states and then defining all possible epistemic actions on these states. Although at first sight our approach seems very different from the approach of [10,3,1], the two are strongly connected. In the main theorem of our paper we show how to construct from our models the models of [1] and vice versa, and prove that these constructions form a categorical adjunction[1].

Our approach has all the advantages of the approach in [3,1], for instance it benefits from a general updating schema and it reflects the epistemic structure of actions. Moreover, our approach does not have the usual weaknesses, for example operations on actions are a natural part of our models, e.g. sequential composition is simply unfolding the coalgebra maps twice and does not need to be defined separately. The generality and abstraction of our coalgebraic models turns out to be extremely useful in proving preservation properties of update. In particular, we prove that positive knowledge is preserved and acquired as a result of epistemic update. We also prove common and nested knowledge properties of epistemic updates induced by specific epistemic actions such as public and private announcements, lying, and in particular unsafe actions of security protocols. Finally, our model directly gives rise to a coalgebraic logic with both epistemic and dynamic modalities. This, for instance, cannot be done for the models of [10,1]. We prove a soundness and completeness result for the resulting logic. As an example of application, we derive the authentication properties of a security protocol.

An extended version of this paper is available electronically [8]. There we illustrate the applicability of our models to general scenarios involving both positive and negative knowledge, by presenting a new coalgebraic proof of the muddy children puzzle and a version of it with cheating children. Our proofs are based on restrictive recursion rather than the usual induction.

2 Coalgebraic Semantics for Actions and Agents

We consider coalgebras of the following signature functor $T : Set \to Set$

$$TX = \mathcal{P}_\kappa(X)^{Ag} \times (1 + X)^{Ac} \times \mathcal{P}(At)$$

where κ is a regular cardinal. A coalgebra map for the above functor is thus a triple $\gamma = \langle ap, up, val \rangle : S \to \mathcal{P}_\kappa(S)^{Ag} \times (1+S)^{Ac} \times \mathcal{P}(At)$. The valuation map val

[1] As noted by one of our referees, our model might share ideas with the recent *epistemic temporal models* of [4].

assigns to each state the facts that are true in that state. The (nondeterministic) appearance of states in S to agents in Ag is modelled by the function $ap : S \to \mathcal{P}(S)^{Ag}$, whereas the (deterministic) effect of actions in Ac on states in S is modelled by a function $up : S \to (1+S)^{Ac}$. Thus, $up(s)(a)$ stands for the effect of the action a on the state s, or the update of s by a. If this effect is the unique element $*$ of 1, that is, if $up(s)(a) = *$[2], we say that action a does not apply in state s; this should be the case, for instance, when a is the announcement of a fact that does not belong to $val(s)$. The choice of functor T automatically yields notions of T-*bisimulation* and T-*bisimilarity* for T-coalgebras (see e.g. [16]).

2.1 Restrictions to the Coalgebras

We are interested in using T-coalgebras to model the effect of communication actions on the information state or knowledge of agents. Examples of such actions are public or secret announcements, and message passing actions in a multi-agent system. We want to model the effect of updates with such actions on the appearances of states to the agents and on the valuations of states. In order to limit the behaviour of our systems to the effect of these actions, we require that the coalgebra maps satisfy some additional conditions, detailed in the following.

The communication actions that we model are *epistemic*, that is, they only affect the information states of agents, while leaving the facts of the world unchanged. Our first restriction, called *preservation of facts*, reflects this point:

$$val(up(s)(a)) = val(s) \quad \text{whenever} \quad up(s)(a) \neq *$$

It says that, if applicable to a state, an action does not change the valuation of that state. So the valuation of the effect of the action is the same as the valuation of the state before the action. We need this restriction to prove the preservation results in Section 2.2. In a more general approach, one can divide the set of actions into two subsets, namely *information-changing* actions and *fact-changing* actions, and only require this restriction for actions of the first type.

Our second restriction concerns the appearance of an update to each agent involved in the corresponding action. For applicable updates $up(s)(a) \neq *$, this will be related to the update of each of the agent's appearances $t \in ap(s)(A)$ with a finite subset of actions $Ac_{a,A} \subseteq Ac$, as follows

$$ap(up(s)(a))(A) = \left\{ up(t)(a') \mid t \in ap(s)(A),\, a' \in Ac_{a,A},\, up(t)(a') \neq * \right\}$$

where the actions $Ac_{a,A}$ depend both on the action a and on agent A's involvement in it, and are intended to capture agent A's appearance of the action a. This relation says that if an action a applies to a state s, then the appearance of its effect to an agent A is the same as the effect of one of the actions in $Ac_{a,A}$

[2] Here and in what follows, we assume that $1 \cap S = \emptyset$. Under this assumption, and to simplify the notation, we regard elements of the set $1 + S$ as being either $*$ or elements of S; that is, we make implicit the isomorphism between $1 + S$ and $1 \cup S$.

on one of the appearances to A of the original state s. The case when $Ac_{a,A}$ is a singleton $\{a'\}$ corresponds to a deterministic view of A about the real action a (with A thinking that a' is happening when in fact a is happening), whereas any non-singleton set $Ac_{a,A}$ captures A's uncertainty about the action taking place. We refer to the collection of all instances of this restriction (one for each action in Ac) as *rationality*.

The *content* of an epistemic action, as its name suggests, describes the information that is being transmitted as a result of the action taking place. We use the following syntax to denote specific contents[3]:

$$\mu := p \mid \Box_A \mu \mid \mathsf{tt} \mid \neg\mu \mid \bigwedge_{i\in\mathbb{I}} \mu_i$$

with $p \in At$ and $\mathbb{I}$ an arbitrary set. That is, the content of an action can be a fact, the knowledge or belief of some other content by an agent[4], the true proposition, the negation of a content, or a potentially infinite conjunction of contents[5]. In particular, the content can involve nested knowledge, as in $\Box_A\Box_B p$. We do not allow contents to refer to (the effect of) actions, as in $[q]-$; this avoids a circularity between requiring each action to have a content and allowing contents to depend on actions. Contents whose only occurrences of the negation operator immediately precede a fact are called *positive contents*, otherwise they are referred to as *negative contents*.

From now on, we assume that each action $a \in A$ has a content μ_a associated to it. Then, a should be applicable precisely to those states where its content μ_a is satisfied. This is encoded as a further restriction on T-coalgebras, referred to as the *content restriction*:

$$up(s)(a) \neq * \quad \text{iff} \quad s \models \mu_a$$

where the relation $\models$ between states and contents of actions is defined by structural induction on contents:

- $s \models p \quad \text{iff} \quad p \in val(s)$
- $s \models \Box_A\mu \quad \text{iff} \quad t \models \mu \quad \text{for all} \quad t \in ap(s)(A)$

and the usual clauses for the true proposition, negation and conjunction.

Definition 1. *An* appearance-update coalgebra *is a T-coalgebra additionally satisfying the preservation of facts, content, and rationality restrictions. We denote the set of all of these restrictions by $\mathcal{R}$.*

---

[3] We emphasise that this is just a syntax for expressing our second restriction on the content s of actions. The logic will be presented in section 5.

[4] Similarly to [3] and as a result of accommodating misinformation actions, our knowledge $\Box_A$ is not necessarily truthful. Indeed, one can also think of $\Box_A$ as belief in contexts where no wrong knowledge is allowed.

[5] The infinite contents are just a technicality that is needed later in order to establish a connection to the model of [1].

2.2 Preservation and Acquisition of Knowledge

An important consequence of the restrictions in $\mathcal{R}$ is the so-called *preservation of positive contents by updates*, made formal in the next result.

Proposition 1. *Let $(S, \langle ap, up, val \rangle)$ be an appearance-update coalgebra. Then for all positive contents μ, all states $s \in S$, and all actions $a \in Ac$ such that $up(s)(a) \neq *$, we have*

$$s \models \mu \quad \Longrightarrow \quad up(s)(a) \models \mu$$

Proof. The statement is proved by induction on μ. If μ is a fact or the negation of a fact, the conclusion follows directly from the preservation of facts. Now suppose that $s \models \mu'$ implies $up(s)(a) \models \mu'$ for all states $s \in S$ and applicable actions $a \in Ac$. Also, let $s \in S$ and $A \in Ag$ be such that $s \models \Box_A \mu'$. To show that $up(s)(a) \models \Box_A \mu'$ for any applicable action a, we use the rationality restriction to reduce $ap(up(s)(a))(A)$ to $\{up(t)(a') \mid t \in ap(s)(A),\ a' \in Ac_{a,A},\ up(t)(a') \neq *\}$. Thus, we must show that $up(t)(a') \models \mu'$ whenever $t \in ap(s)(A)$ and $a' \in Ac_{a,A}$ are such that $up(t)(a') \neq *$. But this follows from the induction hypothesis and the assumption that $s \models \Box_A \mu'$. The cases when μ is the true proposition or a conjunction of contents are trivial.

The above result does not hold for negative contents. That is, there exists an appearance-update coalgebra $(S, \langle ap, up, val \rangle)$, a state $s \in S$ with an applicable action $a \in Ac$ and a negative content μ such that $s \models \mu$ but $up(s)(a) \models \neg\mu$. For an example of such a situation, which gives rise to the epistemic puzzle of *muddy children*, see [8]. It is also not possible to generalise the above result to an exclusive one for positive contents. In particular, any appearance-update coalgebra that contains in its set of actions a neutral action τ with $\mu_\tau = tt$ and $Ac_{\tau,A} = \{\tau\}$ for all $A \in Ag$ is such an example. To see why, we refer the reader to the next section where we prove that such an action preserves all contents.

Another consequence of the restrictions in $\mathcal{R}$ is the following *acquisition of knowledge* after updates:

Proposition 2. *Let $(S, \langle ap, up, val \rangle)$ be an appearance-update coalgebra. Then for all agents $A \in Ag$, all states $s \in S$, and all applicable actions $a \in Ac$ with positive contents $\mu_{a'}$ for all $a' \in Ac_{a,A}$, we have*

$$up(s)(a) \models \Box_A \bigvee_{a' \in Ac_{a,A}} \mu_{a'}$$

Proof. Let $s \in S$ and $a \in Ac$ be such that $up(s)(a) \neq *$. We must show that for $A \in Ag$ we have $s' \models \bigvee_{a' \in Ac_{a,A}} \mu_{a'}$ for all $s' \in ap(up(s)(a))(A)$. By the rationality restriction on $ap(up(s)(a))(A)$, we must show that $up(t)(a') \models \bigvee_{a' \in Ac_{a,A}} \mu_{a'}$ whenever $t \in ap(s)(A)$ and $a' \in Ac_{a,A}$ are such that $up(t)(a') \neq *$. By the content restriction, the positivity of $\mu_{a'}$ and the preservation result we obtain $up(t)(a') \models \mu_{a'}$, which implies $up(t)(a') \models \bigvee_{a' \in Ac_{a,A}} \mu_{a'}$.

The known preservation results in the literature are special cases of our general results. For instance, it has been shown in [3] that contents that do not contain the epistemic modality are preserved under any update.

3 Epistemic Actions

In this section, we present epistemic actions, describe their contents and appearances, and prove their knowledge acquisition effects on agents[6].

Skip. This is the action τ in which nothing happens. We have $\mu_\tau = \mathtt{tt}$ and $Ac_{\tau,A} = \{\tau\}$ for all $A \in Ag$. This particular choice of μ_τ and $Ac_{\tau,A}$ is sufficient to guarantee that, in any appearance-update coalgebra, the skip action does not affect the epistemic content of states; that is, no knowledge is lost or acquired as a result of this action. This is formalised in the next two results, where we write $F : Set \to Set$ for the functor defined by $F(S) = \mathcal{P}_\kappa(S)^{Ag} \times \mathcal{P}(At)$.

Proposition 3. *In any appearance-update coalgebra $(S, \langle ap, up, val\rangle)$ where the set Ac of actions includes the τ action, $up(s)(\tau) \sim_F s$ for any state $s \in S$, where $\sim_F \subseteq S \times S$ denotes the F-bisimilarity relation on the F-coalgebra $(S, \langle ap, val\rangle)$.*

Proof. The statement follows by coinduction, namely by showing that the relation $R \subseteq S \times S$ given by $\{\, (s, up(s)(\tau)) \mid s \in S \,\}$ is an F-bisimulation. The preservation of facts ensures that R only relates states with the same valuations, whereas the rationality restriction guarantees closure of R under appearances.

Since F-bisimilar states satisfy the same content formulas, a stronger preservation of knowledge result can now be formulated for the τ action.

Corollary 1. *Let $(S, \langle ap, up, val\rangle)$ be an appearance-update coalgebra. Then for all contents μ and all states $s \in S$, we have*

$$s \models \mu \quad \Longleftrightarrow \quad up(s)(\tau) \models \mu$$

Public Announcements. The public announcement of a content μ is denoted $\mu!$, and has $Ac_{\mu!,A} = \{\mu!\}$ for all $A \in Ag$. We define *truthful common knowledge* of a content μ among a group β of agents as follows

$$CK_\beta\, \mu := \bigwedge_{\langle A_0, A_1, \ldots, A_n\rangle \in \beta^*} \Box_{A_0}\Box_{A_1}\ldots\Box_{A_n}\mu$$

where $\beta^* = \cup_{i\in\mathbb{N}}\, \beta^i$ is the set of all finite sequences of agents in β, including the empty sequence. Excluding the empty sequence provides us with the notion of *not necessarily truthful common knowledge*, denoted $\Box^*_\beta\mu$.

We now show that the public announcement of a positive content results in truthful common knowledge of that content.

[6] To be in line with the existing literature, we consider contents rather than preconditions of actions. The difference between the two is best seen in an example: the content of a public announcement is simply the announced proposition μ, whereas its precondition is the conjunction of μ with the knowledge of the announcer about μ.

Proposition 4. *For a state s of an appearance-update coalgebra in which the public announcement $\mu!$ with positive μ is possible, we have $up(s)(\mu!) \models CK_{Ag}\, \mu$.*

Proof. We must show that for any state s and any state s' connected to the applicable update $up(s)(\mu!) \neq *$ via any sequence of appearance maps we have $s' \models \mu$. Thus, we have a sequence of states $up(s)(\mu!) = s_0, s_1, \ldots, s_m = s'$ such that for $0 \leq j < m$ and some agent $A_j \in Ag$ we have $s_{j+1} \in ap(s_j)(A_j)$. For $m = 0$, $s_0 \models \mu$ follows from the applicability of update $up(s)(\mu!) \neq *$, the content restriction, and the preservation result. For $m > 0$, we have that s_m is in the following set of nested appearances

$$ap(\ldots(ap(ap(up(s)(\mu!))(A_0))(A_1))\ldots)(A_{m-1})$$

which, by applying the rationality restriction m times, is equal to

$$\{\, up(t_m)(\mu!) \mid t_m \in ap(t_{m-1})(A_{m-1}),\ \ldots,\ t_2 \in ap(t_1)(A_1),\ t_1 \in ap(s)(A_0), \\ \text{and}\ \ up(t_1)(\mu!) \neq *,\ up(t_2)(\mu!) \neq *,\ \ldots,\ up(t_m)(\mu!) \neq * \,\}$$

By the content restriction $up(t_m)(\mu!) \neq *$ is equivalent to $t_m \models \mu$, and from this by the preservation result it follows that $up(t_m)(\mu!) \models \mu$.

The closest special case to this proposition is that of [3], where the authors show that common knowledge of a fact implies preservation of any content under the public announcement of that fact.

Private Announcements. A private announcement $\mu!_\beta$ is the action of announcing the content μ to a subgroup of agents $\beta \subseteq Ag$ with $Ac_{\mu!_\beta,B} = \{\mu!_\beta\}$ for $B \in \beta$ and $Ac_{\mu!_\beta,A} = \{\tau\}$ for $A \notin \beta$.

As expected, one can prove that the private announcement of a positive content to a subgroup of agents results in truthful common knowledge of that content among the subgroup, and has no visible effect outside the subgroup.

Proposition 5. *For $\beta \subseteq Ag$ and a state s of an appearance-update coalgebra in which the private announcement $\mu!_\beta$ with positive μ is possible, we have $up(s)(\mu!_\beta) \models CK_\beta\, \mu$ and $ap(up(s)(\mu!_\beta))(A) \sim^{\mathcal{P}} ap(s)(A)$ for $A \notin \beta$*[7].

Lying. We write $\mu\dagger_A$ for the action with content $\neg\mu$ in which an agent A lies that μ to the rest of the agents. We have $Ac_{\mu\dagger_A,A} = \{\mu\dagger_A\}$ and $Ac_{\mu\dagger_A,B} = \{\mu!\}$ for any $B \neq A$.

Proposition 6. *For any agent $A \in Ag$, $\beta = Ag \setminus \{A\}$, and any state s of an appearance-update coalgebra in which the lying action $\mu\dagger_A$ with a positive μ is possible, we have $up(s)(\mu\dagger_A) \models \Box^*_\beta\, \mu$ and $up(s)(\mu\dagger_A) \models \Box_A \Box^*_\beta\, \mu$.*

Proof. In order to show that $up(s)(\mu\dagger_A) \models \Box^*_\beta\, \mu$, we must show that for any state s and any state s' connected to the applicable update $up(s)(\mu\dagger_A) \neq *$

[7] Here $\sim^{\mathcal{P}}$ denotes the lifting of the bisimilarity relation on S to $\mathcal{P}(S)$, see e.g. [12].

via any sequence of length more than 1 of appearance maps of agents in β, we have $s' \models \mu$. Consider a sequence of states $up(s)(\mu\dagger_A) = s_0, s_1, \ldots, s_m = s'$ with $1 \leq m$, such that for $0 \leq j < m$ and some agent $B_j \in Ag \setminus \{A\}$ we have $s_{j+1} \in ap(s_j)(B_j)$. It follows that s_m is in the following set of nested appearances

$$ap(\ldots(ap(ap(up(s)(\mu\dagger_A))(B_0))(B_1))\ldots)(B_{m-1})$$

which, by applying the rationality restriction m times (once for the lying action $\mu\dagger_A$ and B_0 and $m-1$ times for the public announcement $\mu!$ and B_1 to B_{m-1}), is equal to

$$\{\, up(t_m)(\mu!) \mid t_m \in ap(t_{m-1})(B_{m-1}),\ \ldots,\ t_2 \in ap(t_1)(B_1),\ t_1 \in ap(s)(B_0), \\ \text{and}\ \ up(t_1)(\mu!) \neq *,\ up(t_2)(\mu!) \neq *,\ \ldots,\ up(t_m)(\mu!) \neq * \,\}$$

By the content restriction $up(t_m)(\mu!) \neq *$ is equivalent to $t_m \models \mu$, and from this by the preservation result it follows that $up(t_m)(\mu!) \models \mu$.

Now to show that $up(s)(\mu\dagger_A) \models \Box_A \Box^*_\beta\, \mu$, we must show that $t \models \Box^*_\beta\, \mu$ for all $t \in ap(up(s)(\mu\dagger_A))(A)$. By the rationality restriction we have

$$ap(up(s)(\mu\dagger_A))(A) = \{\, up(w)(\mu\dagger_A) \mid w \in ap(s)(A),\ up(w)(\mu\dagger_A) \neq * \,\}$$

Since $up(w)(\mu\dagger_A) \neq *$ and μ is positive, it follows from $up(s)(\mu\dagger_A) \models \Box^*_\beta\, \mu$ that $up(w)(\mu\dagger_A) \models \Box^*_\beta\, \mu$.

Security Actions. A security action $\mu \star \mu'_{\{A\},\beta,\gamma}$ is a private announcement in an unsafe communication channel, where the intruders in γ change the original content μ, sent by A to the agents in β, to a fake one μ'. In this case we have $Ac_{\mu\star\mu'_{\{A\},\beta,\gamma},A} = \{\mu!_{\beta\cup\{A\}}\}$, $Ac_{\mu\star\mu'_{\{A\},\beta,\gamma},B} = \{\mu'!_{\beta\cup\{A\}}\}$ for agents $B \in \beta$, $Ac_{\mu\star\mu'_{\{A\},\beta,\gamma},C} = \{\mu \star \mu'_{\{A\},\beta,\gamma}\}$ for the intruders $C \in \gamma$, while $Ac_{\mu\star\mu'_{\{A\},\beta,\gamma},D} = \{\tau\}$ for any other agent $D \in Ag \setminus (\{A\} \cup \beta \cup \gamma)$.

Proposition 7. *For any agents $B \in \beta$, $C \in \gamma$, and any state s of an appearance-update coalgebra in which the security action $\mu\star\mu'_{\{A\},\beta,\gamma}$ with positive μ and μ' is possible, we have $up(s)(\mu\star\mu'_{\{A\},\beta,\gamma}) \models \Box_A CK_\beta\, \mu$, $up(s)(\mu\star\mu'_{\{A\},\beta,\gamma}) \models \Box^*_\beta \Box_A\, \mu'$ and $up(s)(\mu \star \mu'_{\{A\},\beta,\gamma}) \models CK_\gamma(\Box_A CK_\beta\, \mu \wedge \Box^*_\beta \Box_A\, \mu')$.*

4 Comparison with Baltag's Coalgebraic Model

We now compare our coalgebraic semantics with that of [1]. In loc. cit., both *epistemic states* and *epistemic actions* are defined via final coalgebras. Two different functors of a similar shape are used to achieve this. However, none of these functors accounts for epistemic updates, which are instead modelled using a partial product between coalgebras of states and coalgebras of actions.

The functor used in [1] to model epistemic states is

$$F : Set \to Set, \qquad F(S) = \mathcal{P}_\kappa(S)^{Ag} \times \mathcal{P}(At)$$

Appearances of states to agents are encoded as elements of $\mathcal{P}_\kappa(S)^{Ag}$, while their valuations are encoded using sets of atomic propositions. Epistemic states are then defined as elements of the final F-coalgebra Ψ. Similarly, epistemic actions are defined as elements of the final coalgebra of the functor

$$G : Set \to Set, \qquad G(\Sigma) = \mathcal{P}_\kappa(\Sigma)^{Ag} \times \mathcal{P}(\Psi)$$

with $\mathcal{P}_\kappa(\Sigma)^{Ag}$ encoding the appearances of actions to agents, and $\mathcal{P}(\Psi)$ encoding the contents of actions (as sets of epistemic states where the actions are applicable). Finally, epistemic updates are modelled using a functor

$$- \otimes - : Coalg(F) \times Coalg(G) \to Coalg(F)$$

which takes a pair consisting of an F-coalgebra $(S, \langle ap_S, val_S\rangle)$ and a G-coalgebra $(\Sigma, \langle ap_\Sigma, cont_\Sigma\rangle)$ to another F-coalgebra whose elements correspond to updates of states in S with actions in Σ. Writing $!_S : S \to \Psi$ for the unique F-coalgebra morphism arising from the finality of Ψ, the coalgebra for the updated states has carrier

$$S \otimes \Sigma = \{(s, \sigma) \in S \times \Sigma \mid \ !_S(s) \in cont_\Sigma(\sigma)\}$$

That is, updated states are pairs consisting of a state $s \in S$ and an action $\sigma \in \Sigma$, with the additional property that the content of the action σ makes it applicable to the state s[8]. The coalgebra map $\langle ap_{S\otimes\Sigma}, val_{S\otimes\Sigma}\rangle : S \otimes \Sigma \to F(S \otimes \Sigma)$ is given by

$$\begin{aligned} ap_{S\otimes\Sigma}(s,\sigma)(A) &= \{(s',\sigma') \in S\otimes\Sigma \mid s' \in ap_S(s)(A),\ \sigma' \in ap_\Sigma(\sigma)(A)\} \\ val_{S\otimes\Sigma}(s,\sigma) &= val_S(s) \end{aligned}$$

That is, the appearances of updated states to agents are computed using both the appearances of the original states and the appearances of the actions producing the updates.

In contrast to the above, our approach uses only one functor, which incorporates both the epistemic and the dynamic aspect of states. This internal modelling of updates is made possible by the fact that we apriorily fix a universe Ac of actions, together with its epistemic structure. The set Ac should be taken to contain those epistemic actions (elements of the final G-coalgebra) which are of interest to the modelling of a particular multi-agent scenario. In this setting, our choice to specify for each action $a \in Ac$ and agent $A \in Ag$, a set $Ac_{a,A}$ of actions that are perceived by A, together with for each action a a content μ_a, gives rise to a coalgebra $(Ac, \langle ap_{Ac}, \mu_{Ac}\rangle)$ of the following functor

$$H : Set \to Set, \qquad H(\Sigma) = \mathcal{P}_\kappa(\Sigma)^{Ag} \times C$$

where the set C consists of equivalence classes of content formulas. Here, two content formulas are said to be (semantically) equivalent if they are satisfied by

[8] Here it is assumed that the applicability of an action is invariant under bisimulation, and therefore an action is applicable to a state precisely when it is applicable to its image under the unique coalgebra morphism into the final F-coalgebra.

the same states of any F-coalgebra. The map ap_{Ac} of the previously mentioned H-coalgebra is given by $ap_{Ac}(a)(A) = Ac_{a,A}$ for $a \in Ac$ and $A \in Ag$, whereas the map μ_{Ac} takes actions $a \in Ac$ to the equivalence class of their content $[\mu_a]$. In this way, we do not distinguish between actions that have both the same epistemic structure and semantically equivalent contents.

In order to make precise the relationship between appearance-update T-coalgebras and the models of [1], we make the dependency of T on the set Ac of actions explicit, and write $T_{Ac} : Set \to Set$ for the functor given by

$$T_{Ac}X = (\mathcal{P}_\kappa X)^{Ag} \times (1 + X)^{Ac} \times \mathcal{P}(At)$$

Next, we let $AppUpCoalg$ denote the category whose objects are pairs (Ac, S), with $Ac = (Ac, \langle ap_{Ac}, \mu_{Ac}\rangle)$ an H-coalgebra and $S = (S, \langle ap_S, up_S, val_S\rangle)$ an appearance-update T_{Ac}-coalgebra. The H-coalgebra Ac encodes the structure on the set Ac of actions required to formulate the content and rationality restrictions of Section 2, whereas the T_{Ac}-coalgebra S specifies a set of states carrying both an epistemic structure and a dynamic structure w.r.t. the actions in Ac. To define the arrows of the category $AppUpCoalg$, we first note that any H-coalgebra morphism $f : Ac \to Ac'$ induces a functor

$$U_f : Coalg(T_{Ac'}) \to Coalg(T_{Ac})$$

which takes a $T_{Ac'}$-coalgebra $(S, \langle ap_S, up_S, val_S\rangle)$ to the T_{Ac}-coalgebra with the same carrier set and appearance and valuation maps, but with an update map w.r.t. the set Ac instead. This update is derived from the curried version $ev(up_S)\colon S \times Ac' \to (1 + S)$ of the update map up_S of the $T_{Ac'}$-coalgebra, as shown below

$$S \times Ac \xrightarrow{id_S \times f} S \times Ac' \xrightarrow{ev(up_S)} 1 + S$$

The curried version of this composition is the update map of the T_{Ac}-coalgebra

$$ev(ev(up_S) \circ (id_S \times f))\colon S \to (1 + S)^{Ac}$$

So we have $U_f(S, \langle ap_S, up_S, val_S\rangle) = (S, \langle ap_S, ev(ev(up_S) \circ (id_S \times f)), val_S\rangle)$. Now the arrows from (Ac, S) to (Ac', S') in the category $AppUpCoalg$ are pairs of maps (f, g) with $f : Ac \to Ac'$ an H-coalgebra morphism and $g : S \to U_f S'$ a T_{Ac}-coalgebra morphism. The former encodes the actions in Ac as actions in Ac', whereas the latter translates the states of the T_{Ac}-coalgebra S to states of the $T_{Ac'}$-coalgebra S'.

The last piece of notation we require before relating our models to those of [1] concerns *characteristic formulas* for states of F-coalgebras. These are infinitary formulas of the form used in Section 2 to specify the contents of epistemic actions, and have the additional property that they characterise individual states of F-coalgebras up to bisimulation. Their existence is guaranteed by the κ-accessibility of F. In particular, for any state ψ of the final F-coalgebra Ψ, there exists a characteristic formula ϕ_ψ with the property that, given any state s of an F-coalgebra S, we have $s \models \phi_\psi$ if and only if $!_S(s) = \psi$.

We are now ready to describe the relationship between the models of [1] and our appearance-update coalgebras. This is given by an adjunction

$$Coalg(F) \times Coalg(G) \underset{R}{\overset{L}{\rightleftarrows}} AppUpCoalg \quad (L \dashv R)$$

Definition 2 (Left adjoint). *We let $L : Coalg(F) \times Coalg(G) \to AppUpCoalg$ be defined by $L(S, \Sigma) = ((\Sigma, \langle ap_\Sigma, \mu_\Sigma \rangle), (S', \langle ap_{S'}, up_{S'}, val_{S'} \rangle))$, where*

- *$\mu_\Sigma(\sigma) = \bigvee_{\psi \in cont_\Sigma(\sigma)} \phi_\psi$, where for $\psi \in \Psi$, ϕ_ψ is the characteristic formula of ψ.*
- *$S' = (S', \langle ap_{S'}, up_{S'}, val_{S'} \rangle)$ is a T_Σ-coalgebra obtained by*
  1. *first letting $S' = (S', \langle ap_{S'}, val_{S'} \rangle) = \cup_{i \in \omega}(S_i, \langle ap_{S_i}, val_{S_i} \rangle)$ where*

     $$S_0 = S, \qquad S_{i+1} = S_i \otimes \Sigma \quad \textit{for } i \in \omega$$

     (Note that, by definition, each of the sets S_i comes equipped with an F-coalgebra structure, and S' inherits this structure.)
  2. *subsequently endowing the set S' with an update map $up_{S'} : S' \to (1 + S')^\Sigma$, by letting*

     $$up_{S'}(s_i)(\sigma) = \begin{cases} (s_i, \sigma) & if (s_i, \sigma) \in S_{i+1} \\ * & otherwise \end{cases}, \quad \textit{for } i \in \omega$$

In informal terms, the functor L constructs an H-coalgebra Σ and a T_Σ-coalgebra S' from a pair consisting of an F-coalgebra S and a G-coalgebra Σ. The H-structure of Σ is determined by the G-structure of Σ in a trivial way: appearances of actions to agents are already defined by the H-structure, whereas the content map $\mu_\Sigma : \Sigma \to C$ acts on an action $\sigma \in \Sigma$ by logically joining all the characteristic formulas of states in the content of σ. The T_Σ-coalgebra S' is obtained by performing consecutive update products with the actions in Σ, first on S, and then on the result of the preceding update product:

$$S \longmapsto S \otimes \Sigma \longmapsto (S \otimes \Sigma) \otimes \Sigma \longmapsto \ldots$$

and subsequently taking the union of the resulting F-coalgebras and endowing it with an update map.

Proposition 8. *The T_Σ-coalgebra S' is an appearance-update coalgebra.*

Proof. We have to show that S' satisfies all the restrictions in $\mathcal{R}$. The preservation of facts follows directly from the definitions of S_i and S': for $i \in \omega$, whenever $up_{S'}(s_i)(\sigma) \in S'$, that is, whenever $(s_i, \sigma) \in S_{i+1}$, we have

$$val_{S'}(up_{S'}(s_i)(\sigma)) = val_{S_{i+1}}(s_i, \sigma) = val_{S_i}(s_i) = val_{S'}(s_i)$$

For the rationality restriction, assuming $up_{S'}(s_i)(\sigma) \in S'$, that is, $(s_i, \sigma) \in S_{i+1}$, we have

$$\begin{aligned} ap_{S'}(up_{S'}(s_i)(\sigma))(A) &= ap_{S_{i+1}}(s_i,\sigma)(A) \\ &= \{(s',\sigma') \in S_i \otimes \Sigma \mid s' \in ap_{S_i}(s_i)(A),\ \sigma' \in ap_\Sigma(\sigma)(A)\} \end{aligned}$$

and

$$\begin{aligned} \{up_{S'}(t)(\sigma') \mid t \in ap_{S'}(s_i)(A), \sigma' \in ap_\Sigma(\sigma)(A),\ up_{S'}(t)(\sigma') \neq *\} = \\ \{(t,\sigma') \mid t \in ap_{S_i}(s_i)(A), \sigma' \in ap_\Sigma(\sigma)(A), (t,\sigma') \in S_{i+1}\} = \\ \{(t,\sigma') \in S_i \otimes \Sigma \mid t \in ap_{S_i}(s_i)(A), \sigma' \in ap_\Sigma(\sigma)(A)\} \end{aligned}$$

for each $i \in \omega$, and therefore

$$\begin{aligned} & ap_{S'}(up_{S'}(s')(\sigma))(A) = \\ & \qquad \{up_{S'}(t)(\sigma') \mid t \in ap_{S'}(s')(A), \sigma' \in ap_\Sigma(\sigma)(A),\ up_{S'}(t)(\sigma') \neq *\} \end{aligned}$$

Finally, for the content restriction, we have, for each $i \in \omega$

$$\begin{array}{rcll} up_{S'}(s_i)(\sigma) \in S' & \text{iff} & (s_i,\sigma) \in S_{i+1} = S_i \otimes \Sigma & \text{iff} \\ !_{S_i}(s_i) \in cont_\Sigma(\sigma) & \text{iff} & !_{S_i}(s_i) \models \mu_\Sigma(\sigma) & \text{iff} \\ !_{S'}(s_i) \models \mu_\Sigma(\sigma) & \text{iff} & s_i \models \mu_\Sigma(\sigma) & \end{array}$$

and hence $up_{S'}(s_i)(\sigma) \in S'$ iff $s_i \models \mu_\Sigma(\sigma)$.

Definition 3 (Right adjoint). *We define $R : AppUpCoalg \to Coalg(F) \times Coalg(G)$ by $R(Ac, S) = ((S, \langle ap_S, val_S\rangle), (Ac, \langle ap_{Ac}, cont_{Ac}\rangle))$, where the map $cont_{Ac} : Ac \to \mathcal{P}(\Psi)$ takes an action $a \in Ac$ to the set of states in the final F-coalgebra which satisfy the formula $\mu_{Ac}(a)$.*

Informally speaking, the functor R takes a pair consisting of an H-coalgebra Ac and an appearance-update T_{Ac}-coalgebra S, and produces an F-coalgebra and a G-coalgebra. The F-coalgebra is obtained from S by forgetting its update map and keeping everything else intact. The G-coalgebra has the same carrier set and epistemic structure as Ac, and a content map obtained essentially by replacing content formulas with their denotations in the final F-coalgebra.

Theorem 1. *L is left adjoint to R.*

Proof. We begin by examining the unit and counit of this adjunction. Since the categories $Coalg(H)$ and $Coalg(G)$ are naturally isomorphic, it is the move from T_{Ac}-coalgebras to F-coalgebras and back that makes the adjunction non-trivial.

For the unit of the adjunction, the inclusions $\eta_{S,\Sigma} : S \to S \cup (S \otimes \Sigma) \cup ((S \otimes \Sigma) \otimes \Sigma) \cup \ldots$ together with the natural isomorphism between $Coalg(H)$ and $Coalg(G)$ give rise to a natural transformation $\eta : Id_{Coalg(F) \times Coalg(G)} \Rightarrow R \circ L$.

For the counit, the maps $\epsilon_{Ac,S} : S \cup (S \otimes Ac) \cup ((S \otimes Ac) \otimes Ac) \cup \ldots \to S$ defined inductively by

$$\epsilon_{Ac,S}(s) = s, \qquad \epsilon_{Ac,S}(s_i, a) = up_S(\epsilon_{Ac,S}(s))(a) \quad \text{for } i \in \omega \text{ and } s_i \in S_i$$

together with the natural isomorphism between $Coalg(H)$ and $Coalg(G)$, yield a natural transformation $\epsilon : L \circ R \Rightarrow Id_{AppUpCoalg}$.

We show that η and ϵ indeed constitute the unit and counit of an adjunction $L \dashv R$. To this end, we fix $(S, \Sigma) \in Coalg(F) \times Coalg(G)$ and $(Ac, S') \in AppUpCoalg$. For $(f, g) : (S, \Sigma) \to R(Ac, S')$, the map $f^{\#} : S \cup (S \otimes \Sigma) \cup ((S \otimes \Sigma) \otimes \Sigma) \cup \ldots \to S'$ defined inductively by

$$f^{\#}(s) = f(s), \qquad f^{\#}(s_i, \sigma) = up_{S'}(f^{\#}(s_i))(g(\sigma)) \quad \text{for } i \in \omega$$

is a T_{Ac}-coalgebra morphism that satisfies $R(g, f^{\#}) \circ \eta_{(S,\Sigma)} = (f, g)$. Furthermore, any T_{Ac}-coalgebra morphism with the above property is defined in this way. For $(h, k) : L(S, \Sigma) \to (Ac, S')$, the map $k^{\flat} : S \to S'$ given by $k\restriction_S$ defines an F-coalgebra morphism that satisfies $\epsilon_{(Ac,S')} \circ L(k^{\flat}, h) = (h, k)$. Furthermore, this last requirement uniquely determines the definition of $k^{\flat}$.

5 Coalgebraic Dynamic Epistemic Logic

Coalgebras give rise to modal logics in different ways, for example the coalgebraic logic of Moss [13], the temporal logic of Jacobs [11], and the modular logic of Cîrstea and Pattinson [5,6]. In previous work [15], we showed how one obtains an algebraic logic from our functor by predicate lifting, and investigated the connection between this logic and the algebraic dynamic epistemic logic of [2,14]. Cîrstea and Pattinson have shown how complete and expressive coalgebraic logics can be derived in a modular fashion for an inductively-defined class of endofunctors on Set. By applying this method to our setting, we obtain a logic with a multi-sorted syntax, which is expressive – that is, two states are bisimilar if and only if they satisfy the same formulas –, and admits a sound and complete proof system. Because of the particular shape of the functor T and of the axioms in the associated proof system, the multi-sorted syntax of this logic can be simplified to the following single-sorted syntax, with no loss in expressiveness

$$\phi ::= tt \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \Box_A \phi \mid [a]\phi$$

The standard knowledge and dynamic modalities, that is, $\Box_A$ (to be read as 'A knows ϕ') and $[a]$ (to be read as 'after a, ϕ'), are recovered by letting $\Box_A \phi ::= [\pi_1][A]\Box\phi$ and $[a]\phi ::= [\pi_2][a][\kappa_2]\phi$[9]. In particular, the statement 'action a does not go through' is captured by the formula $[a]\mathsf{ff}$. Using the simplified syntax, the original proof system is equivalent to the following set of axioms and rules

$$\vdash \bigcirc tt \qquad \vdash \bigcirc\phi \wedge \bigcirc\psi \to \bigcirc(\phi \wedge \psi) \qquad \frac{\vdash \phi \to \psi}{\vdash \bigcirc\phi \to \bigcirc\psi}$$

for $\bigcirc \in \{\Box_A, [a]\}$, and

$$\vdash [a](\phi \vee \psi) \to [a]\phi \vee [a]\psi$$

on top of propositional logic[10]. As a consequence of the results in [6], this proof system is sound and complete w.r.t. T-coalgebras. However, in order to formulate

[9] See [6] for details of the multi-sorted syntax.

[10] As in [6], we include all instances of propositional tautologies and the modus ponens rule into our set of axioms and rules.

a soundness and completeness result w.r.t. appearance-update coalgebras, the restrictions defining appearance-update coalgebras must also be axiomatised. To this end, we add the following axioms to the previous proof system:

$$\vdash [a]\, p \leftrightarrow (\neg[a]\mathsf{ff} \rightarrow p) \qquad \vdash [a]\Box_A\, \phi \leftrightarrow (\neg[a]\mathsf{ff} \rightarrow \bigwedge_{a' \in Ac_{a,A}} \Box_A\, [a']\, \phi)$$

$$\phi_a \leftrightarrow \neg[a]\mathsf{ff}$$

where for an action $a \in Ac$, its content is denoted by ϕ_a. There is one such axiom for each epistemic action a and each (type of) agent A.

Example of Derivation. Consider a simple Man in the Middle Attack: agent A sends a message with factual content p to agent B, but on the way the intruder C changes p to another fact p' and thus B receives p' instead. If we assume that A does not suspect the interception, after sending p he believes that B believes in p. Similarly, upon receipt, B believes that A believes in p'. In security terms and since A and B do not suspect the interception, they will wrongly *authenticate* with each other. We use the encoding of the security action in Section 2 to prove that $\vdash [p \star p'_{A,B,C}]\Box_A\Box_B p$. The proof steps are sketched below:

$$\begin{array}{rl}
 & \vdash \mathsf{tt} \\
\text{(propositional logic)} & \vdash \neg[p \star p'_{A,B,C}]\mathsf{ff} \rightarrow \mathsf{tt} \\
\text{(modular logic)} & \vdash \neg[p \star p'_{A,B,C}]\mathsf{ff} \rightarrow \Box_A\mathsf{tt} \\
\text{(propositional logic)} & \vdash \neg[p \star p'_{A,B,C}]\mathsf{ff} \rightarrow \Box_A(\neg[p!_B]\mathsf{ff} \rightarrow \mathsf{tt}) \\
\text{(modular logic)} & \vdash \neg[p \star p'_{A,B,C}]\mathsf{ff} \rightarrow \Box_A(\neg[p!_B]\mathsf{ff} \rightarrow \Box_B\mathsf{tt}) \\
\text{(propositional logic)} & \vdash \neg[p \star p'_{A,B,C}]\mathsf{ff} \rightarrow \Box_A(\neg[p!_B]\mathsf{ff} \rightarrow \Box_B(p \rightarrow p)) \\
\text{(content axiom)} & \vdash \neg[p \star p'_{A,B,C}]\mathsf{ff} \rightarrow \Box_A(\neg[p!_B]\mathsf{ff} \rightarrow \Box_B(\neg[p!_B]\mathsf{ff} \rightarrow p)) \\
\text{(preservation of facts axiom)} & \vdash \neg[p \star p'_{A,B,C}]\mathsf{ff} \rightarrow \Box_A(\neg[p!_B]\mathsf{ff} \rightarrow \Box_B[p!_B]p) \\
\text{(rationality for } B \text{ wrt } p!_B) & \vdash \neg[p \star p'_{A,B,C}]\mathsf{ff} \rightarrow \Box_A[p!_B]\Box_B p \\
\text{(rationality for } A \text{ wrt } p \star p'_{A,B,C}) & \vdash [p \star p'_{A,B,C}]\Box_A\Box_B p
\end{array}$$

With the additional axioms, we obtain the following result:

Theorem 2 (Soundness and Completeness). *A formula holds in all appearance-update coalgebras if and only if it is derivable in the appearance-update logic.*

Proof. The proof of both soundness and completeness is detailed in [8]. Here we only sketch the completeness proof. This follows the same line as the completeness result for dynamic epistemic logic [3], and is based on a translation between our appearance-update logic (with appearance-update coalgebras as models) and ordinary epistemic logic (with F-coalgebras as models). As in [3], this translation has the property that a formula ϕ is semantically equivalent to its translation ϕ^t. Moreover, the axioms and rules of appearance-update logic ensure that $\vdash \phi \leftrightarrow \phi^t$. These properties, together with our result in [8] that the

final F-coalgebra can be extended to an appearance-update coalgebra, allow us to make use of the completeness result of [6] for F-coalgebras in order to prove completeness of appearance-update logic w.r.t. appearance-update coalgebras.

Acknowledgement. We would like to thank the anonymous referees for valuable suggestions on improving the paper.

References

1. Baltag, A.: A coalgebraic semantics for epistemic programs. In: Proceedings of Coalgebraic Methods in Computer Science. Electronic Notes in Theoretical Computer Science, vol. 82 (2003)
2. Baltag, A., Coecke, B., Sadrzadeh, M.: Epistemic actions as resources. Journal of Logic and Computation, forthcoming
3. Baltag, A., Moss, L.S.: Logics for epistemic programs. Synthese 139 (2004)
4. van Benthem, J., Pacuit, E.: The tree of knowledge in action: towards a common perspective. In: Proceedings of Advances in Modal Logic (2006)
5. Cîrstea, C.: A compositional approach to defining logics for coalgebras. Theoretical Computer Science 327(1), 45–69 (2004)
6. Cîrstea, C., Pattinson, D.: Modular construction of modal logics. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 258–275. Springer, Heidelberg (2004)
7. Cîrstea, C.: On expressivity and compositionality in logics for coalgebras. In: Proceedings of Coalgebraic Methods in Computer Science, Electronic Notes in Theoretical Computer Science 82 (2003)
8. Cîrstea, C., Sadrzadeh, M.: Coalgebraic epistemic update without change of model `http://ecs.soton.ac.uk/~ms6/TechRep.pdf`
9. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge (1995)
10. Gerbrandy, J.: Bisimulations on Planet Kripke. Ph. D. Thesis, University of Amsterdam (1999)
11. Jacobs, B.: The temporal logic of coalgebras via Galois algebras. Mathematical Structures in Computer Science 12, 875–903 (2002)
12. Jacobs, B.: Many-sorted coalgebraic modal logic: a model-theoretic study. Theoretical Informatics and Applications 35, 31–59 (2001)
13. Moss, L.S.: Coalgebraic logic. Annals of Pure and Applied Logic 96, 241–259 (1999)
14. Sadrzadeh, M.: Actions and Resources in Epistemic Logic. Ph.D. Thesis, University of Quebec at Montreal (2005), `http://www.ecs.soton.ac.uk/~ms6/all.pdf`
15. Sadrzadeh, M., Cîrstea, C.: Relating algebraic and coalgebraic logics of knowledge and update. In: Proceedings of the 7th conference on Logic and the Foundations of Game and Decision Theory, pp. 199–208, Liverpool (July 2006)
16. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249, 3–80 (2000)

The Maude Formal Tool Environment

Manuel Clavel[1], Francisco Durán[2], Joe Hendrix[3], Salvador Lucas[4], José Meseguer[3], and Peter Ölveczky[5]

[1] Universidad Complutense de Madrid, Spain
[2] Universidad de Málaga, Spain
[3] University of Illinois at Urbana-Champaign, IL, USA
[4] Universidad Politécnica de Valencia, Spain
[5] University of Oslo, Norway

Abstract. This paper describes the main features of several tools concerned with the analysis of either Maude specifications, or of extensions of such specifications: the ITP, MTT, CRC, ChC, and SCC tools, and Real-Time Maude for real-time systems. These tools, together with Maude itself and its searching and model-checking capabilities, constitute Maude's formal environment.

1 Introduction

Maude is a language and a system based on rewriting logic [1,2]. Maude modules are rewrite theories, while computation with such modules corresponds to efficient deduction by rewriting. Because of its logical basis and its initial model semantics, a Maude module defines a precise mathematical model. This means that Maude and its formal tool environment can be used in three, mutually reinforcing ways: as a declarative programming language, as an executable formal specification language, and as a formal verification system. The Maude system, its documentation, and related papers and applications are available from the Maude website `http://maude.cs.uiuc.edu`.

Besides the built-in support for verifying invariants and LTL formulas, the following tools are also available as part of the Maude formal environment: the Inductive Theorem Prover (ITP) can be used to verify inductive properties of functional modules (§2); the Maude Termination Tool (MTT) can be used to prove termination of functional modules (§3); the Church-Rosser Checker (CRC) can be used to check the Church-Rosser property of functional modules (§4); the Coherence Checker (ChC) can be used to check the coherence (or ground coherence) of unconditional system modules (§5); and the Sufficient Completeness Checker (SCC) can be used to check that defined functions have been fully defined in terms of constructors (§6). Furthermore, if we are dealing with rewriting logic specifications of real-time systems, we can use the Real-Time Maude tool (§7) to both simulate such specifications and to perform search and model-checking analysis of their LTL properties. Full Maude [2,9], an extension of Maude, written in Maude itself, has played a key role in the construction of some of these tools. Full Maude has become a common infrastructure on top of which tools like these can be built, but also environments for other languages, such as, e.g., the Real-Time Maude tool.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 173–178, 2007.
© Springer-Verlag Berlin Heidelberg 2007

In the following sections we summarize the main features of these tools. For further details on them, including user manuals, examples, and restrictions, please check the given references or visit the indicated web sites.

2 The ITP: An Inductive Theorem Prover

The Maude Inductive Theorem Prover tool (ITP) [4] is a theorem-proving *assistant*. It can be used to interactively verify inductive properties of membership equational specifications. An important feature of the ITP is that it supports proofs by structural induction and complete induction. Operations do not have to be completely specified before inductive properties about them can be verified mechanically.

The ITP is a Maude program. It comprises over 8000 lines of Maude code that make extensive use of the reflective capabilities of the system. In fact, rewriting-based proof simplification steps are directly executed by the underlying Maude rewriting engine. The ITP tool is currently available as a web-based application that includes a *module editor*, a *formula editor*, and a *command editor*. These editors allow users to create and modify specifications, to formalize properties about them, and to guide the proofs by filling in and submitting web forms. The web application also offers a goal viewer, a script viewer, and a log viewer. They generate web pages that allow the user to check, print, and save the current state of a proof, the commands that have guided it, and the logs generated in the process by the Maude system. The ITP web-based application can be accessed at `http://maude.sip.ucm.es:8080/webitp/`.

The ITP is still an experimental tool, but the results obtained so far are encouraging. It is the only theorem prover at present that supports reasoning about membership equational logic specifications. The integration of term rewriting with a decision procedure for linear arithmetic with uninterpreted function symbols [5] has been implemented in ITP by exploiting the reflective capabilities of Maude. The ease with which this integration has been accomplished encourages us to add other decision procedures to our tool in the near future. Another interesting extension of the tool is the implementation of the cover set induction method [22], a feature already available in RRL [15].

3 The Maude Termination Tool

Maude, as other equational and rule-based programming languages, has expressive features: advanced typing constructs such as sorts, subsorts, kinds, and memberships; matching modulo axioms; evaluation strategies; and very general conditional rules. Proving termination of programs having such features is nontrivial; furthermore, some of these features are not supported by standard termination methods and tools. Yet, the use of such features may be essential to ensure termination. The Maude Termination Tool (MTT) uses several theory transformations [7,8] to bridge the gap between expressive equational programs and conventional termination tools for (variants of) term rewriting systems, which are used as back-ends. Currently, MU-TERM [16] and AProVE [10] provide the most accurate termination proofs for MTT and they can be used as back-ends. Tools which implement less specific (but still valid) proofs like C*i*ME [6] can be used as well. The transformed termination problems are given to the

back-end tools in TPDB syntax [17]. This makes MTT extensible, so that new tools supporting such syntax can be added as back-ends.

The tool implementation distinguishes two parts: a reflective Maude specification implements the theory transformations described in [7,8], and a Java application connects Maude to the back-end termination tools and provides a graphical user interface. The Java application is in charge of sending the user-provided specification to Maude to perform the transformations. The resulting unsorted unconditional (context-sensitive) rewriting system obtained from such transformations is proved terminating by using the above-mentioned tools. To alleviate the installation requirements on external tools, the application includes support for connecting to the external tools remotely via different alternatives, including sockets and web services.

MTT is available from `http://www.lcc.uma.es/~duran/MTT`.

4 The Church-Rosser Checker

For order-sorted specifications, being Church-Rosser and terminating means not only confluence, but also a *sort decreasingness* property: each normal form has the least possible sort among those of all other equivalent terms. The Church-Rosser Checker (CRC) [3] is a tool that helps checking whether a Maude order-sorted conditional equational specification satisfies this Church-Rosser property.

A specification with an initial algebra semantics can often be ground Church-Rosser even though some of its critical pairs may not be joinable. That is, the specification can often be ground Church-Rosser without being Church-Rosser for terms with variables. The CRC can be used to check specifications with an initial algebra semantics that have already been proved terminating and now need to be checked to be ground Church-Rosser. If the specification cannot be shown to be ground Church-Rosser by the tool, proof obligations consisting of a set of critical pairs and a set of membership assertions that must be shown, respectively, ground-joinable, and ground-rewritable to a term with the required sort are generated and are given back to the user as a useful guide in the attempt to establish the ground Church-Rosser property. Since this property is in fact inductive, in some cases the ITP (§2) can be enlisted to prove some of these proof obligations. In other cases, the user may in fact have to modify the original specification by carefully considering the information conveyed by the proof obligations.

The tool is written in Maude, and is in fact an *executable specification* of the formal inference system that it implements. A complete execution environment for the tool has been integrated within Full Maude.

The tool, together with its documentation and some examples, is available from `http://www.lcc.uma.es/~duran/CRC`.

5 The Maude Coherence Checker

Coherence is a key executability requirement for rewrite theories. It allows reducing the, in general undecidable, problem of computing rewrites of the form $[t]_{E\cup A} \longrightarrow [t']_{E\cup A}$, with A a set of equational attributes (associativity, commutativity, identity) for which

matching algorithms exist and E a set of equations, to the much simpler and decidable problem of computing rewrites of the form $[t]_A \longrightarrow [t']_A$.

The Maude Coherence Checker (ChC), which is written in Maude using a reflective design as an extension of Full Maude, provides a *coherence decision procedure* for order-sorted system modules whose equations and rules are unconditional. The tool generates a set of critical pairs, whose coherence guarantees that of the entire system module. It then checks whether each of these pairs is coherent. The equational part of the system module given as input to the tool is always assumed to be ground Church-Rosser and terminating. The CRC (§4) and the MTT (§3) can be used to try to prove such properties.

For Maude system modules, which have an initial model semantics, the weaker requirement of *ground coherence*, that is, coherence for ground terms, is enough. When the ChC tool cannot prove coherence—either because this fails, or because the input specification falls outside the class of decidable theories—it outputs a set of *proof obligations* associated with the critical pairs that it could not prove coherent. The user can then interact with the ChC tool to try to prove ground coherence by a constructor-based process of reasoning by cases. In the end, either: (1) all proof obligations are discharged and the module is shown to be ground coherent; or (2) proving ground coherence can be reduced to proving that the inductive validity of a set of equations follows from the equational part of the input system module, for which the ITP can be used (§2); or (3) it is not possible to reduce some of the proof obligations to inductively proving some equations. Case (3) may be a clear indication that the specification is not ground coherent, so that a new specification should be developed.

The ChC tool, together with its documentation and some examples, is available from `http://www.lcc.uma.es/~duran/ChC`.

6 The Sufficient Completeness Checker

The Maude Sufficient Completeness Checker (SCC) [12] is a tool for checking that each operation in an equational Maude specification is defined on all valid inputs. The SCC verifies that the constructor operator declarations are annotated with the `ctor` attribute, and that enough equations have been given so that the remaining operations reduce to constructor terms. Specifications may import any of the built-in Maude modules. The tool uses the characterization of sufficient completeness given in [13] which allows for operations to be intentionally partial by declaring these operations at the kind level rather than at the sort level.

The tool is designed for unparameterized, order-sorted, left-linear, and unconditional Maude specifications that are ground terminating and Church-Rosser. It is a decision procedure for this class when every associative symbol is commutative, but for associative symbols that are not commutative it uses an algorithm from [14] based on machine learning techniques that works well in practice. If the specification is not sufficiently complete, the SCC returns a counterexample to aid the user in identifying errors. The tool is not complete for specifications with non-linear or conditional axioms, but nevertheless has proven useful in identifying errors in such specifications.

The SCC accepts interactive commands to check the sufficient completeness of a Maude module, and internally constructs a *propositional tree automaton* [14] whose language is empty iff the Maude module is sufficiently complete. The emptiness check is performed by a C++ tree automata library named CETA. Recently, the tool has been extended to check several important completeness problems of context-sensitive specifications [11].

SCC is available from its website at `http://maude.cs.uiuc.edu/tools/scc`.

7 The Real-Time Maude Tool

The Real-Time Maude tool [19] extends Maude to support the formal specification and analysis of real-time systems. The system's state space and its *instantaneous* transitions are defined, as in Maude, by, respectively, a membership equational logic theory and a set of rewrite rules. Time elapse is modeled by *tick rewrite rules* of the form `{`t`} => {`t'`} in time` u `if` *cond*, where `{_}` is an operator that encloses the state.

Real-Time Maude extends Maude's efficient rewriting, search, and LTL model-checking capabilities to the timed setting by: (i) analyzing behaviors up to a given time duration; and (ii) by having a *time sampling* treatment of *dense* time, in which only some moments in time are visited. Real-Time Maude is implemented in Maude, and achieves high performance by *simultaneously* transforming a real-time module *and* a query into a semantically equivalent Maude rewrite theory and a Maude query.

Real-Time Maude has proved useful to model real-time systems in an object-oriented way. In particular, the ease and flexibility with which an appropriate form of communication can be defined has been exploited in state-of-the-art applications including: (i) The AER/NCA protocol suite for multicast in active networks [20], where Real-Time Maude analysis was able to find all known bugs in AER/NCA, as well as some previously unknown bugs not discovered by traditional testing and simulation; and (ii) the OGDC wireless sensor network algorithm [21], where Real-Time Maude simulations provide more reliable estimates of the *performance* of OGDC than the simulation tool used by the OGDC developers.

At the theoretical level, we have given simple and easily checkable conditions for object-oriented specifications that ensure that Real-Time Maude analyses are sound and complete also for dense time [18].

The Real-Time Maude tool is available, together with its documentation and several case studies, from `http://www.ifi.uio.no/RealTimeMaude`.

Acknowledgements. The authors would like to thank Narciso Martí-Oliet for suggesting us to write this paper, and for his insightful comments and very constructive suggestions. F. Durán has been supported by Spanish Research Project TIN2005-09405-C02-01. J. Hendrix was supported by ONR Grant N00014-02-1-0715. S. Lucas was supported by the EU (FEDER) and the Spanish MEC, under grants TIN 2004-7943-C04-02 and HA 2006-0007, and by the Generalitat Valenciana under grant GV06/285. P. Ölveczky was supported by the Research Council of Norway.

References

1. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.: Maude: specification and programming in rewriting logic. Th. Comp. Sci. 285(2), 187–243 (2002)
2. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: All About Maude, A High-Performance Logical Framework, vol. 4350 of LNCS (to appear)
3. Clavel, M., Durán, F., Eker, S., Meseguer, J.: Building equational proving tools by reflection in rewriting logic. In: CAFE: An Industrial-Strength Alg. Formal Method, Elsevier, Amsterdam (2000)
4. Clavel, M., Palomino, M., Riesco, A.: Introducing the ITP tool: a tutorial. J. of Universal Computer Science 12(11), 1618–1650 (2007)
5. Clavel, M., Palomino, M., Santa-Cruz, J.: Integrating decision procedures in reflective rewriting-based theorem provers. In: Antoy, S., Toyama, Y. (eds.) Procs. WRS'04.
6. Contejean, E., Marché, C., Monate, B., Urbain, X.: Proving termination of rewriting with C*i*ME. In: Rubio, A. (ed.) Procs. of WST'03, pp. 71–73 (2003)
7. Durán, F., Lucas, S., Meseguer, J., Marché, C., Urbain, X.: Proving termination of membership equational programs. In: Sestoft, P., Heintze, N. (eds.) Procs. PEPM'04.
8. Durán, F., Lucas, S., Meseguer, J., Marché, C., Urbain, X.: Proving operational termination of membership equational programs. Higher-Order and Symb. Comp. (to appear)
9. Durán, F., Meseguer, J.: Maude's Module Algebra. Science of Computer Programming 66(2), 125–153 (2007)
10. Giesl, J., Schneider-Kamp, P., Thiemann, R.: AProVE 1.2: Automatic termination proofs in the dependency pair framework. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 281–286. Springer, Heidelberg (2006)
11. Hendrix, J., Meseguer, J.: On the completeness of context-sensitive order-sorted specifications. Tech. Report UIUCDCS-R-2007-2812, U. of Illinois (2007)
12. Hendrix, J., Meseguer, J., Ohsaki, H.: A sufficient completeness checker for linear order-sorted specifications modulo axioms. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 151–155. Springer, Heidelberg (2006)
13. Hendrix, J., Ohsaki, H., Meseguer, J.: Sufficient completeness checking with propositional tree automata. Tech. Report UIUCDCS-R-2005-2635, U. of Illinois (2005)
14. Hendrix, J., Ohsaki, H., Viswanathan, M.: Propositional tree automata. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 50–65. Springer, Heidelberg (2006)
15. Kapur, D., Zhang, H.: An overview of rewrite rule laboratory (RRL). J. Computer and Mathematics with Applications 29(2), 91–114 (1995)
16. Lucas, S.: MU-TERM: A tool for proving termination of context-sensitive rewriting. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 200–209. Springer, Heidelberg (2004)
17. Marché, C., Rubio, A., Zantema, H.: The Termination Problems Data Base: format of input files (March 2005) Available at `http://www.lri.fr/~marche/tpdb/`
18. Ölveczky, P.C., Meseguer, J.: Abstraction and completeness for Real-Time Maude. In: Procs. WRLA'06 (2006)
19. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symb. Comp. 20(1/2), 161–196 (2007)
20. Ölveczky, P.C., Meseguer, J., Talcott, C.L.: Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. Formal Methods in System Design 29, 253–293 (2006)
21. Ölveczky, P.C., Thorvaldsen, S.: Formal modeling and analysis of the OGDC wireless sensor network algorithm in Real-Time Maude. In: FMOOD'07 (to appear)
22. Zhang, H., Kapur, D., Krishnamoorthy, M.S.: A mechanizable induction principle for equational specifications. In: Lusk, E., Overbeek, R. (eds.) 9th International Conference on Automated Deduction. LNCS, vol. 310, pp. 162–181. Springer, Heidelberg (1988)

Bifinite Chu Spaces

Manfred Droste[1] and Guo-Qiang Zhang[2,*]

[1] Institute of Computer Science
Leipzig University, 04158 Leipzig, Germany
[2] Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, Ohio 44106, U.S.A.
gq@case.edu

Abstract. This paper studies colimits of sequences of finite Chu spaces and their ramifications. We consider three base categories of Chu spaces: the generic Chu spaces (**C**), the extensional Chu spaces (**E**), and the biextensional Chu spaces (**B**). The main results are: (1) a characterization of monics in each of the three categories; (2) existence (or the lack thereof) of colimits and a characterization of finite objects in each of the corresponding categories using monomorphisms/injections (denoted as **iC**, **iE**, and **iB**, respectively); (3) a formulation of bifinite Chu spaces with respect to **iC**; (4) the existence of universal, homogeneous Chu spaces in this category. Unanticipated results driving this development include the fact that: (a) in **C**, a morphism (f, g) is monic iff f is injective and g is surjective while for **E** and **B**, (f, g) is monic iff f is injective (but g is not necessarily surjective); (b) while colimits always exist in **iE**, it is not the case for **iC** and **iB**; (c) not all finite Chu spaces (considered set-theoretically) are finite objects in their categories. This study opens up opportunities for further investigations into recursively defined Chu spaces, as well as constructive models of linear logic.

1 Introduction

Within semantic frameworks for programming languages, a basic approach to the study of infinite objects is through their finite approximations. This is true both within an individual domain, as well as with domains collectively. A salient example of the latter is Plotkin's approach to SFP [13], where a class of domains is constructed systematically by taking colimits of sequences of finite partial orders. An important component of this framework is the notion of embedding-projection pair, capturing when one partial order is an approximation of another. An interesting outcome of this process is that completeness of an individual domain, the property that makes a cpo complete, becomes a natural by-product obtained by taking colimits of finite structures. In domain theory, the SFP- (or bifinite) domains now form an important cartesian closed category of domains, see [1].

* Corresponding author.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 179–193, 2007.
© Springer-Verlag Berlin Heidelberg 2007

In this paper we study colimits of sequences of finite Chu spaces. This entails the use of monic morphisms (or monomorphisms) as a way to formulate the substructure relationship. Such an innocuous attempt led to striking differentiations of the notion dictated by the extensionality properties of the underlying spaces. We consider three base categories of Chu spaces: the generic Chu spaces (**C**), the extensional Chu spaces (**E**), and the biextensional Chu spaces (**B**). The main results are: (1) a characterization of monics in each of the three categories; (2) existence (or the lack thereof) of colimits of countable sequences and a characterization of finite objects in each of the corresponding categories using monomorphisms/injections (denoted as **iC**, **iE**, and **iB**, respectively); (3) a formulation of bifinite Chu spaces with respect to **iC**; (4) the existence of universal, homogeneous Chu spaces in this category. Unanticipated results driving this development include the fact that: (a) in **C**, a morphism (f, g) is monic iff f is injective and g is surjective while for **E** and **B**, (f, g) is monic iff f is injective (but g is not necessarily surjective); (b) while colimits always exist in **iE**, it is not the case for **iC** and **iB**; (c) not all finite Chu spaces (considered set-theoretically) are finite objects in their categories.

Bifinite Chu spaces can be viewed, in an intuitive category-theoretic sense, as "countable" objects which are approximable by the finite objects of the category. The class of bifinite Chu spaces is very rich (up to isomorphism, there are uncountably many such spaces). However, we show that there is a single bifinite Chu space U which contains any other bifinite Chu space as a subspace. Moreover, U can be chosen to be homogeneous, i.e. to bear maximal possible degree of symmetry, and with this additional property U is unique up to isomorphism.

Our interest in Chu spaces stems from a number of recent developments. Chu spaces provide a suitable model of linear logic, originating from a general categorical construction introduced by Barr and his student [2,3]. The rich mathematical content of Chu spaces has been extensively illustrated by Pratt and his collaborators in a variety of settings, ranging from concurrency to logic and category theory [16,17,18,19,21,22]. In particular, Pratt shows that all small categories can be embedded in Chu(Set, 2) [20].

Chu spaces are closely related to the topic of Formal Concept Analysis (FCA [8,25]). Both areas use the same objects but the morphisms considered in FCA are different. Chu spaces are also related to domains [25]. In [11], a class called casuistries was introduced as a "continuous" version of Chu spaces, and yet maintaining the constructions desired as a model of linear logic. On the other hand, if instead of Chu transformations, Chu spaces are equipped with what are called *approximable mappings* [9,24], one obtains a cartesian closed category equivalent to the category of algebraic lattices and Scott continuous functions [10]. For this to work properly, a modified notion of formal concept, called *approximable concept*, needs to be used [26]. This way, an infinite concept can be approximated by finite ones.

Universal objects have played an important role in the development of domain theory. For example, the early work of Scott [23] and Plotkin [14] showed that with universal objects, domain equations can be treated by a calculus of retracts.

By studying Chu spaces that are colimits of sequences of finite objects, we hope to understand these spaces from a constructive angle, formulate a notion of completeness, and study the existence of universal, homogeneous objects. Recursively defined Chu spaces as well as models of linear logic within bifinite Chu spaces are some topics worth revisiting in light of this paper.

The rest of the paper is organized as follows. Section 2 recalls basic terminologies and gives a characterization of monic morphisms in the categories **C**, **E**, and **B**. Section 3 studies colimits in the categories **iC**, **iE**, and **iB**. Section 4 characterizes finite objects in **iC**, **iE**, and **iB**. Section 5 introduces bifinite Chu spaces and shows the existence of universal, homogeneous bifinite Chu spaces using finite amalgamation. Lengthier proofs will appear in the full paper.

2 Chu Spaces and Monic Morphisms

We recall some basic definitions to fix notation, following [4]. Readers interested in more details should consult [17]. In [17], the morphisms are called Chu transformations.

Definition 1. *A Chu space over a set Σ is a triple (A, r, X) where A is a set whose elements can be considered as* objects *and X is a set whose elements can be regarded as* attributes*. The satisfaction relation r is a function $A \times X \to \Sigma$. A morphism from a Chu space (A, r, X) to a Chu space (B, s, Y) is a pair of functions (f, g), with $f : A \to B$ and $g : Y \to X$ such that for any $a \in A$ and $y \in Y$, $s(f(a), y) = r(a, g(y))$. To alleviate the notational burden, we refer to a morphism by $\varphi = (f, g)$, and refer to the forward component by $\varphi^+ = f$ and the backward component by $\varphi^- = g$.*

For all the examples we consider in this paper, $\Sigma = \{0, 1\}$. If Σ is left unspecified, then it is assumed to contain at least two elements, denoted as 0 and 1. A Chu space (A, r, X) has two equivalence relations built-in. One is on the rows, where the a-th row corresponds to a function $r(a, -) : A \to \Sigma$. Two rows a, b are *equivalent* if $r(a, -) = r(b, -)$. Similarly, an equivalence relation exists on columns, defined by equality $r(-, x) = r(-, y)$ for $x, y \in X$. A Chu space (A, r, X) is called *extensional* if $r(-, x) = r(-, y)$ implies $x = y$, i.e., r does not contain repeated columns. Similarly, a Chu space (A, r, X) is *separable* if it does not contain repeated rows. In topological analogy, if we think of objects in A as points and attributes in X as open sets, then separable Chu spaces are those for which distinct points can be differentiated by the open sets containing them (such spaces are called T_0). A Chu space is *biextensional* if it is both separable and extensional.

We denote by **C** the category of Chu spaces and morphisms defined above, and **E** and **B** the full subcategories of extensional and biextensional Chu spaces, respectively. Composition of morphisms reduces to functional compositions of the components: $\varphi_1 \circ \varphi_2 = (\varphi_1^+ \circ \varphi_2^+, {\varphi_2}^- \circ {\varphi_1}^-)$, noting that the second component goes backwards. For abbreviation, objects are denoted as C_i for short, where $\mathsf{C}_i := (A_i, r_i, X_i)$. We refer to A_i the *object set*, and X_i the *attribute set* of

C_i, respectively. As a refinement of an observation in [11], we have the following result which will be useful for subsequent developments of the paper.

Proposition 1. *Suppose* $\varphi_1, \varphi_2 : \mathsf{C} \to \mathsf{C}'$ *are morphisms in* $\mathbf{C}$*. Then*

1. *if* C *is extensional, then* $\varphi_1^+ = \varphi_2^+$ *implies* $\varphi_1^- = \varphi_2^-$*;*
2. *if* C' *is separable, then* $\varphi_1^- = \varphi_2^-$ *implies* $\varphi_1^+ = \varphi_2^+$*;*
3. *if* C *and* C' *are biextensional, then* $\varphi_1^- = \varphi_2^-$ *iff* $\varphi_1^+ = \varphi_2^+$*.*

Thus, the forward and backward components in a morphism determine each other uniquely in the category of biextensional Chu spaces.

Proof. Let us write $\mathsf{C} = (A, r, X)$ and $\mathsf{C}' = (A', r', X')$. First we show (1). Suppose C is extensional and $\varphi_1^+ = \varphi_2^+$. Then for all $x' \in X'$ and $a \in A$ we have

$$r(a, \varphi_1^-(x')) = r'(\varphi_1^+(a), x') = r'(\varphi_2^+(a), x') = r(a, \varphi_2^-(x'))$$

Hence $\varphi_1^-(x') = \varphi_2^-(x')$ by extensionality of C. Now (2) follows from (1) by duality, and (1) and (2) imply (3).

As a first order of business, we consider monic morphisms, which capture the notion of a "substructure". In categorical terms, a morphism $\varphi : \mathsf{C}_1 \to \mathsf{C}_2$ is *monic* (or mono) if for any other morphisms $\varphi_i : \mathsf{C}^* \to \mathsf{C}_1$ $(i = 1, 2)$ such that $\varphi \circ \varphi_1 = \varphi \circ \varphi_2$, we have $\varphi_1 = \varphi_2$.

Remark. To make a distinction in our reference to morphisms at different levels, we reserve the term monic, mono, epi, etc for Chu spaces, and use one-to-one, onto, injective, surjective for the functions on the underlying sets. When properties on the underlying functions carry over to Chu spaces, we occasionally mix the terms.

Proposition 2. *We have*

1. *A morphism* $\varphi : \mathsf{C} \to \mathsf{C}'$ *in* $\mathbf{C}$ *is monic iff* φ^+ *is injective and* φ^- *is surjective.*
2. *A morphism* $\varphi : \mathsf{C} \to \mathsf{C}'$ *in* $\mathbf{E}$ *is monic iff* φ^+ *is injective.*
3. *A morphism* $\varphi : \mathsf{C} \to \mathsf{C}'$ *in* $\mathbf{B}$ *is monic iff* φ^+ *is injective.*
4. *Suppose* $\varphi : (A, r, X) \to (B, s, Y)$ *is a morphism in* $\mathbf{C}$ *and* (B, s, Y) *is extensional. If* φ^+ *is surjective, then* φ^- *is injective.*

The second and third items above would not be so surprising if the injectivity of φ^+ implied the surjectivity of φ^- in $\mathbf{B}$ and $\mathbf{E}$. But this is not the case.

Example 1. Consider $\mathsf{C} := (\{a\}, r, \{x_1, x_2\})$, with $r(a, x_1) = 0$ and $r(a, x_2) = 1$; $\mathsf{C}' := (\{b\}, r', \{y\})$, with $r'(b, y) = 0$. Then the constraints $f(a) = b$ and $g(y) = x_1$ satisfy the property that $r'(f(a), y) = 0 = r(a, g(y))$ and the pair (f, g) gives rise to a morphism. Clearly $\mathsf{C}, \mathsf{C}' \in \mathbf{B}$ and f is injective, but g is not surjective. With respect to item (2) in the proposition, this means that the backward component of a monic morphism in $\mathbf{E}$ and $\mathbf{B}$ need not be surjective.

Remark. Using a similar proof, we can show that a morphism $\varphi : \mathsf{C} \to \mathsf{C}'$ is monic if and only if φ^- is surjective, in the category of separable Chu spaces. We omitted this statement in Prop. 2 because we do not consider the category of separable Chu spaces in the rest of the paper.

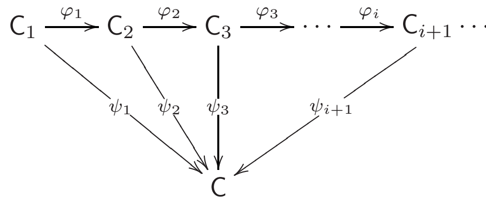
3 Colimits

We are interested in the subcategories of **C**, **E**, and **B** with monic morphisms, denoted as **iC**, **iE**, and **iB**, respectively. Let us begin with an unexpected observation that colimits do not exist in **iC** in general. For this purpose, we recall the definition of colimits, here formulated in **iC**, but it can easily be seen as an instantiation of a general notion [12] – we will only consider colimits of ω-sequences here. We then show that colimits do exist in **iE** and **iB**.

Definition 2. *An ω-sequence in* **iC** *is a family* $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$

$$\mathsf{C}_1 \xrightarrow{\varphi_1} \mathsf{C}_2 \xrightarrow{\varphi_2} \mathsf{C}_3 \xrightarrow{\varphi_3} \cdots \mathsf{C}_{i-1} \xrightarrow{\varphi_{i-1}} \mathsf{C}_i \xrightarrow{\varphi_i} \mathsf{C}_{i+1} \cdots$$

Definition 3. *A cone from an ω-sequence $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$ to a Chu space $\mathsf{C} := (A, r, X)$ is a family of mappings $\mathsf{C}_i \xrightarrow{\psi_i} \mathsf{C}$ such that $\psi_{i+1} \circ \varphi_i = \psi_i$, for all $i \geq 1$, i.e., the diagram*



commutes.

A cone $(\mathsf{C}_i \xrightarrow{\psi_i} \mathsf{C})_{i\geq 1}$ is universal if for any other cone $(\mathsf{C}_i \xrightarrow{\psi'_i} \mathsf{C}')_{i\geq 1}$ such that $\psi'_{i+1} \circ \varphi_i = \psi'_i$ for all $i \geq 1$, there exists a unique $\mathsf{C} \xrightarrow{\psi} \mathsf{C}'$ such that $\psi \circ \psi_i = \psi'_i$ for all $i \geq 1$. Such a universal cone, if exists, is called the colimit of the family $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$, while ψ is called the mediating map. In this case we write $\mathsf{C} = \operatorname{colim}_i(\mathsf{C}_i, \varphi_i)$.

Theorem 1. *Colimits do not always exist in* **iC***.*

Proof. Consider Chu spaces $\mathsf{C}_i := (\{1, \ldots, i\}, r_i, \{1, \ldots, i\})$, such that $r_i(a, x) = 1$ if $a \leq x$, and $r_i(a, x) = 0$ otherwise. Define $\varphi_i : \mathsf{C}_i \to \mathsf{C}_{i+1}$ such that $\varphi_i^+(a) = a$ for $a = 1, \ldots, i$, and $\varphi_i^-(i+1) = i$, but $\varphi_i^-(x) = x$ otherwise. Observe that C_i is biextensional. It is straightforward to verify that φ_is are indeed morphisms: for all $1 \leq a \leq i$ and $1 \leq x \leq i+1$, $r_{i+1}(\varphi_i^+(a), x) = 1$ iff $r_{i+1}(a, x) = 1$ iff $a \leq x$ iff $r_i(a, \varphi_i^-(x)) = 1$. Hence φ_i is monic.

Consider $\mathsf{C} := (\mathbb{N}, r, \mathbb{N})$, with $r(a, x) = 1$ iff $a \leq x$, and $r(a, x) = 0$ otherwise. Define $\psi_i : \mathsf{C}_i \to \mathsf{C}$ by letting ${\psi_i}^+$ be inclusions and ${\psi_i}^-(x) = x$ if $x \leq i$ and ${\psi_i}^-(x) = i$ for $x > i$. One readily checks that $(\psi_i : \mathsf{C}_i \to \mathsf{C})_{i\geq 1}$ is a cone. Consider another cone defined by $\mathsf{C}' := (\mathbb{N}, r', \mathbb{N} \cup \{t\})$, where r' extends r with $r'(a, t) = 1$ for all $a \in \mathbb{N}$. Define $\psi'_i : \mathsf{C}_i \to \mathsf{C}'$ by ${\psi'_i}^+ = {\psi_i}^+$ and letting ${\psi'_i}^-$ extend ψ_i^- with ${\psi'_i}^-(t) = i$. Clearly, $(\psi'_i : \mathsf{C}_i \to \mathsf{C}')_{i\geq 1}$ is also a cone.

Now we can infer that the colimit does not exist. More specifically, suppose $(\psi_i^* : \mathsf{C}_i \to \mathsf{C}^*)_{i\geq 1}$ with $\mathsf{C}^* = (A^*, r^*, X^*)$ were a colimit. First consider a mediating map $\psi' : \mathsf{C}^* \to \mathsf{C}'$. We have $\psi'^-(t) = x^*$ for some $x^* \in X^*$. Then for each $a^* \in A^*$ we obtain $r^*(a^*, x^*) = r^*(a^*, \psi'^-(t)) = r'(\psi'^+(a^*), t) = 1$, thus $r^*(-, x^*) = 1$. Next consider a mediating map $\psi : \mathsf{C}^* \to \mathsf{C}$. Since $\psi^- : \mathbb{N} \to X^*$ must be onto, $\psi^-(n) = x^*$ for some $n \in \mathbb{N}$. We obtain $0 = r(n+1, n) = r_{n+1}(n+1, n) = r_{n+1}(n+1, {\psi_{n+1}^*}^-(x^*)) = r^*({\psi_{n+1}^*}^+(n+1), x^*) = 1$, a contradiction.

Subsequently we will show that particular ω-sequences of Chu spaces do have colimits. For this we provide a generic construction. It is the standard construction in the category of sets, assimilated into the context of Chu spaces. We phrase it explicitely since we will often refer to it.

Construction 1. Let $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$ be an ω-sequence of Chu spaces where $\mathsf{C}_i = (A_i, r_i, X_i)$ and $\varphi_i^+ : A_i \to A_{i+1}$ is the inclusion mapping, for each $i \geq 1$. Consider $\mathsf{C} := (A, r, X)$ where

$$\begin{aligned} A &:= \textstyle\bigcup_{i\geq 1} A_i, \\ X &:= \{(x_j)_{j\geq 1} \mid \forall j \geq 1,\ x_j \in X_j \ \&\ \varphi_j^-(x_{j+1}) = x_j\}, \\ r(a, (x_j)_{j\geq 1}) &:= r_i(a, x_i) \text{ if } a \in A_i \ (i \geq 1). \end{aligned}$$

Subsequently, we will denote a sequence $(x_j)_{j\geq 1} \in X$ often by $\tilde{x}$.

For each $i \geq 1$, define $\mathsf{C}_i \xrightarrow{\psi_i} \mathsf{C}$ by $\psi_i^+(a) := a$ and $\psi_i^-(\tilde{x}) := x_i$ for all $a \in A_i$ and $\tilde{x} \in X$.

In Construction 1, observe that possibly $X = \emptyset$. Note that the relation r is well-defined since if $i \geq 1$ and $a \in A_i$, then $x_i = \varphi_i^-(x_{i+1})$ so $r_{i+1}(a, x_{i+1}) = r_i(a, x_i)$; inductively we obtain $r_j(a, x_j) = r_i(a, x_i)$ for each $j > i$.

Clearly, ψ_i is a morphism. Then we have, for each $\tilde{x} \in X$,

$$(\varphi_i^- \circ \psi_{i+1}^-)(\tilde{x}) = \varphi_i^-(x_{i+1}) = x_i = \psi_i^-(\tilde{x})$$

and for any $a \in A$,

$$(\psi_{i+1}^+ \circ \varphi_i^+)(a) = a = \psi_i^+(a)$$

Therefore, $\psi_{i+1} \circ \varphi_i = \psi_i$, and $(\mathsf{C}_i \xrightarrow{\psi_i} \mathsf{C})_{i\geq 1}$ is indeed a cone. We note:

Proposition 3. *If an ω-sequence $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$ in* **iC** *has a colimit, then this colimit is provided, up to isomorphism, by the cone $(\mathsf{C}_i \xrightarrow{\psi_i} \mathsf{C})_{i\geq 1}$ of Construction 1.*

Proof. Let $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$ have a colimit $(\mathsf{C}_i \xrightarrow{\psi'_i} \mathsf{C}')_{i\geq 1}$ in **iC** where $\mathsf{C}' = (A', r', X')$. By Proposition 2(1), the mappings φ_i^+ are injective and the mappings φ_i^- are surjective, and we may assume the φ_i^+s to be inclusions. Now construct $\mathsf{C} = (A, r, X)$ and $\psi_i : \mathsf{C}_i \to \mathsf{C}$ $(i \geq 1)$. as in Construction 1. We claim that each ψ_i $(i \geq 1)$ is a morphism in **iC**. By Proposition 2(1), it remains to show that ψ_i^- is onto. Using that the φ_j^-s are onto, for any $x_i \in X_i$ we can easily find $\tilde{x} \in X$ with $x_i = \varphi_i^-(\tilde{x})$.

Since C' is the colimit, there is a unique $\psi : \mathsf{C}' \to \mathsf{C}$ in $\mathbf{iC}$ such that $\psi \circ \psi'_i = \psi_i$ for all $i \geq a$. Then $\psi^+ : A' \to A$ is injective. If $a \in A_i$ $(i \geq 1)$, then $a = \psi_i^+(a) = \psi^+ \circ {\psi_i^+}'(a)$ and $\psi'_i(a) \in A'$, so ψ^+ is onto. Further, $\psi^- : X \to X'$ is onto, and we claim that ψ^- is injective. Let $\tilde{x}, \tilde{y} \in X$ with $\psi^-(\tilde{x}) = \psi^-(\tilde{y})$. For each $i \geq 1$, then $x_i = \psi_i^-(\tilde{x}) = {\psi'_i}^- \circ \psi^-(\tilde{x}) = {\psi'_i}^- \circ \psi^-(\tilde{y}) = \psi_i^-(\tilde{y}) = y_i$, showing $\tilde{x} = \tilde{y}$. Hence ψ is an isomorphism.

In contrast to Theorem 1, we have the following.

Theorem 2. *Colimits exist in* $\mathbf{iE}$*, as given by Construction 1.*

Proof. Let $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$ be an ω-sequence in $\mathbf{iE}$ where $\mathsf{C}_i = (A_i, r_i, X_i)$ for each $i \geq 1$. By Proposition 2(2), the mappings φ_i^+ are injective, and we may assume the φ_i^+s to be inclusions. Now construct $\mathsf{C} = (A, r, X)$ and $\psi_i : \mathsf{C}_i \to \mathsf{C}$ $(i \geq 1)$ as in Construction 1. We claim that C is extensional. Let $\tilde{x}, \tilde{y} \in X$ and assume that $r(-, \tilde{x}) = r(-, \tilde{(y)})$. We need to show that $\tilde{x} = \tilde{y}$. Indeed, let $i \geq 1$ and choose any $a \in A_i$. Then $r_i(a, x_i) = r(a, \tilde{x}) = r(a, \tilde{y}) = r_i(a, y_i)$. So $r_i(-, x_i) = r_i(-, y_i)$ and thus $x_i = y_i$ as C_i is extensional. Hence $\tilde{x} = \tilde{y}$, and C is extensional.

For universality, let $(\mathsf{C}_i \xrightarrow{\psi'_i} \mathsf{C}')_{i\geq 1}$ be a cone, where $\mathsf{C}' = (A', r', X')$. Define $\psi : \mathsf{C} \to \mathsf{C}'$ by letting $\psi^+ : A \to A'$ be such that $\psi^+(a) := {\psi'}_i^+(a)$ if $a \in A_i$, and letting $\psi^- : X' \to X$ be given as $\psi^-(x') := ({\psi'}_m^-(x'))_{m\geq 1}$. Then ψ^+ is well-defined because for any $1 \leq i < j$, ${\psi'}_j^+(a) = {\psi'}_i^+(a)$; also ψ^- is well-defined because for any $j \geq 1$, $\varphi_j^-({\psi'}_{j+1}^-(x')) = {\psi'}_j^-(x')$, and hence the sequence $({\psi'}_m^-(x'))_{m\geq 1}$ belongs to X. Further, ψ is a morphism. We have $\psi \circ \psi_i = {\psi'}_i$ for all $i \geq 1$ because $\psi^+(\psi_i^+(a)) = \psi^+(a) = {\psi'}_i(a)$, and $\psi_i^-(\psi^-(x')) = \psi_i^-(({\psi'}_j^-(x'))_{j\geq 1}) = {\psi'}_i^-(x')$ for all $a \in A_i$ and $x' \in X'$, by definitions.

The mediating morphism ψ is a morphism in $\mathbf{iE}$ because ψ^+ is injective, and by Prop. 2(2), it is monic. The mediating morphism is unique because its values are fixed by the commutativity requirements of the colimit diagram.

We now consider the biextensional case. In order to avoid potential confusion of terminology, we call a Chu space $\mathsf{C} := (A, r, X)$ with finite A and X a finite Chu structure. Finite objects in categorical terms will be studied in the next section, as we will learn that finite objects and finite Chu structures do not always agree.

Theorem 3. *Colimits exist in* $\mathbf{iB}$ *for sequences of finite Chu structures. They do not exist in general in* $\mathbf{iB}$*. If for an* ω*-sequence in* $\mathbf{iB}$ *there is a cone to some Chu space in* $\mathbf{iB}$*, then the sequence has a colimit in* $\mathbf{iB}$*.*

The proof of this result, contained in the full paper, involves a typical König's lemma–argument for finite Chu structures which we will encounter again later.

It is informative to think about the example given in the proof of Theorem 1. By the Construction 1, $\mathsf{C}' = (\mathbb{N}, r', \mathbb{N} \cup \{t\})$ is the colimit both in $\mathbf{iE}$ and in $\mathbf{iB}$. There is indeed a monic morphism ψ from C' to $\mathsf{C} = (\mathbb{N}, r, \mathbb{N})$, where ψ^+ is the identity (injection), and ψ^- is the inclusion (but not onto).

Also note that even though Theorem 2 confirms that colimit always exists for extensional Chu spaces, the counterexample for Theorem 3 shows that colimits

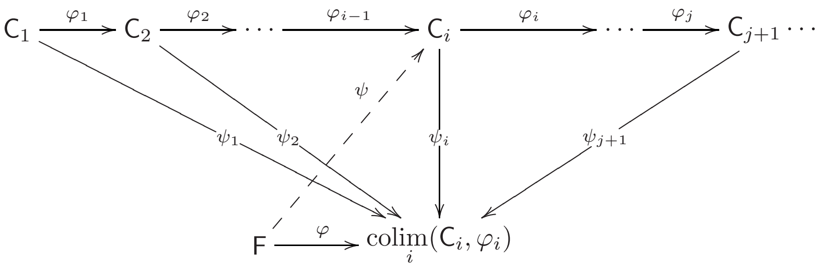
for infinite structures may have weird behaviors with unintended effects. This invites us to look more into objects constructed as colimits of finite structures, in the next sections.

4 Finite Objects

In studying patterns of approximation in Chu spaces, finite objects play an important role since they serve as the basis of approximation. In most cases, one expects finite objects to correspond to finite structures, objects whose constituents are finite sets. In categorical terms, finite objects are captured using colimits in a standard way, and the notion of "approximation" is captured by monic morphisms. Therefore, we work with categories **iC**, **iE**, and **iB**. However, since Prop. 2 indicates that what counts as monic morphisms depends on extensionality, the existence of colimits and the characterization of finite objects are not straightforward set-theoretic generalizations obtained by treating each component of Chu spaces separately.

We give a characterization of the finite objects of **iC**. Surprisingly, not all finite structures in **iC** are finite objects; finite objects are characterized as extensional structures with finite object set instead. The following definition is phrased in **iC**; but as a general categorical concept it can be made explicit in **iE** and **iB** as well, and we do not repeat this here.

Definition 4. *An object* F *of* **iC** *is finite if for every* ω*-sequence* $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$ *of Chu spaces having a colimit, for every morphism* $\varphi : \mathsf{F} \to \operatorname{colim}_i(\mathsf{C}_i, \varphi_i)$ *in* **iC** *there exist* $i \geq 1$ *and a morphism* $\psi : \mathsf{F} \to \mathsf{C}_i$ *such that the diagram*



commutes, i.e., $\psi : \mathsf{F} \to \mathsf{C}_i$ *is such that* $\varphi = \psi_i \circ \psi$*.*

If Σ is finite and $\mathsf{F} = (A, r, X)$ is a finite object in **iC**, one can show that both A and X are finite sets, i.e. F is a finite Chu space. However, somewhat surprisingly (at least to us), the converse does not hold, as already simple examples show, see Example 2 below. The following result characterizes the finite objects of **iC**.

Theorem 4. *An object* $\mathsf{F} = (B, s, Y)$ *is finite in* **iC** *iff* B *is finite and* F *is extensional. In this case,* $|Y| \leq |\Sigma|^{|B|}$*; in particular, if* Σ *is finite, so is* Y*.*

A proof will be given in the full paper.

Next we give two examples to illustrate Theorem 4.

Example 2. Let $\Sigma = \{0,1\}$ and $\mathsf{F} := (\{\star\}, r, \{1,2\})$, a finite Chu space. If $r(\star,1) = r(\star,2) = 0$, then F is not extensional and thus, by Theorem 4, not a finite object of **iC**. To see this more explicitly, one can construct a sequence $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$ as in the proof of Theorem 4, with $Y' = \emptyset$.

Example 3. Only in this example, let Σ be an arbitrary (possibly infinite) set, and let $\mathsf{F} = (\{\star\}, r, \Sigma)$ with $r(\star, \sigma) = \sigma$ for each $\sigma \in \Sigma$. Then F is extensional, and by Theorem 4, F is a finite object of **iC**. Trivially, if Σ is infinite, F is not a finite Chu space.

Theorem 5. *In the category* **iE**, *if* $\mathsf{F} = (B, s, Y)$ *is finite then* B *is finite.*

An independent proof is needed even though we follow a similar path as the proof of Theorem 4. Not only should we make sure that the Chu spaces involved are all extensional, but also the monic morphisms are characterized differently. These entail non-trivial modifications from the proof of Theorem 4.

Proof. Suppose $\mathsf{F} = (B, s, Y)$ is a finite object in **iE**. Suppose B is infinite. Then we can write $B = A_1 \cup \{a_i \mid i \geq 1\}$, where $A_1 \cap \{a_i \mid i \geq 1\} = \emptyset$. Fix $c \in Y$. Let $\mathsf{C}_i := (A_1 \cup \{a_1, \ldots, a_i\}, r_i, \{X_i\})$, where $X_i = \{c\}$ and r_i is r restricted to the product $(A_1 \cup \{a_1, \ldots, a_i\}) \times \{c\}$. Clearly, all C_is are extensional. For morphisms $\varphi_i : \mathsf{C}_i \to \mathsf{C}_{i+1}$, define φ_i^+ as inclusions, and $\varphi_i^- : X_{i+1} \to X_i$ the identity. By Theorem 2, the colimit $(\mathsf{C}_i \xrightarrow{\psi_i} \mathsf{C})_{i\geq 1}$ with $\mathsf{C} = (A, r, X)$ of the sequence $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$ exists, and can be taken as the one given in Construction 1. Since each X_i is a singleton, X is a singleton as well. Thus we may assume $A = B$. With φ^+ identity and φ^- inclusion, we obtain a monic morphism φ from F to C. Hence there is a monic morphism ψ from F to some C_i which makes the required diagram commute. But then $\varphi^+(a_{i+1}) = \psi_i^+(\psi^+(a_{i+1})) \neq a_{i+1}$, a contradiction.

The converse of Theorem 5 is not true. To show this, we adapt the counterexample for the second part of Theorem 3 as follows. Let $\mathsf{C}_i := (\mathbb{N}, r_i, \mathbb{N} \cup \{c\})$, with $r_i(a, x) = 1$ iff $(x + i - 1) \mod a = 0$ for $a, x \in \mathbb{N}$, and $r_i(-, c) = 1$. Intuitively, r_i is obtained by starting from the countable identity matrix r_1 from the i-th column. The morphism $\varphi_i : \mathsf{C}_i \to \mathsf{C}_{i+1}$ is defined by $\varphi_i^+ := id_{\mathbb{N}}$, and $\varphi_i^-(x) = x + 1$, but we keep c constant. Then, the colimit of this sequence is $\mathsf{C} := (\mathbb{N}, r, \{c\})$, with $r(-, c) = 1$. Now let $\mathsf{B} := (\{1,2\}, s, \{c\})$, with inclusion and identity paired to form a morphism φ from B to C. There cannot be a morphism ψ from B to any C_i, because $\psi^- : (\mathbb{N} \cup \{c\}) \to \{c\}$ cannot be defined, simply because B's column contains two 1s, and each $r_i(2, i) = 0$. Hence B is a finite extensional Chu space and thus a finite object of **iC** but not of **iE**.

Definition 5. *A Chu space* (A, r, X) *over* Σ *is called* complete, *if for any mapping* $f : A \to \Sigma$ *there is* $x \in X$ *with* $f = r(-, x)$.

Theorem 6. *In the category* **iE**, $\mathsf{F} = (B, s, Y)$ *is finite iff* B *is finite and* F *is complete.*

A proof will be presented in the full paper. The following easy remark shows that the structure of complete extensional Chu spaces $\mathsf{C} = (A, r, X)$ is very restricted: it is completely determined, up to isomorphism, by the cardinality of the object set A.

Remark 1. Let $\mathsf{C} = (A, r, X)$ and $\mathsf{C}' = (A', r', X')$ be two complete extensional Chu spaces with $|A| = |A'|$. Then C and C' are isomorphic in **iC**.

Proof. Choose a bijection $\varphi^+ : A \to A'$. By the assumption on C, for each $x' \in X'$ there is a uniquely determined $x \in X$ with $r(-, x) = r'(-, x') \circ \varphi^+$. The mapping $\varphi^- : X' \to X$ with $\varphi^-(x') = x$ yields a Chu morphism $\varphi = (\varphi^+, \varphi^-)$, and φ^- is bijective by the assumption on C'.

Similar to Theorem 6, we have the following. However, an independent proof is needed because the structures used in the proof for Theorem 6 are not biextensional.

Theorem 7. *In the category* **iB**, $\mathsf{F} = (B, s, Y)$ *is finite iff* B *is finite and* F *is complete.*

Corollary 1. *Let* $\mathsf{C}, \mathsf{C}' \in \mathbf{C}$ *be such that* C *is extensional. Let* $\varphi : \mathsf{C} \to \mathsf{C}'$ *be a monic in* $\mathbf{C}$*, and assume that* C' *is a finite object in* $\mathbf{C}$*. Then* C *is also finite in* $\mathbf{C}$*. The analogous statements hold true, if* $\mathbf{C}$ *is replaced by* $\mathbf{E}$ *or* $\mathbf{B}$*, respectively.*

5 Bifinite Chu Spaces

In this section, we will investigate Chu spaces which are, intuitively and in a category-theoretic sense, countable objects and approximable by the finite objects in the category. That is, we will define bifinite Chu spaces as colimits of a sequence of (strongly) finite Chu spaces. We will then show that this subcategory of **iC** contains a universal homogeneous object.

Recall that the finite objects of **iC** may have an infinite attribute set, if Σ is infinite (cf. Example 3). For technical reasons (cf. the proofs of Theorem 8 and Proposition 5), we will need that the objects employed here have a finite and non-empty set of attributes. We will call a space F in **iC** *strongly finite*, if F is a finite object in **iC** and a finite Chu space with non-empty set of attributes. Clearly, if Σ is finite, the finite and the strongly finite objects of **iC** with non-empty sets of attributes coincide.

Definition 6. *A Chu space in* **iC** *is called bifinite if it is isomorphic to the colimit (with respect to* **iE***) of a chain of stronly finite objects in* **iC***. The corresponding full subcategory of bifinite Chu spaces of* $\mathbf{C}$ *and* **iC** *are denoted as* $\mathbf{C}_{\text{bif}}$ *and* $\mathbf{iC}_{\text{bif}}$*, respectively.*

As an example, consider the sequence of strongly finite biextensional spaces $(\mathsf{C}_i, \varphi_i)_{i \geq 1}$ described in the proof of Theorem 1. As shown there, this sequence has no colimit in the category **iC**. But by Theorem 2, the sequence has a colimit

with respect to the category **iE**. This colimit thus belongs to $\mathbf{iC}_{\text{bif}}$. Moreover, we will see below in Theorem 8, that this space is also a colimit of the given sequence with respect to the category $\mathbf{iC}_{\text{bif}}$.

Recall that any finite object of **iC** is extensional, hence any bifinite Chu space is also extensional. It would not be interesting to formulate the concept of bifinite spaces in **iE** or **iB**, i.e. as colimits of chains of finite objects of **iE** resp. **iB**: By Theorems 6 and 7, these finite objects are complete. One can show that colimits of chains of complete extensional objects are again complete and extensional. Hence any two such 'bifinite' objects (in **iE** or **iB**) with countably infinite object set are isomorphic by Remark 1. In contrast, we show that if $\mathbf{iC}_{\text{bif}}$ is very large:

Proposition 4. $\mathbf{iC}_{\text{bif}}$ *contains at least continuously many non-isomorphic objects.*

Proof. Consider a strictly increasing sequence of finite subsets $A_1 \subset A_2 \subset \ldots \subset \mathbb{N}$ of $\mathbb{N}$. We define a sequence $(\mathsf{C}_i, \varphi_i)_{i\geq 1}$ as follows. For each $i \geq 1$, let $\mathsf{C}_i = (A_i, r_i, X_i)$ with $X_i = \{1, \ldots, i\}$ and $r_i(a, j) = 1$ if $a \in A_j$ and $r_i(a, j) = 0$ otherwise, for any $a \in A_i$, $j \in X_i$. We let φ_i^+ be the inclusion mapping, $\varphi_i^-(j) = j$ if $1 \leq j \leq i$, and $\varphi_i^-(i+1) = i$. As colimit of this sequence of strongly finite objects we obtain, up to isomorphism, $(\mathsf{C}_i \xrightarrow{\psi_i} \mathsf{C})_{i\geq 1}$ with $\mathsf{C} = (\mathbb{N}, r, \mathbb{N} \cup \{\infty\})$, $r(a, j) = 1$ if $a \in A_j$ and $r(a, j) = 0$ otherwise, for any $a, j \in \mathbb{N}$, further $r(-, \infty) = 1$, and ψ_i^+ inclusion, $\psi_i^-(j) = j$ if $1 \leq j \leq i$ and $\psi_i^+(j) = i$ if $i < j \in \mathbb{N} \cup \{\infty\}$. Note that in C the set $\{a \in \mathbb{N} \mid r(a, i) = 1\}$ equals A_i if $i \in \mathbb{N}$, and $\mathbb{N}$ if $i = \infty$. Hence the bifinite space C constructed in this way determines the sequence of subsets $(A_i)_{i\geq 1}$ uniquely, and two different sequences give rise to non-isomorphic bifinite spaces. Since there are continously many such sequences, the result follows.

Remark 2. By cardinality arguments, one can show that up to isomorphism $\mathbf{iC}_{\text{bif}}$ has size $|\Sigma|^\omega$; this equals the continuum if Σ has size at most continuum.

With the restriction of objects to bifinite Chu spaces, colimits now exist, in contrast to Theorem 1.

Theorem 8. *Colimits exist in* $\mathbf{iC}_{\text{bif}}$.

The technical content of the result is that $\mathbf{iC}_{\text{bif}}$ is closed in **iC** and in **iE** with respect to taking colimits of sequences in $\mathbf{iC}_{\text{bif}}$, and these colimits taken in **iE** constitute the colimits of the given sequences with respect to $\mathbf{iC}_{\text{bif}}$.

To make the paper self-contained, we recall briefly a result of Droste and Göbel [6] concerning the existence of a universal, homogeneous object in an algebroidal category. Let **G** be a category in which all the morphisms are monic, and $\mathbf{G}^*$ a full subcategory of **G**. Individually, an object U of **G** is called

– $\mathbf{G}^*$-*universal* if for any object A in $\mathbf{G}^*$, there is a morphism $f : A \to U$;
– $\mathbf{G}^*$-*homogeneous* if for any A in $\mathbf{G}^*$ and any pair $f, g : A \to U$, there is an isomorphism $h : U \to U$ such that $f = h \circ g$;

Intuitively, $\mathbf{G}^*$-homogeneity means that each isomorphism between two $\mathbf{G}^*$-substructures of U extends to an automorphism of U; this means that U has maximal possible degree of symmetry.

Collectively, the category $\mathbf{G}^*$ is said to have the *amalgamation property* if for any $f_1 : A \to B_1$, $f_2 : A \to B_2$ in $\mathbf{G}^*$, there exist $g_1 : B_1 \to B$, $g_2 : B_2 \to B$ in $\mathbf{G}^*$ such that $g_1 \circ f_1 = g_2 \circ f_2$.

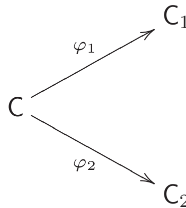
Definition 7. *Let* $\mathbf{G}$ *be a category in which all morphisms are monic. Then* $\mathbf{G}$ *is called* algebroidal*, if* $\mathbf{G}$ *has the following properties:*

1. $\mathbf{G}$ *has a weakly initial object,*
2. *Every object of* $\mathbf{G}$ *is a colimit of an* ω*-chain of finite objects,*
3. *Every* ω*-sequence of finite objects has a colimit, and*
4. *The number of (up to isomorphism) finite objects of* $\mathbf{G}$ *is countable and between any pair of finite objects there exist only countably many morphisms.*

Theorem 9. *(Droste and Göbel) Let* $\mathbf{G}$ *be an algebroidal category with all morphisms monic. Let* $\mathbf{G}_f$ *be the full subcategory of finite objects of* $\mathbf{G}$*. Then there exists a* $\mathbf{G}$*-universal,* $\mathbf{G}_f$*-homogeneous object iff* $\mathbf{G}_f$ *has the amalgamation property. Moreover, in this case the* $\mathbf{G}$*-universal,* $\mathbf{G}_f$*-homogeneous object is unique up to isomorphism.*

Proposition 5. *The category* $\mathbf{iC}_{\mathrm{bif}}$ *contains an initial object. The strongly finite objects of* $\mathbf{iC}$ *are precisely the finite objects of* $\mathbf{iC}_{\mathrm{bif}}$*. If* Σ *is countable, there are only countably many non-isomorphic finite objects in* $\mathbf{iC}_{\mathrm{bif}}$*. Between any pair of finite objects there are only finitely many injections. Moreover, the finite objects of* $\mathbf{iC}_{\mathrm{bif}}$ *have the amalgamation property.*

Proof. The space $(\emptyset, \emptyset, \{x\})$ is the initial object of $\mathbf{iC}_{\mathrm{bif}}$, since all spaces in $\mathbf{iC}_{\mathrm{bif}}$ have non-empty attribute sets. Next, we show only the amalgamation property; the rest is easy to see. Suppose $\mathsf{C} := (A, r, X)$, $\mathsf{C}_1 := (A_1, r_1, X_1)$, and $\mathsf{C}_2 := (A_2, r_2, X_2)$ are strongly finite objects in $\mathbf{iC}$ such that $A = A_1 \cap A_2$, and let $\varphi_1 : \mathsf{C} \to \mathsf{C}_1$ and $\varphi_2 : \mathsf{C} \to \mathsf{C}_2$ be morphisms in $\mathbf{iC}$.



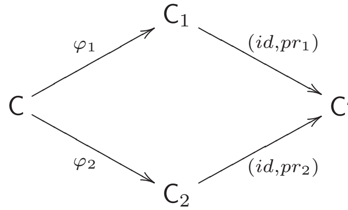
Construct $\mathsf{C}' := (A', r', X')$ as:

$$
\begin{aligned}
A' &= A_1 \cup A_2 \\
X' &= \{(x_1, x_2) \in X_i \times X_2 \mid \varphi_1^{-}(x_1) = \varphi_2^{-}(x_2)\} \\
r'(a, (x_1, x_2)) &= r_1(a, x_1) \quad \text{if } a \in A_1 \\
r'(a, (x_1, x_2)) &= r_2(a, x_2) \quad \text{if } a \in A_2.
\end{aligned}
$$

Note that in case $a \in A_1 \cap A_2$, we have

$$\begin{aligned} r_1(a, x_1) &= r(a, \varphi_1^-(x_1)) \\ &= r(a, \varphi_2^-(x_2)) \\ &= r_2(\varphi_2^+(a), x_2) \\ &= r_2(a, x_2). \end{aligned}$$

To see that C' is extensional, suppose $(x_1, x_2), (y_1, y_2) \in X'$ are such that $r'(a, (x_1, x_2)) = r'(a, (y_1, y_2))$ for all $a \in A_1 \cup A_2$. By the definition of C', then, for each $a \in A_1$, we have $r_1(a, x_1) = r_1(a, y_1)$. By the extensionality of C_1, we have $x_1 = y_1$. Similarly, by the extensionality of C_2, we have $x_2 = y_2$ and so $(x_1, x_2) = (y_1, y_2)$, as required.



It is easy to check further that $(id, pr_1) : \mathsf{C}_1 \to \mathsf{C}'$ and $(id, pr_2) : \mathsf{C}_2 \to \mathsf{C}'$ are morphisms in **iC**.

By Proposition 5, the following result immediately follows from Theorem 9.

Theorem 10. *Let Σ be countable. Then* $\mathbf{iC}_{\text{bif}}$ *is an algebroidal category containing a universal homogeneous object U. Moreover, U is unique up to isomorphism.*

Since $\mathbf{iC}_{\text{bif}}$ contains spaces with an attribute set of size continuum, it follows that the attribute set of U also has size continuum. However, we just note that since the proof of Theorem 9 is constructive, we can *construct a sequence* $(\mathsf{C}_i, \varphi_i)_{i \geq 1}$ whose colimit is the universal homogeneous object U.

We remark that $\mathbf{iC}_{\text{bif}}$ does not contain all countable extensional Chu spaces (just as not all countable cpos are SFP). Let $\mathsf{C} = (\mathbb{N}, r, \mathbb{N})$ be the biextensional Chu space described in the proof of Theorem 3.3. We claim that C is not bifinite.

Indeed, choose the sequence $(\mathsf{C}_i, \varphi_i)_{i \geq 1}$, the space $\mathsf{C}' = (\mathbb{N}, r', \mathbb{N} \cup \{t\})$ and the monics $\psi_i' : \mathsf{C}_i \to \mathsf{C}'$ as in the proof of Theorem 3.3. By Construction 1, $(\psi_i' : \mathsf{C}_i \to \mathsf{C}')_{i \geq 1}$ is the colimit of the chain $(\mathsf{C}_i, \varphi_i)_{i \geq 1}$ in **iE**. By Theorem 5.4, this is also the colimit of the sequence of finite spaces C_i in the category $\mathbf{iC}_{\text{bif}}$. Consider the morphisms $(\psi_i : \mathsf{C}_i \to \mathsf{C})_{i \geq 1}$ described in the proof of Theorem 3.3. Now if C was bifinite, there would be a unique morphism $\psi : \mathsf{C}' \to \mathsf{C}$ in $\mathbf{iC}_{\text{bif}}$ making the diagram commute. But then $\psi^+ = id_{\mathbb{N}}$, and $\psi^-(n) = t$ for some $n \in \mathbb{N}$, yielding $0 = r(n+1, n) = r'(n+1, t) = 1$, a contradiction.

6 Conclusion

Chu spaces are a general framework for studying the dualities of objects and properties; points and open sets; terms and types, under rich mathematical

contexts, with important connections to several sub-disciplines in computer science and mathematics. Traditionally, the study on Chu spaces had a "non-constructive" flavor. There was no framework in which to study constructions on Chu spaces with respect to their behavior in permitting a transition from finite to infinite in a continuous way and to study which constructs are continuous functors in a corresponding algebroidal category. The work presented here provides a basis for a constructive analysis of Chu spaces and opens the door to a more systematic investigation of such an analysis in a variety of settings.

References

1. Amadio, R., Curien, P.-L.: Domains and Lambda-Calculi. Cambridge University Press, Cambridge (1998)
2. Barr, M.: *-Autonomous categories, with an appendix by Po Hsiang Chu. Lecture Notes in Mathematics, vol. 752. Springer, Heidelberg (1979)
3. Barr, M.: *-Autonomous categories and linear logic. Mathematical Structures in Computer Science 1, 159–178 (1991)
4. Devarajan, H., Hughes, D., Plotkin, G., Pratt, V.: Full completeness of the multiplicative linear logic of Chu spaces. In: 14th Symposium on Logic in Computer Science (Trento, 1999), pp. 234–243. IEEE Computer Society Press, Los Alamitos (1999)
5. Erné, M.: General Stone duality. Topology and Its Applications 137, 125–158 (2004)
6. Droste, M., Göbel, R.: Universal domains and the amalgamation property. Mathematical Structures in Computer Science 3, 137–159 (1993)
7. Droste, M.: Universal homogeneous causal sets. Journal of Mathematical Physics 46, 122503 1–10 (2005)
8. Ganter, B., Wille, R.: Formal Concept Analysis. Springer, Heidelberg (1999)
9. Hitzler, P., Zhang, G.-Q.: A cartesian closed category of approximable concept structures. In: Pfeiffer, Wolff (eds.) Proceedings of the International Conference on Conceptual Structures, Huntsville, Alabama, USA, July, Lecture Notes in Artificial Intelligence (to appear)
10. Hitzler, P., Krötzsch, M., Zhang, G.-Q.: A categorical view on algebraic lattices in Formal Concept Analysis. Fundamenta Informaticae 74(2-3), 301–328 (2006)
11. Lamarche, F.: From Chu spaces to cpos. In: Theory and Formal Methods of Computing 94, pp. 283–305. Imperial College Press (1994)
12. Mac Lane, S.: Categories for the Working Mathematician. Springer, Heidelberg (1971)
13. Plotkin, G.: A powerdomain construction. SIAM J. Comput. 5, 452–487 (1976)
14. Plotkin, G.: T^ω as a universal domain. J. Comp. Sys. Sci. 17, 209–236 (1978)
15. Plotkin, G.: Notes on the Chu construction and recursion.(accessed January 2007), http://boole.stanford.edu/pub/gdp.pdf
16. Pratt, V.: Chu spaces. School on Category Theory and Applications, Textos Mat. SCr. B, vol. 21, pp. 39–100, Univ. Coimbra, Coimbra (1999)
17. Pratt, V.: Higher dimensional automata revisited. Math. Structures Comput. Sci. 10, 525–548 (2000)
18. Pratt, V.: Chu spaces from the representational viewpoint. Ann. Pure Appl. Logic 96, 319–333 (1999)

19. Pratt, V.: Towards full completeness of the linear logic of Chu spaces. Mathematical foundations of programming semantics (Pittsburgh, PA, 1997), Electronic Notes in Theoretical Computer Science, vol. 7, p. 18 (1997)
20. Pratt, V.: The Stone gamut: a coordinatization of mathematics. In: Proceedings of 10th Annual Symposium on Logic in Computer Science, pp. 444–454 (1995)
21. Pratt, V.: Chu spaces and their interpretation as concurrent objects. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 392–405. Springer, Heidelberg (1995)
22. Pratt, V.: Chu spaces as a semantic bridge between linear logic and mathematics. Theoretical Computer Science 294, 439–471 (2003)
23. Scott, D.: Data types as lattices. SIAM J. Comput. 5, 522–586 (1976)
24. Scott, D.: Domains for denotational semantics. In: Nielsen, M., Schmidt, E.M. (eds.) Automata, Languages, and Programming. LNCS, vol. 140, pp. 577–613. Springer, Heidelberg (1982)
25. Zhang, G.-Q.: Chu spaces, concept lattices, and domains. In: Proceedings of the 19th Conference on the Mathematical Foundations of Programming Semantics, Montreal, Canada, March 2003. Electronic Notes in Theoretical Computer Science, vol. 83, p. 17 (2004)
26. Zhang, G.-Q., Shen, G.: Approximable concepts, Chu spaces, and information systems. In: De Paiva, V., Pratt, V. (eds.) Theory and Applications of Categories, Special Volume on Chu Spaces: Theory and Applications, vol. 17(5), pp. 80–102 (2006)

Structured Co-spans: An Algebra of Interaction Protocols*

José Luiz Fiadeiro and Vincent Schmitt

Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
{jose,vs27}@mcs.le.ac.uk

Abstract. We extend the theory of (co-)spans as a means of providing an algebraic approach to complex interactions as they arise in software-intensive systems. In order to make interconnections independent of the nature of components involved, interaction protocols are formalised not in terms of morphisms (i.e. part-of relationships) but a generalised notion of (co-)span in which the arms are structured morphisms – the head (the glue of the protocol) and the hands (the interfaces of the protocol) belong to different categories, the category of glues being coordinated over that of the interfaces. The proposed generalization sheds some additional light into adjunctions in bicategories, namely on the factorisation of left adjoint 2-sided enrichments.

1 Introduction

Software is becoming an integral part of a range of products and services performing vital functions in all sectors of economic and social activity. In such *software-intensive systems*, software applications are required to *interact*, in a seamless way, with other software components, devices, sensors, even humans. The complexity involved in building the software components that will be deployed in such systems in not so much on the "size" of their code but on the number and intricacy on the interactions in which they will be involved, what in [6] we have called *social complexity*. From an algebraic point of view, social complexity raises new challenges with respect to the more established *physiological complexity*, i.e. the fact that a complex whole can be understood as a composition of its parts. The basic difference is that it does not make sense to see software-intensive systems as being compositions, in an algebraic sense, of simpler components. There is not a notion of whole to which the parts contribute but, rather, a number of autonomous entities that interact with each other through external connectors.

This is why it is so important to put the notion of interaction at the centre of research in software-intensive system modelling, and to support methods and languages that separate interaction concerns from computational ones. In the past, we developed

* This work was partially supported through the IST-2005-16004 Integrated Project *SENSORIA: Software Engineering for Service-Oriented Overlay Computers*.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 194–208, 2007.
© Springer-Verlag Berlin Heidelberg 2007

a categorical framework supporting the separation between "computation" and "coordination" as architectural dimensions in software development [9]. This framework is based on what we have called "coordinated categories" [5] – concrete categories (faithful functors) that externalise the interfaces used by components to interact with other components. From an algebraic point of view, we propose to work with "structured morphisms"[1], i.e. pairs $<f,S>$ where $f:A \rightarrow \mathbf{G}S$ is a $\mathbf{C}$-morphism, $A:\mathbf{C}$, $S:\mathbf{D}$, and $\mathbf{G}:\mathbf{D} \rightarrow \mathbf{C}$. The motivation is that $\mathbf{G}$ "forgets" the computational part of the objects of $\mathbf{D}$ and returns their interfaces; structured morphisms capture interactions that do not depend on the computational processes involved in components.

Ultimately, the (autonomic) entities that we wish to interconnect need not be organised in a category. Typically, in a category of systems, morphisms capture a "component-of" or "sub-system" relationship. As already motivated, in software-intensive systems it does not make sense to talk about "component-of" relationships in an algebraic way. Therefore, we decided to look for algebraic mechanisms of interconnection that can capture peer-to-peer interactions among autonomous components. That is why, in this paper, we report on the use of co-spans – pairs $<f_A,f_B>$ where $f_A:A \rightarrow S$ and $f_B:B \rightarrow S$ are morphisms of a category $\mathbf{D}$. Co-spans (and their dual – spans) have been deserving increasing attention in computer science, namely when $\mathbf{D}$ is a category of graphs (or variants of graphs) as models of concurrent processes or reactive systems [14] – generalised automata or transition systems in one sense or another. For instance, spans can be used for defining composition operations along interfaces, which capture the behaviour of communicating parallel processes [11].

Because we want the application of interaction protocols to be "agnostic" to the nature of the computations that are performed by the peers, we want that the protocol be based on the interfaces that components have available for interacting with each other, not on the computations that they perform locally. This suggests that the interactions should be established between objects of a category of interfaces, not between behaviours. That is, we should work with co-spans based on structured morphisms – triples $<f_A,S,f_B>$ where S is an object of $\mathbf{D}$ and $f_A:A \rightarrow \mathbf{G}S$, $f_B:B \rightarrow \mathbf{G}S$ are structured morphisms of a coordinated category $\mathbf{G}:\mathbf{D} \rightarrow \mathbf{C}$.

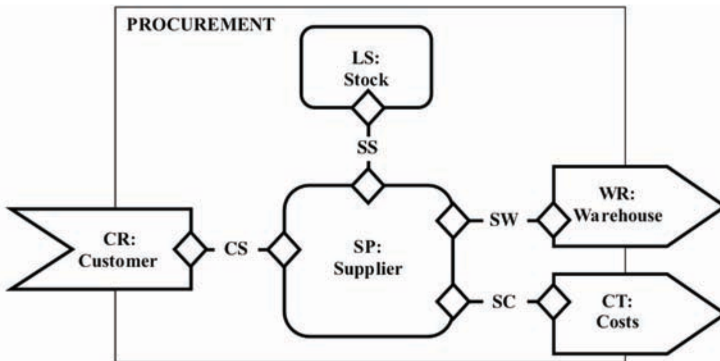
Our approach is also different to the traditional uses of (co-)spans in that we are interested in a more declarative setting in which the objects of $\mathbf{D}$ are not operational models of behaviour (automata, transition systems, and so on), but specifications or designs of protocols. This is why we are interested in other categories than that of graphs. In fact, we will work over coordinated categories in general, which include graphs and other models of concurrency, but also logical and algebraic specifications [5,10,13].

Our purpose in this paper is to generalise the theory of co-spans to support an algebraic approach to interactions in software-intensive systems as discussed above. In Section 2, for further motivation, we present a case study that we have been developing for service-oriented modelling in the context of the SENSORIA project. In Section 3, we discuss in more detail the notion of interaction protocol that we have in mind and the role played by structured co-spans. Finally, in Section 4, we investigate the properties of bicategories of structured co-spans, which leads to some interesting new results in adjuncions of 2-sided enrichments.

2 Modules for Software-Intensive Systems

The work that we present in this paper has been inspired by research that we have been developing within the IST-FET Integrated Project SENSORIA – *Software Engineering for Service-Oriented Overlay Computers* – on the emerging service-oriented computing paradigm, generalising methods and techniques already proposed for web-service and grid technologies. From the point of view of software-intensive systems, services can be understood as autonomous, platform-independent computational entities that can be described, published, discovered, and dynamically assembled for developing massively distributed, interoperable, evolvable systems. In this paper, we do not address the publication and run-time discovery process that characterises the service-oriented paradigm. This is because we concentrate on the static structure of systems, not on the process through which they can be dynamically configured.

The modelling language that we have been defining in SENSORIA – SRML – offers a notion of module through which composite services can be specified as assemblies of internal components and externally procured services [7]. In order illustrate and motivate the notion of module, we use a typical procurement business process involving a supplier *SP*, a warehouse *WR*, a local stock *LS*, a price look-up facility *CT*, and a customer *CR*.



This module declares *SP* and *LS* as components. Components are the computational units that constitute the core of the module and are typed by what we call *business roles*; in the example, *SP* plays the business role of *Supplier* and *LS* of *Stock*. A business role specifies the activity performed by a component in terms of a collection of transitions. As an example (see [7] for an explanation of the syntax), the business role *Stock* models a behaviour pattern that is typical of a database view:

```
BUSINESS ROLE Stock is
  INTERACTIONS
    rpl get(product):nat
    prf set(product,nat)
  ORCHESTRATION
    local qoh:product→nat
    transition
      triggeredBy get(p)
      sends qoh(p)
```

```
transition
   triggeredBy set(p,n)
   effects qoh(p)'=n
```

The model provided through a business role is independent of the language in which the component is programmed and the platform in which it is deployed. The “orchestration”, i.e. the specification of the pattern of behaviour exhibited by the component, is independent of the specific parties that are actually interconnected with it in any given run-time configuration; a component is totally independent in the sense that it does not invoke services of any specific co-party – it just offers an interface of two-way interactions in which it can participate. Interconnections with other entities are established through what we call *wires*, as discussed below.

Modules can identify external parties that play a role in the business process – *WR*, *CT* and *CS*. Making certain parties external reflects looser coupling and late binding. For instance, making the warehouse *WR* an external party reflects the fact that the choice of warehouse should probably be made at run-time, e.g. taking into account properties of the customer like its location. Every external party is typed by what we call a *business protocol*, which specifies a stateful interaction between a component and the corresponding party. In SRML, this specification is given in a temporal logic of interactions. As an example (once again, please see [7] for an explanation of the syntax used in business protocols), consider the behaviour required of a warehouse:

```
BUSINESS PROTOCOL Warehouse is
_____________________________________________
  INTERACTIONS
     r&s check&lock
     snd confirm
  BEHAVIOUR
     initially check&lock🔔?
     check&lock⌛ ⊃ (check&lock✓? ensures confirm🔔!)
     check&lock✓? ⊃ (check&lock✝? exceptif confirm🔔!)
```

Basically, we are stating that (1) in the initial state the warehouse is ready to receive a request for engaging in the interaction *check&lock* (which the wire *SW* connects to *BA* – the booking agent), (2) the warehouse promises to issue *confirm* if a commitment to the deal proposed by *check&lock* is received within an agreed delay, and (3) the commitment can be revoked until the *confirm* is actually issued. The difference with respect to business roles is that, instead of an orchestration, a business protocol declares the set of properties that the co-party is required to adhere to. Otherwise, both business roles and protocols share the same kind of declaration of the interactions in which they can be involved, what we call their *signatures*.

Modules can offer an external interface for other modules to use its services – *CR* in the case at hand. The corresponding business role specifies constraints on the interactions that the module supports as a service provider such as the order in which they expect invocations to be made or deadlines for the user to commit.

Finally, *wires* connect the components and external interfaces of a module. In the case of *PROCUREMENT* these are *CS*, *SS, SW* and *SC*. Wires are labelled by connectors that coordinate the interactions in which the parties are jointly involved. In SRML, we model the interaction protocols involved in these connectors as separate, reusable entities. Just like business roles and protocols, an interaction protocol is specified in

terms of a number of interactions. Because interaction protocols establish a relationship between two parties, the interactions in which they are involved are divided in two subsets called roles – *A* and *B*. The "semantics" of the protocol is provided through a collection of sentences – what we call *interaction glue* – that establish how the interactions are coordinated. This may include routing events and transforming sent data to the format expected by the receiver. As an example, consider the following protocol used in the wire *SS* that connects *Supplier* and *Stock*:

```
INTERACTION PROTOCOL Custom1 is
  ROLE A
    ask S1(product,nat):bool
    tll S2(product,nat)
    tll S3(product,nat)
  ROLE B
    rpl R1(product):nat
    prf R2(product,nat)
  COORDINATION
    S1(p,n) = R1(p)≥n
    S2(p,n) ⊃ R2(p,R1(p)+n)
    R1(p)≥n ∧ S3(p,n) ⊃ R2(p,R1(p)–n)
    R1(p)<n ⊃ ¬S3(p,n)
```

The wire itself is specified in SRML in a tabular form as follows:

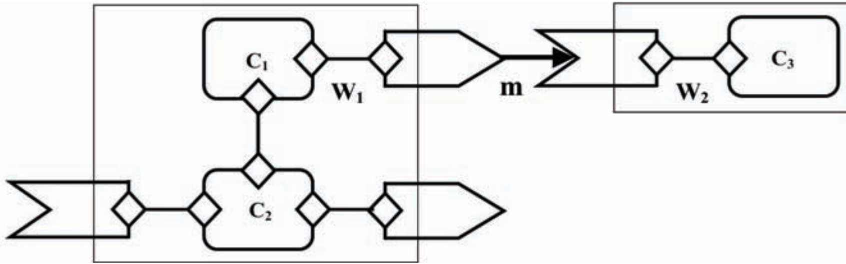
SP Supplier		**SS**		**LS** Stock
ask checkStock **tll** incStock **tll** decStock	S_1 S_2 S_3	Custom1	R_1 R_2	**rpl** get **prf** set

The name bindings instantiate the two roles of the interaction protocol with *Supplier* and *Stock*, respectively, thus establishing that the interactions between the two parties satisfy the following properties:

```
checkStock(p,n)=(get(p)≥n)
incStock(p,n) ⊃ set(p,get(p)+n)
get(p)≥n ∧ decStock(p,n) ⊃ set(p,get(p)–n)
get(p)<n ⊃ ¬decStock(p,n)
```

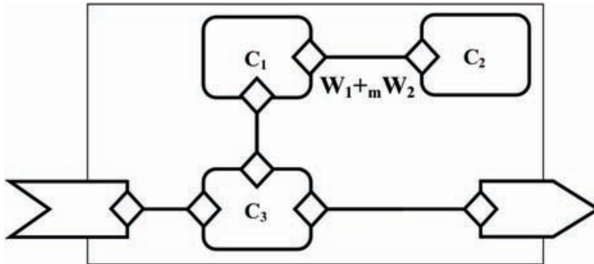
That is, the boolean value returned by *checkStock(p,n)* as invoked by the supplier is computed by the local stock by checking if the value returned by *get(p)* is greater or equal to *n*. The protocol also stipulates that to a request from the supplier for *incStock(p,n)* the local stock executes *set(p,get(p)+n)*. Likewise, to a request from the supplier for *decStock(p,n)* the local stock executes *set(p,get(p)–n)* only if *get(p)* returns a value greater than or equal to *n*; otherwise, the request is not accepted.

The fact that business protocols are specifications over a logic of interactions is important because it will allow us to compose modules by matching the properties required by an external interface of one module with those provided by another. The matching involves what we call an *external wire m*: this is a mapping from the interactions of the "requires" external interface to the interactions of the "provides" external interface that preserves the properties, i.e. *m* defines an interpretation between the theories of the business protocols involved.



An external wire is based on an "empty" interaction protocol, i.e. it does not superpose any additional coordination effects to the (internal) wires W_1 and W_2, it just binds the interactions declared in the external interfaces.

The composition of the two modules results from the composition of the two wires W_1 and W_2 via the mapping *m* to provide a wire between the two components.



Notice that there is no composition law on components, just on connectors (which extends to wires). Components correspond to software applications, possibly implemented in different languages and running in different platforms; therefore, it does not make sense to compose in the same way that, for instance, a compiler links a number of modules to produce an executable program. This is where we see the difference between social and physiological complexity as already mentioned, which motivates the need for a different algebraic approach.

3 The Algebraic Structure of Connectors

An algebraic formalisation of this notion of module and module composition has been given in [8] from the point of view of a notion of correctness defined based on the theory of institutions [10]. In this paper, we will explore the algebraic structure of connectors in more detail and in a more general setting that does not require the level of detail that we used in [8].

As motivated in Section 2, interactions constitute the core and the unifying element of the proposed approach to systems modelling: all the models that we work with – business roles, business protocols and interaction protocols – are based on structures of interactions. We assume that these structures are organised in a category ***SIGN*** (of signatures) whose morphisms capture "part-of" relationships, i.e. a morphism $\sigma{:}S_1 \rightarrow S_2$ formalises the way a signature (structure of interactions) S_1 is part of S_2 up to

a possible renaming of the interactions and corresponding parameters. In order to support composition, we further assume that ***SIGN*** is finitely co-complete.

The other structure that is important for interaction protocols is that of the glues; we assume that glues can themselves be organised in a category ***IGLU*** and that a functor ***sign:IGLU→SIGN*** returns, for every glue, the structure of interactions (signature) that are being coordinated by the protocol. As a consequence, a morphism $\sigma:G_1\to G_2$ of glues captures the way G_1 is a sub-protocol of G_2, again up to a possible renaming of the interactions and corresponding parameters. That is, σ identifies the glue that, within G_2, captures the way G_1 coordinates the interactions $\mathbf{sign}(G_1)$ as a part of $\mathbf{sign}(G_2)$. In fact, because we need to be able to compose interaction protocols, we assume that ***IGLU*** is also a finitely co-complete category.

In this formal setting, every interaction protocol P consists of an interaction glue G together with two signature morphisms $\pi_A:roleA\to\mathbf{sign}(G)$ and $\pi_B:roleB\to\mathbf{sign}(G)$. The fact that the roles of the protocol are signatures, and not glues, is important because, as motivated in Section 2, wires establish interconnections between entities (components or external interfaces) purely through relationships between the interactions in which the entities can be involved. These relationships are "syntactic" and are established through the roles of the interaction protocol. If we were to include properties in the roles, we would be involving the computational properties of the entities to which the role is connected. More precisely, we remain agnostic as to the nature of the entities that we wish to interconnect. The only assumption that we make is that each such entity n has a defined signature $\mathbf{sign}(n):\mathbf{SIGN}$.

The need for separating the mechanisms available for coordinating interactions from the computations that entities execute internally suggests that we work with *coordinated categories* [5]; asking ***sign:IGLU→SIGN*** to be coordinated means that:

- ***sign*** is faithful, i.e. ***IGLU*** is concrete over ***SIGN*** in the sense of [1].
- ***sign*** lifts colimits, i.e. given any diagram $\mathbf{dia}:I\to\mathbf{IGLU}$ and colimit $(\mathbf{sign}(G_i)\to A)_{i:I}$ of $(\mathbf{dia};\mathbf{sign})$ there exists a colimit $(G_i\to G)_{i:I}$ of $\mathbf{dia}$ such that $\mathbf{sign}(G_i\to G)=(\mathbf{sign}(G_i)\to A)$.
- ***sign*** has discrete structures in the sense of [1], i.e. every signature A has a 'discrete lift' meaning that there exists $iglu(A):\mathbf{IGLU}$ such that, for every $f:A\to\mathbf{sign}(G)$, there is $f':iglu(A)\to G$ such that $\mathbf{sign}(f')=f$.

These properties capture the notion of separation of 'coordination' from 'computation' in the following sense:

- Making ***sign*** faithful means that the computational aspects do not give rise to other interactions than those captured through signatures.
- Lifting colimits means that glues can be composed if their signatures can, and that the signature of the composed glue does not depend on the computations performed by the components.
- The existence of discrete structures means that every signature A has a "realisation" (a discrete lift) as a glue $iglu(A)$ in the sense that, using A to interconnect a glue G, which is achieved through a morphism $f:A\to\mathbf{sign}(G)$, is tantamount to using $iglu(A)$ through any $f':iglu(A)\to G$ *such that* $\mathbf{sign}(f')=f$. Notice that, because ***sign*** is faithful, there is only one such f', which means that f and

f' are, essentially, the same. That is, sources of morphisms in diagrams in ***IGLU*** are, essentially, signatures, which is why we decided to work with structured morphisms in interaction protocols.

Coordinated categories have strong algebraic properties, almost as strong as those of topological categories [1]: a coordinated category is topological iff ***sign*** lifts colimits uniquely. Examples include specifications as theories (or theory presentations) in institutions [10], as well as models of concurrency [13] where signatures consist of process alphabets.

Some of the properties that we will find useful are [5]:

- The functor ***sign*** admits a left adjoint ***iglu**:**SIGN**→**IGLU***.
- The units of the adjunction are identities and the co-units are epis.
- The functor ***sign*** preserves colimits.

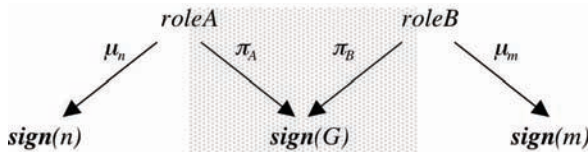
In order to understand the role played by interaction protocols, consider once again the wire *SS* discussed in Section 2:

SP Supplier	◇—	**SS**	—◇	**LS** Stock
ask checkStock	S_1	Custom1	R_1	**rpl** get
tll incStock	S_2		R_2	**prf** set
tll decStock	S_3			

The wire establishes two signature morphisms: one from the *ROLE_A* of *Custom1* to the signature of *Supplier*, and the other from *ROLE_B* to *Stock*. For instance, the latter is given by the following fragment of the table:

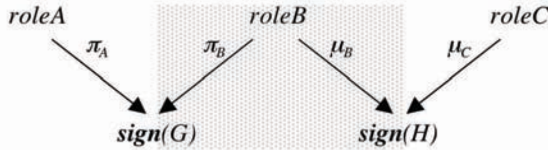
ROLE_B	μ	Stock
R_1	→	get
R_2	→	set

We call a *connector* for a wire $n \leftrightarrow m$ between entities n and m in a module, a structure $<\mu_n,\pi_A,G,\pi_B,\mu_m>$ where $<\pi_A,G,\pi_B>$ is an interaction protocol P and $<\mu_n,\mu_m>$ are the morphisms that connect the roles of P to the entities n and m. Such a connector defines the following diagram in ***SIGN***:



The interaction protocol $<\pi_A,G,\pi_B>$ corresponds to the shadowed part of the diagram. Although this fragment is a co-span in ***SIGN***, the protocol itself is not because it involves the glue G. Indeed, without the computational aspects of the glue it would not be possible to coordinate the interactions between n and m. That is, co-spans in ***SIGN*** are not expressive enough to formalise interaction protocols.

The significance of the difference becomes apparent when we consider the composition of two interaction protocols $<\pi_A,G,\pi_B>$ and $<\mu_B,H,\mu_C>$. We know how to compose the corresponding co-spans in ***SIGN*** through a pushout of the shadowed triangle, but the pushout does not deliver us a glue:



On the other hand, we have already seen that working with co-spans in ***IGLU*** does not make sense because, by allowing the roles to involve computational aspects, the morphisms that connect the roles of the protocol to the entities would bring in computational aspects of the entities into their interconnection.

This is the motivation for studying the properties of structures of the form $<\pi_A,G,\pi_B>$, which we call *structured co-spans*. More precisely, our aim is to define and study the properties of a bicategory whose objects are signatures and whose 1-cells consist of interaction protocols.

4 Structured Co-spans

In this section we define and study the algebraic properties of structured co-spans. We start by recalling some basic definitions and properties of bicategories but only as a reminder – we refer the reader to either the original paper by Bénabou [2] or the more accessible textbook [3]; notice that many other papers are available on this topic but the terminology may change slightly from the one that we use ([3]).

We start by recalling that bicategories were introduced to consider generalisations of categorical constructions to the case in which the identity and composition laws are satisfied only "up to isomorphism". A *bicategory* $\boldsymbol{V}$ consists of:

- A class $|\boldsymbol{V}|$ of objects (also called 0-cells)
- For each pair $<A,B>$ of objects, a category $\boldsymbol{V}(A,B)$ whose objects are called arrows (or 1-cells) and whose morphisms are called 2-cells
- For every triple $<A,B,C>$ of objects, a composition law given by a (bi)functor $;_{A,B,C}$: $\boldsymbol{V}(A,B)\times\boldsymbol{V}(B,C)\rightarrow\boldsymbol{V}(A,C)$
- For every object A an identity arrow $1_A:A\rightarrow A$

The typical axioms of categories are replaced by the existence of a number of natural isomorphisms and coherence conditions. For simplicity, we omit these properties and refer the reader to [3]. We have already mentioned that typical examples of bicategories in computer science are (co-)spans of graphs [11,14].

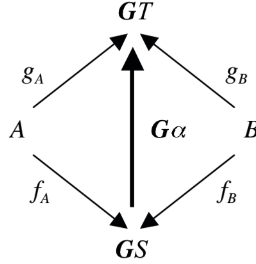
A similar generalisation applies to functors. Given bicategories $\boldsymbol{V}$ and $\boldsymbol{W}$, a *lax functor* $\boldsymbol{F}:\boldsymbol{V}\rightarrow\boldsymbol{W}$ consists of:

- A map sending objects A of $\boldsymbol{V}$ to objects $\boldsymbol{F}A$ of $\boldsymbol{W}$
- Functors $\boldsymbol{F}_{A,B}:\boldsymbol{V}(A,B)\rightarrow\boldsymbol{W}(\boldsymbol{F}A,\boldsymbol{F}B)$ for every pair $<A,B>$ of objects of $\boldsymbol{V}$, a
- 2-cells $\boldsymbol{F}^2_{f,g}:\boldsymbol{F}f;\boldsymbol{F}g\rightarrow\boldsymbol{F}(f;g)$ for every composable $<f,g>$ in $\boldsymbol{V}$, natural in f and g
- 1-cells $\boldsymbol{F}^0_A:1_{\boldsymbol{F}A}\rightarrow\boldsymbol{F}1_A$ for every object A of $\boldsymbol{V}$

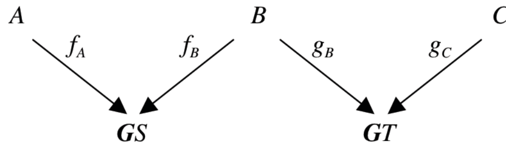
subject to coherence conditions [3]. A lax functor $\boldsymbol{F}$ is a *pseudo-functor* when all the $\boldsymbol{F}^2_{f,g}$ and $\boldsymbol{F}^0_A$ are invertible.

Definition 4.1. Given an adjunction $\boldsymbol{F} \dashv \boldsymbol{G}:\boldsymbol{D} \rightarrow \boldsymbol{C}$, where $\boldsymbol{D}$ has pushouts, we define the bicategory ***co-span(G)*** of $\boldsymbol{G}$-structured co-spans as follows:

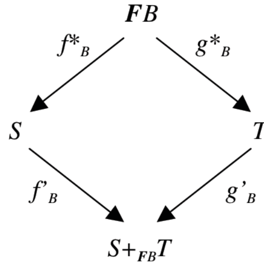
- The objects are those of $\boldsymbol{C}$
- The arrows (1-cells) are triples $<f_A:A \rightarrow \boldsymbol{G}S, S, f_B:B \rightarrow \boldsymbol{G}S>$ where S is an object of $\boldsymbol{D}$ and f_A, f_B are morphisms of $\boldsymbol{C}$
- A 2-cell $\alpha:<f_A,S,f_B> \rightarrow <g_A,T,g_B>$ is a $\boldsymbol{D}$-morphism $\alpha:S \rightarrow T$ that makes the following diagram commute



- Composition of < S > and < T >



is $<f_A;\boldsymbol{G}f'_B, S+_BT, g_C;\boldsymbol{G}g'_B>$ obtained through the following pushout in $\boldsymbol{D}$ where $f^*_B=\boldsymbol{F}f_B;\varepsilon_S$ and $g^*_B=\boldsymbol{F}g_B;\varepsilon_T$



- The identities are $<id_A;\eta_A,\boldsymbol{F}A,id_A;\eta_A>$.

As could be expected:

Remark 4.2. For every category $\boldsymbol{C}$ with pushouts, ***co-span(1***$_{\boldsymbol{C}}$***)*** is the well-know category of co-spans over $\boldsymbol{C}$, which we denote by ***co-span(C)***.

The following property is easily proved:

Proposition 4.3. Let $\boldsymbol{F}:\boldsymbol{C} \rightarrow \boldsymbol{D}$ be a functor between two categories with pushouts.

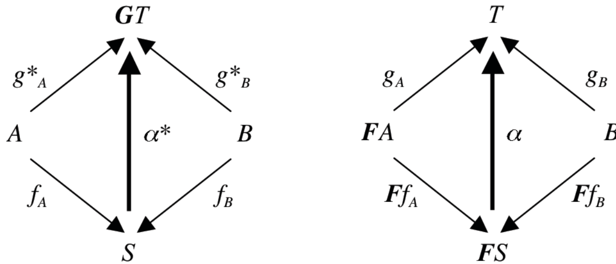
- $\boldsymbol{F}$ extends to a lax $\dot{\boldsymbol{F}}$***:co-span(C)→co-span(D)*** by pointwise translation.
- $\dot{\boldsymbol{F}}$ is normal, i.e. it sends identity 1-cells to identities.
- If $\boldsymbol{F}$ preserves pushouts, $\boldsymbol{F}$ is a pseudo-functor.

We are now going to analyse the lifting of adjunctions. As established in [12], bicategories admit more general morphisms than lax functors – the so-called "2-sided enrichments", which together with the appropriate 2-cells and 3-cells form the so-called tricategory ***Caten***. The definition of adjoint one-cells makes sense in any bicategory and, in particular, in ***Caten*** where they are characterised as follows:

Theorem 4.4 ([12] Proposition 2.7)**.** A left adjoint 2-sided enrichment $\boldsymbol{F{:}V \rightarrow W}$ is a pseudo-functor such that each functor $\boldsymbol{F_{A,B}}$: $V(A,B) \rightarrow W(FA,FB)$ has a right adjoint.

Consider now the case in which we are given an adjunction $\boldsymbol{F \dashv G{:}D \rightarrow C}$, where $\boldsymbol{D}$ and $\boldsymbol{C}$ have pushouts:

- Because $\boldsymbol{F}$ preserves pushouts, $\dot{\boldsymbol{F}}$ is a pseudo-functor
- Each functor $\dot{\boldsymbol{F}}_{A,B}$: ***co-span(C)(A,B)→co-span(D)***$(\dot{F}A,\dot{F}B)$ has a right-adjoint based on the isomorphisms between the two hom-sets:

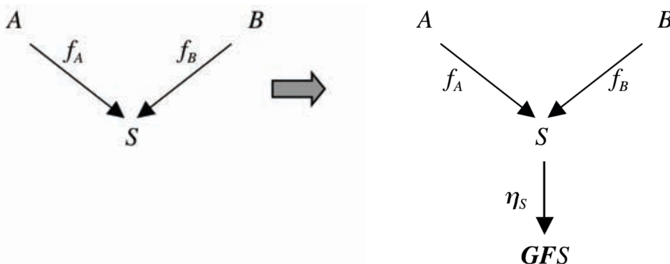


Corollary 4.5. Given an adjunction $\boldsymbol{F \dashv G{:}D \rightarrow C}$ where $\boldsymbol{D}$ and $\boldsymbol{C}$ have pushouts, $\dot{\boldsymbol{F}}$:***co-span(C)→co-span(D)*** is a left adjoint 2-sided enrichment.

Notice that nothing can be inferred from this result about the lax functor $\dot{\boldsymbol{G}}$***:co-span(D)→co-span(C)***. We are now going to see that ***co-span(G)*** allows us to strengthen the case.

Proposition 4.6. Given an adjunction $\boldsymbol{F \dashv G{:}D \rightarrow C}$ where $\boldsymbol{D}$ and $\boldsymbol{C}$ have pushouts, we define a pseudo functor ***F*:co-span(C)→co-span(G)*** as follows:

- $\boldsymbol{F}$* is the identity on objects
- Every 1-cell $<f_A,S,f_B>$ is mapped to $<f_A;\eta_S,\boldsymbol{FS},f_B;\eta_S>$, and the 2-cells $\alpha{:}<f_A,S,f_B>\rightarrow<g_A,T,g_B>$ to $\boldsymbol{F}\alpha{:}<f_A;\eta_S,\boldsymbol{FS},f_B;\eta_S>\rightarrow<g_A;\eta_T,\boldsymbol{FT},g_B;\eta_ST>$. Notice that, being a left adjoint, $\boldsymbol{F}$ preserves colimits, which justifies that we do obtain a pseudo functor˙



If we consider the hom-categories, it is easy to see that we have lifted the adjunction $\boldsymbol{F \dashv G{:}D \rightarrow C}$ to an adjunction $\boldsymbol{co\text{-}span(G)}(A,B) \rightarrow \boldsymbol{co\text{-}span(C)}(A,B)$.

Proposition 4.7. $\boldsymbol{F^*{:}co\text{-}span(C) \rightarrow co\text{-}span(G)}$ is a left adjoint 2-sided enrichment. Moreover, because $\boldsymbol{F^*}$ is the identity on objects, we obtain a right adjoint that is a lax functor $\boldsymbol{{}^*G{:}co\text{-}span(G) \rightarrow co\text{-}span(C)}$.

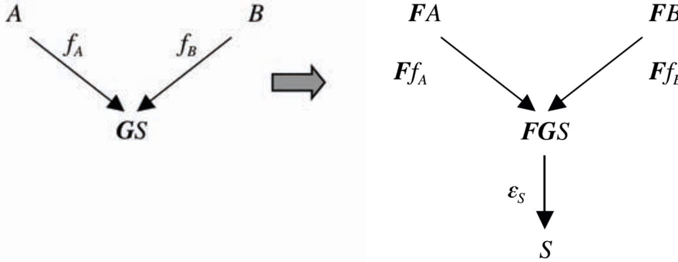
It is interesting to analyse the construction of the right-adjoint:

- $\boldsymbol{{}^*G}$ is again the identity on objects
- Every 1-cell $<f_A,S,f_B>$ is mapped to $<f_A,\boldsymbol{G}S,f_B>$, and the 2-cells $\alpha{:}<f_A,S,f_B> \rightarrow <g_A,T,g_B>$ to $\boldsymbol{G}\alpha{:}<f_A,\boldsymbol{G}S,f_B> \rightarrow <g_A,\boldsymbol{G}T,g_B>$.

Recall that the identities for structured co-spans are of the form $<id_A;\eta_A,\boldsymbol{F}A,id_A;\eta_A>$. Hence, $\boldsymbol{{}^*G}_A=\eta_A$. Moreover, $\boldsymbol{G}$ does not necessarily preserve pushouts. This is why we cannot guarantee that $\boldsymbol{{}^*G}$ is a pseudo-functor. However if $\boldsymbol{G}$ defines a coordinated category, we know that it preserves colimits and the units of the adjunction are identities.

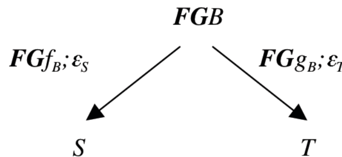
Proposition 4.8. If $\boldsymbol{F \dashv G{:}D \rightarrow C}$ is a coordinated category, we have an adjunction $\boldsymbol{F^* \dashv {}^*G{:}co\text{-}span(G) \rightarrow co\text{-}span(C)}$ of pseudo functors.

Consider now what happens on the side of $\boldsymbol{co\text{-}span(D)}$. We have an obvious pseudo functor based on $\boldsymbol{F}$ and the functors $\boldsymbol{co\text{-}span(G)}(A,B) \rightarrow \boldsymbol{co\text{-}span(D)}(\boldsymbol{F}A,\boldsymbol{F}B)$ that map structured co-spans $<f_A,S,f_B>$ to the co-spans $<f^*_A,S,f^*_B>$:

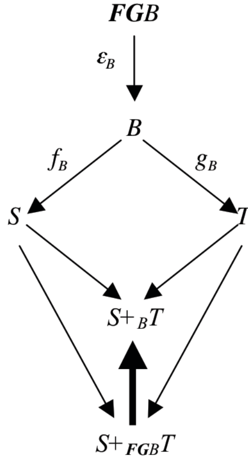


These functors are isomorphisms, leading to a left adjoint 2-sided enrichment $\boldsymbol{{}^*F}$.

Mapping co-spans $<f_A,S,f_B>$ over $\boldsymbol{D}$ to structured co-spans $<\boldsymbol{G}f_A,S,\boldsymbol{G}f_B>$ seems equally obvious, but the mapping of the composition deserves some attention. If we consider $<f_A,S,f_B>;<g_B,T,g_C>$, the composition of the images is given by a pushout of:



Because $\boldsymbol{FG}f_B;\varepsilon_S=\varepsilon_B;f_B$ and $\boldsymbol{FG}g_B;\varepsilon_T=\varepsilon_B;g_B$, we have in fact:



The universal properties of the colimit return a morphism $S+_{FGB}T \rightarrow S+_{B}T$. If we work with a coordinated category, ***G*** is faithful, which implies that the co-units are epis. In this case, it is easy to see that the morphisms $S+_{FGB}T \rightarrow S+_{B}T$ are in fact isomorphisms, which makes ***G**** a pseudo functor.

We can now summarise our results.

Theorem 4.9. Let $\boldsymbol{F}:\boldsymbol{C} \rightarrow \boldsymbol{D}$ be a functor between two categories with pushouts.

- ***F*** extends to a normal lax functor $\dot{\boldsymbol{F}}$:***co-span(C)→co-span(D)***; if ***F*** preserves pushouts, $\dot{\boldsymbol{F}}$ is a pseudo-functor.
- If ***F*** has a right adjoint $\boldsymbol{G}:\boldsymbol{D} \rightarrow \boldsymbol{C}$:
- $\dot{\boldsymbol{F}}$ is a left adjoint 2-sided enrichment.
- $\dot{\boldsymbol{F}}$ factorises as $co\text{-}span(C) \xrightarrow{F^*} co\text{-}span(G) \xrightarrow{{}^*F} co\text{-}span(D)$ where ***F**** has a lax right adjoint ****G***

Theorem 4.10. Let $\boldsymbol{G}:\boldsymbol{D} \rightarrow \boldsymbol{C}$ be a coordinated category.

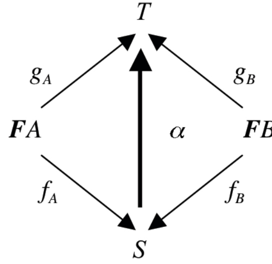
- The right adjoint ****G*** is a pseudo functor
- ****F*** has a pseudo right adjoint ***G****
- $\dot{\boldsymbol{F}} \dashv \dot{\boldsymbol{G}}$:***co-span(D)→co-span(C)*** is an adjunction of pseudo-functors

Our final result is a generalisation of the factorisation that we defined for $\dot{\boldsymbol{F}}$:***co-span(C)→co-span(D)*** to a general lax functor.

Definition 4.11. Given a lax functor $\boldsymbol{F}:\boldsymbol{V} \rightarrow \boldsymbol{W}$, we define the bicategory $\boldsymbol{V_F}$ as follows:

- $|\boldsymbol{V_F}| = |\boldsymbol{V}|$
- $\boldsymbol{V_F}(A,B) = \boldsymbol{W}(\boldsymbol{F}A,\boldsymbol{F}B)$

Notice that, in the case of $\dot{\boldsymbol{F}}$:***co-span(C)→co-span(D)*** what we obtain is a bicategory whose hom-cats are of the form:



which are isomorphic to ***co-span(G)(A,B)*** if ***F*** has a right adjoint ***G:D→C***.

Our last result is a canonical factorisation of left adjoint 2-sided enrichments:

Theorem 4.12. Every lax functor ***F: V→W*** factorises as $V \xrightarrow{F^*} V_F \xrightarrow{{}^*F} W$ where ***F**** is an identity on objects and ****F*** an identity on hom-cats. If ***F*** is a left adjoint 2-sided enrichment so is ***F**** and its right adjoint is lax.

5 Concluding Remarks

In this paper, we have shown how the notion of interaction protocol that we are developing within the SENSORIA project used for modelling interconnections in service-oriented systems can be given an algebraic semantics over an extension of the theory of co-spans. The extension is motivated by the fact that, whereas we want the interaction protocol to use a rich formalism to specify the coordination mechanisms superposed by the glue, its interfaces should be purely "syntactic" so as to avoid any assumption on the computations performed by the entities being interconnected.

More precisely, given a coordinated category ***sign:IGLU→SIGN***, using ***co-span(SIGN)*** for interconnections is too poor because it does not support the definition of coordination mechanisms, and using ***co-span(IGLU)*** is too strong because the interfaces involve computational aspects. This is why we proposed to work over an algebraic structure ***co-span(sign)*** that is based instead on ***sign***-structured morphisms.

We showed how ***co-span(sign)*** constitutes a bicategory. In fact, we investigated the more general issue of how the ***co-span*** construction relates to functors. We showed how a functor between the base categories induces a lax-functor between the corresponding bicategories of co-spans, and how adjunctions give rise to adjoint 2-sided enrichments. This allowed us to strengthen some results on adjunctions in the tricategory ***Caten***, namely by generalising the construction of ***co-span(sign)*** to a canonical factorisation of lax functors. This is a line that we would like to pursue on its own, although the "computational" inspiration that comes from (structured) co-spans and coordinated categories is very welcome.

From the point of view of software-intensive system modelling, it is clear that structured morphisms over coordinated categories have been proving to provide a richer algebraic framework when it comes to formalising interconnection mechanisms. This is another avenue that we want to keep exploring in SENSORIA.

Acknowledgments

We would like to acknowledge the contribution of Antónia Lopes with whom much of the work around coordinated categories in general and interaction protocols in particular, has been developed, and to thank our colleagues in SENSORIA, the IFIP WG1.3 group members and observers, and the participants of the Workshop on Applied and Computational Category Theory (ACCAT) 2007 for valuable feedback.

References

1. Adámek, J., Herrlich, H., Strecker, G.: Abstract and Concrete Categories. John Wiley & Sons, New York Chichester Brisbane Toronto Singapore (1990)
2. Bénabou, J.: Introduction to bicategories. In: Complementary Definitions of Programming Language Semantics. LNCS, vol. 42, pp. 1–77. Springer, Heidelberg (1967)
3. Borceux, F.: Handbook of Categorical Algebra 1. Cambridge University Press, Cambridge (1994)
4. Ehrig, H., Orejas, F., Braatz, B., Klein, M., Piirainen, M.: A component framework for system modeling based on high-level replacement systems. Software Systems Modeling 3, 114–135 (2004)
5. Fiadeiro, J.L.: Categories for Software Engineering. Springer, Heidelberg (2004)
6. Fiadeiro, J.L.: Designing for software's social complexity. IEEE Computer 40(1), 34–39 (2007)
7. Fiadeiro, J.L., Lopes, A., Bocchi, L.: A formal approach to service-oriented architecture. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 193–213. Springer, Heidelberg (2006)
8. Fiadeiro, J.L., Lopes, A., Bocchi, L.: Algebraic semantics of service component modules. In: Fiadeiro, J.L., Schobbens, P.Y. (eds.) Algebraic Development Techniques. LNCS, vol. 4409, pp. 37–55. Springer, Heidelberg (2007)
9. Fiadeiro, J.L., Lopes, A., Wermelinger, M.: A mathematical semantics for architectural connectors. In: Backhouse, R., Gibbons, J. (eds.) Generic Programming. LNCS, vol. 2793, pp. 190–234. Springer, Heidelberg (2003)
10. Goguen, J., Burstall, R.: Institutions: abstract model theory for specification and programming. Journal ACM 39(1), 95–146 (1992)
11. Katis, P., Sabadini, N., Walters, R.F.C.: Bicategories of processes. Journal of Pure and Applied Algebra 115, 141–178 (1997)
12. Kelly, G.M., Labella, A., Schmitt, V., Street, R.: Categories enriched on two sides. Journal of Pure and Applied Algebra 168, 53–98 (2002)
13. Sassone, V., Nielsen, M., Winskel, G.: A classification of models for concurrency. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 82–96. Springer, Heidelberg (1993)
14. Sassone, V., Sobocinski, P.: Reactive systems over cospans. In: LICS'05, pp. 311–320. IEEE Computer Society, Los Alamitos (2005)

Graphical Encoding of a Spatial Logic for the π-Calculus*

Fabio Gadducci and Alberto Lluch Lafuente

Dipartimento di Informatica, Università di Pisa
largo Bruno Pontecorvo 3c, I-56127 Pisa, Italia
gadducci@di.unipi.it, lafuente@di.unipi.it

Abstract. This paper extends our graph-based approach to the verification of spatial properties of π-calculus specifications. The mechanism is based on an encoding for mobile calculi where each process is mapped into a graph (with interfaces) such that the denotation is fully abstract with respect to the usual structural congruence, i.e., two processes are equivalent exactly when the corresponding encodings yield isomorphic graphs. Behavioral and structural properties of π-calculus processes expressed in a spatial logic can then be verified on the graphical encoding of a process rather than on its textual representation. In this paper we introduce a modal logic for graphs and define a translation of spatial formulae such that a process verifies a spatial formula exactly when its graphical representation verifies the translated modal graph formula.

1 Introduction

Spatial logics are formalisms for expressing behavioral and topological properties of system specifications, given as processes of a calculus. Besides the temporal modalities of the Hennessy-Milner tradition, these logics include ingredients for reasoning about the structural properties of a system. The connective **0** represents e.g. the (processes structurally congruent to the) empty system, and the formula $\phi_1|\phi_2$ is satisfied by processes that can be decomposed into two parallel components, satisfying ϕ_1 and ϕ_2, respectively.

The origins of such logics can be tracked back to early work on logics for reasoning about networks of processes (e.g. the *multiprocess network logic* of [22]). Recent approaches include logics for concurrent software system specifications given in process calculi like the π-calculus [3,4] and the ambient calculus [7], or for data structures such as graphs [5], heaps [23], and trees [6]. The approach we proposed in [16] to the verification of (recursion-free) spatial formulae [3] for π-calculus specifications is based on a graphical encoding for nominal calculi [15]. Even if a few articles had been already proposed on the verification of graphically described systems (see e.g [1,21,25]), to the best of our knowledge our approach was the only one that exploited a graphical presentation for the

* Research partially supported by the EU FP6-IST IP 16004 SEnSOria (*Software Engineering for Service-Oriented Overlay Computers*).

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 209–225, 2007.
© Springer-Verlag Berlin Heidelberg 2007

verification of behavioral and spatial properties of (finite) processes of a nominal calculus. A closely related work is the spatial logic for bigraphs [19] introduced in [8]. Since bigraphs are one of the foremost graphical languages for nominal calculi, the resulting logic can express some structural properties of e.g. π-calculus specifications. The resulting bigraphs logic is however quite different from our proposal, which is instead of the tradition of Courcelle's monadic second order logic [12]. In particular, the approach presented in [19] is a static spatial logic that does not consider temporal connectives. An extension to a dynamic bigraph logic by the same authors is under development.

Our graph-based approach was introduced by describing first the encoding of (possibly recursive) processes of the π-calculus [15] and then an algorithm for the verification on such representations of spatial properties expressed by recursion-free formulae [16]. The present paper proposes an encoding of (possibly recursive) formulae in a spatial logic for processes into formulae in a modal graph logic. Our encoding is sound and complete: a process verifies a spatial formula exactly when its graphical representation verifies the translated formula.

The main novelty of this work is the modal graph logic we introduce. Indeed, at first we tried to obtain an encoding of spatial formulae using an existing graph logic. However, the approaches we are aware of turned out not to be expressive enough. For instance, the logics reported in [1,12,21] do not properly model notions like freshness, while the spatial logic for graphs in [5] is static and does not consider the temporal dimension. We have thus devised a graph logic equipped with a modal operator that captures the names of those items involved in a graph transformation and that ensures that the new items are fresh, i.e., different from any item in the formula and in the transformed graph.

Our paper provides a mechanism for specifying spatial formulae on the graphical representation of processes. We believe that our approach offers novel insights into the specification of graph formulae, thanks especially to the link with spatial logics; moreover, it offers further evidence of the adequacy of graph-based formalisms for system design and specification; finally, it suggests a rich and flexible formalism for expressing properties of graph transformations.

The structure of the paper is as follows. Section 2 summarizes the π-calculus and the spatial logic proposed in [3]. Sections 3 and 4 recall a few definitions on graphs with interfaces [9] and their rewriting. Section 5 presents an encoding of π-calculus processes into graphs with interfaces, streamlining the proposal in [15]. Section 6 illustrates a set of graph rewriting rules for simulating process reductions as well as for assisting the encoding. Section 7 defines our modal graph logic, and Section 8 proposes our encoding for spatial formulae.

2 The π-Calculus and a Spatial Logic

This section recalls the basics of one of the foremost calculi for specifying distributed systems, namely the π-calculus [18], and of a logic [3] for expressing spatial properties of a system specified as a process of that calculus.

Definition 1 (processes). *Let $\mathcal{N}$ be a set of* names*; let $\mathcal{X}$ be a set of process variables; and let $\Delta = \{a(b), \overline{a}b \mid a, b \in \mathcal{N}\}$ be the set of* prefix operators*. A* process *P is a term generated by the syntax*

$$P ::= \ \mathtt{0} \quad | \quad (\nu a)P \quad | \quad P \mid P \quad | \quad \delta.P \quad | \quad \delta.x \quad | \quad rec_x.P$$

where $a \in \mathcal{N}$, $x \in \mathcal{X}$ and $\delta \in \Delta$. We denote by $\mathcal{P}$ the set of closed *processes, i.e., such that each process variable x occurs inside the scope of a $\mathrm{rec}_x.-$ operator.*

The standard definition for the set of free names of a process P, denoted by $\mathtt{fn}(P)$, is assumed. Similarly for α-convertibility, with respect to the *restriction* operators $(\nu a)P$ and the *input* operators $b(a).P$: in both cases, the name a is bound in P, and it can be freely α-converted.

Using the definition above, the behavior of a process P is described as a relation over *abstract processes*, i.e., a relation closed under structural congruence.

Definition 2 (structural congruence). *The* structural congruence *for processes is the relation $\equiv \subseteq \mathcal{P} \times \mathcal{P}$, closed under process construction and α-conversion, inductively generated by the following set of axioms*

$$P \mid Q \equiv Q \mid P \quad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \quad P \mid \mathtt{0} \equiv P \quad \mathrm{rec}_x.P \equiv P\{^{rec_x.P}/_x\}$$
$$(\nu a)\mathtt{0} \equiv \mathtt{0} \quad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \quad (\nu a)(P \mid Q) \equiv P \mid (\nu a)Q \text{ for } a \notin \mathtt{fn}(P)$$

As usual, $P\{^Q/_x\}$ denotes process P after the substitution of each free occurrence of process variable x with process Q.

Definition 3 (reductions). *The* reduction relation *for processes is the equivalence relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$, closed under the structural congruence $\equiv$, inductively generated by the following set of axioms and inference rules*

$$\frac{}{a(b).P \mid \overline{a}c.Q \rightarrow P\{^c/_b\} \mid Q} \qquad \frac{P \rightarrow Q}{(\nu a)P \rightarrow (\nu a)Q} \qquad \frac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R}$$

The first rule denotes process communication: process $\overline{a}c.Q$ is ready to communicate the (possibly global) name c along channel a; it then synchronizes with process $a(b).P$, and the local name b is substituted by c on the residual process P (avoiding the capture of name c). The latter rules state the closure of the reduction relation with respect to restriction and parallel composition.

We now recall the spatial logic for the π-calculus presented in [3].

Definition 4 (spatial logic). *Let $V_{\mathcal{N}}$ be a set of* name variables*, and $V_{\mathcal{SF}}$ a set of propositional variables. A* spatial formula *is a term generated by the syntax*

$$\phi ::= \mathtt{tt} \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{0} \mid \phi|\phi \mid \eta \circledR \phi \mid \exists x.\phi \mid \text{И} x.\phi \mid \eta = \eta' \mid \Diamond\phi \mid Z \mid \mu Z.\phi$$

where $\eta, \eta' \in V_{\mathcal{N}} \uplus \mathcal{N}$, $x \in V_{\mathcal{N}}$ and $Z \in V_{\mathcal{SF}}$. We denote by $\mathcal{SF}$ the set of well-formed *formulae, i.e., such that each propositional variable Z occurs inside the scope of a $\mu Z.-$ operator and an even number of negation operators and each name variable x occurs inside the scope of a name quantifier $\exists x.-$ or И$x.-$.*

Boolean connectives and fixpoints have the usual meaning; $\mathbf{0}$ characterizes processes that are structurally congruent to the empty process; $\phi_1|\phi_2$ holds for processes that are structurally congruent to the composition of two subprocesses, satisfying ϕ_1 and ϕ_2; $\eta\circledR\phi$ is true for those processes such that ϕ holds after the revelation of name η; $\exists x.\phi$ characterizes processes such that ϕ holds for some name in $\mathcal{N}$; $Ин x.\phi$ holds for a process P if ϕ holds for some name of $\mathcal{N}$ that is *fresh* with respect to P and ϕ; $\eta = \eta'$ requires η and η' to be equal; and $\Diamond\phi$ is satisfied by a process P if P can be reduced into Q and Q satisfies ϕ.

The semantics of a (well-formed) formula is given in terms of the domain $\mathcal{P}_\mathcal{S}$ of *Psets*. A Pset is a family of processes that is closed under structural congruence and name permutations, for all the names outside its *support*. Intuitively, the support for a Pset is a set of names that are relevant for the property satisfied by the family of processes, i.e., such that any permutation of those names outside the support does not affect the property.

Definition 5 (Pset [4]). *Let $\mathcal{Y}$ be a set of processes. Then $\mathcal{Y}$ forms a Pset if it is closed under structural congruence and there exists a finite set of names $N \subset \mathcal{N}$ such that $P\{^a/_b,^b/_a\} \in \mathcal{Y}$ for all $a, b \notin N$ and $P \in \mathcal{Y}$.*

Every Pset $\mathcal{Y}$ has a least support [4, Prop. 4.13], denoted $\mathtt{supp}(\mathcal{Y})$. For instance, the set $\mathcal{P}$ of all processes is a Pset with empty support.

Formulae with open propositional variables are interpreted under an environment $\sigma : V_{\mathcal{SF}} \to \mathcal{P}_\mathcal{S}$. The semantics of $Ин x.\phi$ requires x to be instantiated with a name that is fresh with respect to ϕ and to any process in the Psets to which the open propositional variables of ϕ are mapped, i.e., the name must be different from any name in ϕ or in the least support of $\sigma(Z)$ for any open propositional variable Z in ϕ. Such a set of names is defined as $\mathtt{n}_\sigma(\phi) = \mathtt{n}(\phi) \cup \bigcup_{Z\in\mathtt{fpv}(\phi)} \mathtt{supp}(\sigma(Z))$, where $\mathtt{fpv}(\phi)$ and $\mathtt{n}(\phi)$ denote the set of the free propositional variables of ϕ and the set of names of ϕ, respectively.

Definition 6 (spatial logic semantics). *Let ϕ be a (well-formed) spatial formula and let σ be a mapping for the free propositional variables of ϕ into Psets. The denotation $[\![\phi]\!]_\sigma$, mapping a formula ϕ into a Pset, is defined by structural induction according to the following rules*

$$
\begin{array}{rlrl}
[\![\mathtt{tt}]\!]_\sigma &= \mathcal{P} & [\![a\circledR\phi]\!]_\sigma &= \{P \mid \exists P'.P \equiv (\nu a)P' \textit{ and } P' \in [\![\phi]\!]_\sigma\} \\
[\![\neg\phi]\!]_\sigma &= \mathcal{P} \setminus [\![\phi]\!]_\sigma & [\![\exists x.\phi]\!]_\sigma &= \bigcup_{a\in\mathcal{N}} [\![\phi\{^a/_x\}]\!]_\sigma \\
[\![\phi_1 \vee \phi_2]\!]_\sigma &= [\![\phi_1]\!]_\sigma \cup [\![\phi_2]\!]_\sigma & [\![Ин x.\phi]\!]_\sigma &= \bigcup_{a\notin\mathtt{n}_\sigma(\phi)} ([\![\phi\{^a/_x\}]\!]_\sigma \setminus \{P \mid a \in \mathtt{fn}(P)\}) \\
[\![\mathbf{0}]\!]_\sigma &= \{P \mid P \equiv \mathtt{0}\} & [\![a = b]\!]_\sigma &= \mathcal{P} \textit{ if } a = b \textit{ and } \emptyset \textit{ otherwise} \\
[\![Z]\!]_\sigma &= \sigma(Z) & [\![\mu Z.\phi]\!]_\sigma &= \bigcap\{\mathcal{Y} \mid \mathcal{Y} \subseteq [\![\phi]\!]_{\sigma[^\mathcal{Y}/_Z]}\}
\end{array}
$$

$$
\begin{array}{rl}
[\![\phi_1|\phi_2]\!]_\sigma &= \{P \mid \exists P_1, P_2.P \equiv P_1|P_2 \textit{ and } P_1 \in [\![\phi_1]\!]_\sigma \textit{ and } P_2 \in [\![\phi_2]\!]_\sigma\} \\
[\![\Diamond\phi]\!]_\sigma &= \{P \mid \exists Q.P \to Q \textit{ and } Q \in [\![\phi]\!]_\sigma\}
\end{array}
$$

The restriction on the use of negation actually guarantees each possible function $\lambda Y.[\![\phi]\!]_{\sigma[Y/Z]}$ to be monotonic, so that fixed points are well defined.

3 Graphs and Their Extension with Interfaces

We recall a few definitions concerning (typed hyper-)graphs, and their extension with *interfaces*, referring to [2,9] for a more detailed introduction.

Definition 7 (graphs). *A (hyper-)*graph *is a four-tuple $\langle V, E, s, t\rangle$ where V is the set of nodes, E is the set of edges and $s, t : E \to V^*$ are the source and target functions. A (hyper-)graph morphism is a pair of functions $\langle f_V, f_E\rangle$ preserving the source and target functions, i.e., $f_V \circ s = s \circ f_E$ and $f_V \circ t = t \circ f_E$.*

Henceforth, when not explicitly stated the components of a graph G are assumed to be V_G, E_G, s_G and t_G. We shall consider *typed graphs* [10], i.e., graphs labeled over a structure that is itself a graph.

Definition 8 (typed graphs). *Let T be a graph. A* typed graph *G over T is a graph $|G|$, together with a graph morphism $\tau_G : |G| \to T$. A* morphism *between T-typed graphs $f : G_1 \to G_2$ is a graph morphism $f : |G_1| \to |G_2|$ consistent with the typing, i.e., such that $\tau_{G_1} = \tau_{G_2} \circ f$.*

In the following, a type graph T is chosen. Then, in order to inductively define the encoding for processes, we need to provide operations over typed graphs. The first step is to equip them with "handles" for interacting with an environment.

Definition 9 (graphs with interfaces). *A T-typed* graph with interfaces *(shortly,* GWI*) is a triple $\mathbb{G} = \langle i_G, G, o_G\rangle$, for G a T-typed graph and $i_G : I_G \to G$, $o_G : O_G \to G$ the* input *and* output *T-typed graph morphisms.*

An interface graph morphism *$f : \mathbb{G} \Rightarrow \mathbb{H}$ is a triple of graph morphisms $\langle f_I, f, f_O\rangle$, preserving the input and output morphisms.*

The category of T-typed graphs with interfaces is denoted by I-T-**Graph**. We let $I \xrightarrow{i} G \xleftarrow{o} O$ denote a graph (of *body* G) with *input* interface I and *output* interface O. With an abuse of notation, we sometimes refer to the image of the input and output morphisms as inputs and outputs, respectively. An interface graph morphism is *interface preserving* if it preserves node identity on interfaces.

In order to define our process encoding, we introduce two operators on graphs with *discrete* interfaces (GWDIs), i.e., such that their set of edges is empty.

Definition 10 (two operators). *Let $\mathbb{G} = I \xrightarrow{i} G \xleftarrow{j} J$ and $\mathbb{G}' = J \xrightarrow{j'} G' \xleftarrow{o} O$ be* GWDI*s. Then, their* sequential composition *is the* GWDI *$\mathbb{G} \circ \mathbb{G}' = I \xrightarrow{i'} G'' \xleftarrow{o'} O$, for G'' the disjoint union $G \uplus G'$, modulo the equivalence on nodes induced by $j(x) = j'(x)$ for all $x \in N_J$, and i', o' the uniquely induced arrows.*

Let $\mathbb{G} = I \xrightarrow{i} G \xleftarrow{o} O$ and $\mathbb{H} = I' \xrightarrow{i'} H \xleftarrow{o'} O'$ be GWDI*s with compatible interfaces.*[1] *Then, their* parallel composition *is the* GWDI *$\mathbb{G} \otimes \mathbb{H} = (I \cup I') \xrightarrow{i''} G' \xleftarrow{o''} (O \cup O')$, for G' the disjoint union $G \uplus H$, modulo the equivalence on nodes induced by $o(y) = o'(y)$ for all $y \in N_O \cap N_{O'}$ and $i(y) = i'(y)$ for all $y \in N_I \cap N_{I'}$, and i'', o'' the uniquely induced arrows.*

[1] That is, any node in $N_I \cap N_{I'}$ has the same type in I and I' (similarly for $N_O \cap N_{O'}$).

With an abuse of notation, the set-theoretic operators are defined component-wise, and the typing morphism is extended accordingly. Intuitively, the sequential composition $\mathbb{G} \circ \mathbb{G}'$ is obtained by taking the disjoint union of the bodies of $\mathbb{G}$ and $\mathbb{G}'$, and gluing the outputs of $\mathbb{G}$ with the corresponding inputs of $\mathbb{G}'$. The parallel composition $\mathbb{G} \otimes \mathbb{H}$ is obtained by taking the disjoint union of the bodies of $\mathbb{G}$ and $\mathbb{H}$, additionally gluing the inputs (outputs) of $\mathbb{G}$ with the corresponding inputs (outputs) of $\mathbb{H}$. The operations are concretely defined, modulo the choice of canonical representatives for the set-theoretic operations: the result is independent of such a choice, up-to isomorphism of the body graphs.

A *graph expression* is a term over the syntax containing all graphs with discrete interfaces as constants, and parallel and sequential composition as binary operators. An expression is *well-formed* if all occurrences of those operators are defined for the interfaces of their arguments, according to Definition 10; its interfaces are computed inductively from the interfaces of the graphs occurring in it, and its *value* is the graph obtained by evaluating all operators in it.

4 Rewriting Graphs with Interfaces

This section recalls the basics of the double-pushout (DPO) approach to graph transformation, as presented in [11,13]. More precisely, it directly introduces the extension of the approach to GWIs, which is needed for our modeling purposes.

Definition 11 (graph production). *A* graph production *is a pair of arrows* $\langle l : \mathbb{K} \to \mathbb{L}, r : \mathbb{K} \to \mathbb{R} \rangle$ *in* I-T-**Graph** *such that* l *is monic and* r *is injective on interfaces. A* T*-typed* graph transformation system *(*GTS*)* $\mathcal{G}$ *is a tuple* $\langle T, P, \pi \rangle$ *where* T *is the type graph,* P *is a set of production names and* π *is a function mapping each name to a* T*-typed production.*

A production $\pi(p)$ is often denoted by a *span* $\mathbb{L} \xleftarrow{l} \mathbb{K} \xrightarrow{r} \mathbb{R}$, and indicated just by the name p. For l to be monic means that all its components are injective.

Definition 12 (derivation). *Let* $p : \mathbb{L} \xleftarrow{l} \mathbb{K} \xrightarrow{r} \mathbb{R}$ *be a* T*-typed production and* $\mathbb{G}$ *a* T*-typed* GWI*. A* match *of* p *in* $\mathbb{G}$ *is a monic* $m_L : \mathbb{L} \to \mathbb{G}$*. A* direct derivation *from* $\mathbb{G}$ *to* $\mathbb{H}$ *via production* p *at a match* $m_{\mathbb{L}}$ *is a diagram as depicted in Fig. 4, where (1) and (2) are actually pushout squares in* I-T-**Graph***. We thus write* $p/m : \mathbb{G} \Longrightarrow \mathbb{H}$*, for* m *the morphism* $\langle m_L, m_K, m_R \rangle$*, or simply* $\mathbb{G} \Longrightarrow \mathbb{H}$*.*

Let p be a production, let $p/m : \mathbb{G} \Longrightarrow \mathbb{H}$ be a direct derivation and let $tr(p/m)$ be the partial function $r^* \circ (l^*)^{-1} : \mathbb{G} \to \mathbb{H}$. By construction, $tr(p/m)$ is injective on interfaces. The derivation p/m is *interface preserving* if $tr(p/m)$ is so. Given a sequence of derivations $p_i/m_i : \mathbb{G}_i \Longrightarrow \mathbb{H}_{i+1}$, we call it *unambiguous* whenever $G_i = G_j$ implies $i = j$, i.e., if a graph never occurs twice in a computation.

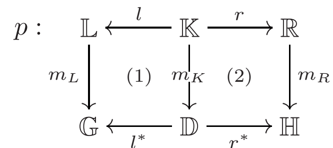


Fig. 1. A direct derivation

In the rest of the paper we implicitly restrict our attention to unambiguous sequences of interface preserving derivations.

5 From Processes to Graphs

We now present an encoding of π-calculus processes into graphs with interfaces, based on the encoding introduced in [15].

The type graph T_π is defined in Fig. 2. Note that all edges have at most one node in the source, connected by an incoming tentacle; the nodes in the target list are instead always enumerated clock-wise, starting from the only incoming tentacle, unless otherwise specified by an enumerating label. For example, the edge ν has the node $\bullet$ as source, and the node $\circ$ as target. The edge *op* actually stands as a concise representation for two edges, namely *in* and *out*, with the same source and target: they have the node $\bullet$ as source and the node list $\langle \bullet, \circ, \circ \rangle$ as target, further specified by the enumerating labels 0, 1, and 2.

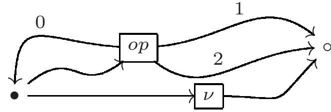


Fig. 2. The graph T_π ($op \in \{in, out\}$)

The type graph is used to model processes syntactically, and our encoding corresponds to the usual construction of the tree associated to a term of an algebra: names are interpreted as variables, so that they are mapped to leaves of the tree and can be safely shared. Intuitively, a tree with a node of type $\bullet$ as root corresponds to a process, whilst each node of type $\circ$ basically represents a name. Clearly, the operators *in* and *out* simulate the input and output prefixes, respectively; and operator ν stands for restriction. Furthermore, note that there is instead no explicit operator accounting for parallel composition.

A class of graphs is now needed, such that all processes can be encoded into a graph expression. Let $p \notin \mathcal{N}$: our choice is depicted in Fig. 3, for all $a, b \in \mathcal{N}$.

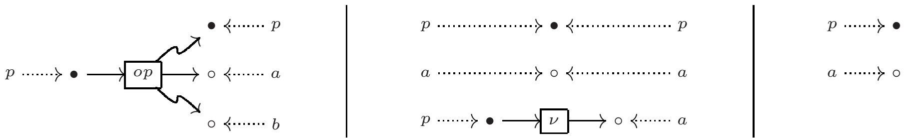


Fig. 3. Graphs $op_{a,b}$ ($op \in \{in, out\}$); id_p, id_a, and ν_a; $\mathtt{0}_p$ and $\mathtt{0}_a$

Finally, let id_Γ and $\mathtt{0}_\Gamma$ be a shorthand for $\bigotimes_{a \in \Gamma} id_a$ and $\bigotimes_{a \in \Gamma} \mathtt{0}_a$, respectively, for a finite set of names $\Gamma \subset \mathcal{N}$. The encoding of finite processes into GWDIs, mapping each finite process into a graph expression, is presented below.

Definition 13 (encoding for finite processes). *Let P be a finite process, and let Γ be a set of names, such that $\mathtt{fn}(P) \subseteq \Gamma$. The* process encoding $[\![P]\!]_\Gamma$, *mapping a process P into a* GWDI, *is defined by structural induction according to the following rules (where $\{c\} \uplus \Gamma$ implies that $c \notin \Gamma$)*

$$
\begin{array}{ll}
[\![(\nu a)P]\!]_\Gamma = \begin{cases} [\![P]\!]_\Gamma & \textit{if } a \notin \mathtt{fn}(P) \\ (id_p \otimes \nu_c \otimes id_\Gamma) \circ [\![P\{^c/_a\}]\!]_{\{c\} \uplus \Gamma} & \textit{otherwise} \end{cases} & \\
[\![P \mid Q]\!]_\Gamma = [\![P]\!]_\Gamma \otimes [\![Q]\!]_\Gamma & [\![a(b).P]\!]_\Gamma = (in_{a,c} \otimes id_\Gamma) \circ [\![P\{^c/_b\}]\!]_{\{c\} \uplus \Gamma} \\
[\![\mathtt{0}]\!]_\Gamma \quad = \mathtt{0}_p \otimes \mathtt{0}_\Gamma & [\![\overline{a}b.P]\!]_\Gamma \quad = (out_{a,b} \otimes id_\Gamma) \circ [\![P]\!]_\Gamma
\end{array}
$$

Note the conditional rule for $(\nu a).P$: it is required for removing the occurrence of useless restriction operators, i.e., those that bind a name not occurring in the process. The mapping is well-defined, since the resulting graph expression is well-formed, and the encoding $[\![P]\!]_\Gamma$ is a graph with interfaces $(\{p\} \cup \Gamma, \emptyset)$.

The mapping $[\![\cdot]\!]$ is not surjective: there are graphs with interfaces $(\{p\} \cup \Gamma, \emptyset)$ that are not (isomorphic to) the image of any process. Nevertheless, our encoding is sound and complete, as stated by the proposition below (adapted from [15]).

Proposition 1 (correct process encoding). *Let P, Q be finite processes and Γ a set of names such that $\mathtt{fn}(P) \cup \mathtt{fn}(Q) \subseteq \Gamma$. Then, $P \equiv Q$ iff $[\![P]\!]_\Gamma \cong [\![Q]\!]_\Gamma$.*

The notation $\mathbb{G} \cong \mathbb{H}$ indicates the existence of an interface preserving isomorphism between the GWDIs. That notion is used to tackle recursive processes.

Definition 14 (colimits of ω-chain). *Let $\omega = \mathbb{G} = \mathbb{G}_0 \rightarrow \mathbb{G}_1 \rightarrow \mathbb{G}_2 \ldots$ be a chain of injective, interface preserving morphisms. Then, the colimit $\mathtt{col}(\omega)$ is a GWI $\mathbb{H}$ and a family $f_i : \mathbb{G}_i \rightarrow \mathbb{H}$ of injective, interface preserving morphisms, making the diagram commute.*

Clearly, a colimit always exists, and it is uniquely defined, up-to $\cong$. In the following, we postulate a choice for colimits. Hence, in order to encode recursive processes as infinite graphs, a colimit construction is performed.

Definition 15 (recursive encoding). *Let $P[x]$ be an open process, such that the single process variable x may occur free in P. Let $\omega_{P[x]} = \{[\![P_i]\!]_\Gamma \mid i \in \mathbb{N}\}$ be the chain such that $P_0 = P[^{\mathtt{0}}/_x]$ and $P_{i+1} = P[^{P_i}/_x]$, with the obvious interface preserving morphisms. Then, $[\![rec_x.P]\!]_\Gamma$ denotes the colimit $\mathtt{col}(\omega_{P[x]})$.*

In other terms, each open process $P[x]$ defines a continuous functor on the GWDIs with interfaces $(\{p\} \cup \Gamma, \emptyset)$, for each set of names Γ such that $\mathtt{fn}(P) \subseteq \Gamma$, and the colimit is thus calculated evaluating the chain in the standard way.

Two recursive processes may be mapped to isomorphic GWDIs, even if they are not structurally congruent. Nevertheless, the extended encoding is still sound.

6 Process Reductions vs. Graph Rewrites

This section introduces two rules for simulating the reduction relation as well as a few rules that are useful for the encoding of the logic. These rules form the GTS $\mathcal{G}_\phi$ over which encoded formulae are interpreted. Note that, since it suffices for our purposes, all the spans involve only graphs with empty output interfaces.

So, let us start with rule p_π (depicted in Fig. 4) for simulating the reduction relation over processes. Let us explain our notation. The nodes may be labeled.

If the label is an element in $\{p\} \cup \mathcal{N}$, that means that the node is actually in the image of the input interface. Otherwise, the label is a natural number, and it is used just for describing the actions performed by the rule, so that e.g. the $\circ$ nodes identified by 2 and 3 are coalesced by the rule. These identifiers are of course arbitrary: they correspond to the actual elements of the set of nodes/interfaces, and they unambiguously characterise the (interface preserving) span of functions.

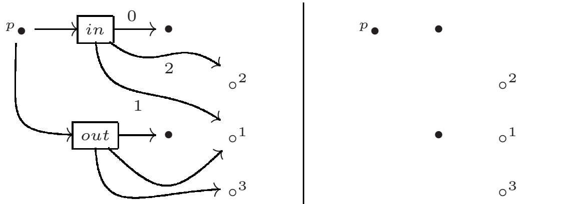


Fig. 4. The rule p_π for synchronization

Another rule p'_π is needed: it is the same as p_π, but with nodes 1 and 3 coalesced. It is noteworthy that two rules suffice to recast the reduction semantics for the π-calculus. The structural rules are taken care of by the embedding of a graph into a larger one, thus simulating the closure of reduction with respect to contexts. Similarly, no instance of the rules is needed, since graph isomorphism takes care of the closure with respect to structural congruence.

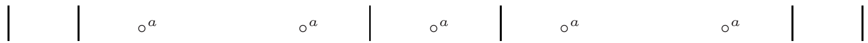


Fig. 5. Rules for introducing (left), checking for (center) and removing a name (right)

Fig. 6. The rule for revealing a restricted name

We now introduce a set of "house-keeping" rules for performing specific tasks requested by our encoding of the spatial logic. The rule p_n for adding nodes to the interface is depicted on the left of Fig. 5. Since the left-most and middle graphs are empty, the rule can be applied to any graph resulting in the addition of a node to the right-most graph. This rule is going to be used in conjunction with rule p_r: it reveals a restricted name (see Fig. 6), consuming a restriction-edge and coalescing the attached node with the image of an interface node.

The identity rule $p_{\exists_i}$, on the center of Fig. 5, tests the presence of a node among the inputs. Finally, the garbage collection rule p_{g_i}, on the right of Fig. 5, removes a name from the interface: note that the DPO formalism ensures that the rule is applied to an isolated node only. Analogous rules p_{g_b} and p_{g_r} are needed for removing isolated nodes that represent names and are not in the image of the input morphism and for removing useless restriction operators, respectively.

7 Modal Graph Logic

This section introduces our flavor of graph logic, inspired by [1,12,21] and resulting in a monadic second order μ-calculus with a first-order action modality. In particular, our logic recalls [1], where a fragment of Courcelle's monadic second order logic [12], combined with the propositional μ-calculus, is considered.

Definition 16 (graph logic syntax). *Let V_Z be a set of propositional variables, V_n a set of node variables, V_e and V_E sets of first and second order edge variables, respectively, and finally $\langle T, P, \pi \rangle$ a* GTS*. The set $\mathcal{GF}$ of* modal graph formulae *over the* GTS *$\langle T, P, \pi \rangle$ is the set of terms generated by*

$$\psi ::= \mathtt{tt} \mid \theta \mid \neg\psi \mid \psi \vee \psi \mid \langle p(\boldsymbol{x}, \boldsymbol{x'})\rangle\psi \mid \exists x.\psi \mid \exists y.\psi \mid \exists Y.\psi \mid Z \mid \mu Z.\psi$$
$$\theta ::= \epsilon = \epsilon \mid y = y \mid \tau(y) = t_e \mid y \in Y \qquad \epsilon ::= \eta \mid i(\eta) \mid s(y) \mid t[k](y)$$

for $k \in \{0, 1, 2\}$, $x \in V_n$, $\eta \in V_n \cup \mathcal{N}$, $y \in V_e$, $\boldsymbol{x}, \boldsymbol{x'} \in V_n^*$, $Y \in V_E$, $Z \in V_Z$, $t_e \in E_T$, and $p \in P$.

For readability sake, in the above definition p indicates an interface preserving rule, and $\boldsymbol{x}$, $\boldsymbol{x'}$ are vectors of node variables indexed over the nodes of the left-hand side $\mathbb{L}$ and of the right-hand side $\mathbb{R}$, respectively. The resulting modal operator $p(\langle x_1, \ldots, x_n\rangle, \langle x'_1, \ldots, x'_m\rangle).\phi$ bounds the $n + m$ variables in ϕ. In the following we consider closed formulae only, i.e., formulae where each occurrence of a node, edge, edge set or propositional variable is bound.

The logic includes booleans, a first-order node quantifier, first and second-order edge quantifiers, a modal operator, fixpoints, and equalities of edge identities or nodes (possibly referred to by node variables), the source or k-th target of an edge, or the images of an input, denoted by η, $s(y)$, $t[k](y)$, and $i(\eta)$ respectively. Note the lack of constraints on the number of tentacles departing from an edge variable, so that a formula as $t[i](e) = x$ might turn out to be always false.

We introduce now the concept of Gsets, sets of GWDIs closed under graph isomorphism and permutations of interface nodes outside its *support*.

Definition 17 (Gset). *Let $\mathcal{Y}$ be a set of* GWDI*s. Then $\mathcal{Y}$ forms a Gset if there exists a finite set of nodes N occurring in the input interface of each graph in $\mathcal{Y}$ and such that $f(\mathbb{G}) \in \mathcal{Y}$ for all* GWDI*s $\mathbb{G} \in \mathcal{Y}$ and graph isomorphisms f preserving the identity of the nodes in N.*

Each Gset $\mathcal{Y}$ has a finite support, denoted by $\mathtt{supp}(\mathcal{Y})$; and (the union of) the encoding of (the members of) a Pset turns out to be a Gset. We let $\mathtt{n}_\rho(\psi)$ denote the set of names of a formula ψ under a valuation ρ, defined as $\mathtt{n}(\psi) \cup \bigcup_{Z \in \mathtt{fpv}(\psi)} \mathtt{supp}(\rho(Z))$, for $\mathtt{fpv}(\psi)$ and $\mathtt{n}(\psi)$ the set of the free propositional variables and the names of a formula ψ (constants and free name variables).

The formulae of the logic are intended to be interpreted over Gsets. We let S denote the family of GWDIs, and $\mathcal{V}$ and $\mathcal{E}$ the sets of all nodes and edge names used in I-T-**Graph**, respectively, indexed by the GWDI they belong to.

Definition 18 (graph logic semantics). *Let ψ be a (closed) modal graph formula and let ρ be an* environment*, i.e., a tuple $\rho = \langle \rho_x, \rho_y, \rho_Y, \rho_Z \rangle$ of mappings $\rho_x : V_x \to \mathcal{V}$, $\rho_y : V_y \to \mathcal{E}$, $\rho_Y : V_Y \to 2^{\mathcal{E}}$ and $\rho_Z : V_Z \to 2^S$ from node, edge, edge set and propositional variables into nodes, edges, edge sets, and Gsets, respectively. The denotation $[\![\psi]\!]_\rho$, mapping a formula ψ into a Gset, is defined by structural induction according to the following rules*

$$
\begin{array}{rlrl}
[\![\mathtt{tt}]\!]_\rho &= S & [\![\exists y.\psi]\!]_\rho &= \{\mathbb{G} \in S \mid \exists e \in E_{\mathbb{G}}.\mathbb{G} \in [\![\psi]\!]_{\rho[e/y]}\} \\
[\![\neg\psi]\!]_\rho &= S \setminus [\![\psi]\!]_\rho & [\![\exists Y.\psi]\!]_\rho &= \{\mathbb{G} \in S \mid \exists E \subseteq E_{\mathbb{G}}.\mathbb{G} \in [\![\psi]\!]_{\rho[E/Y]}\} \\
[\![Z]\!]_\rho &= \rho_Z(Z) & [\![\mu Z.\psi]\!]_\rho &= \bigcap\{\mathcal{Y} \mid \mathcal{Y} \subseteq [\![\psi]\!]_{\rho[^{\mathcal{Y}}/_Z]}\} \\
[\![\theta]\!]_\rho &= \|\rho(\theta)\| & [\![p(\boldsymbol{x}, \boldsymbol{x'})\psi]\!]_\rho &= \{\mathbb{G} \mid p/m : \mathbb{G} \to \mathbb{H} \text{ and } \mathbb{H} \in [\![\psi]\!]_{\rho'}\} \\
[\![\psi_1 \vee \psi_2]\!]_\rho &= [\![\psi_1]\!]_\rho \cup [\![\psi_2]\!]_\rho & [\![\exists x.\psi]\!]_\rho &= \bigcup_{a \in \mathtt{n}_\rho(\psi)} [\![\psi]\!]_{\rho[^a/_x]}
\end{array}
$$

where $\|\rho(\theta)\|$ maps true and false to S_M and $\emptyset$, respectively; and $\rho' = tr^\dagger(p/m) \circ (\rho \cup \{\boldsymbol{x} \mapsto m(N_{\mathbb{L}})\}) \cup \{\boldsymbol{x'} \mapsto m(N_{\mathbb{R}})\}$, for $N_{\mathbb{L}}$ and $N_{\mathbb{R}}$ the nodes of the left-hand and the right-hand side of the rule p and $tr^\dagger(p/m)$ the total extension of $tr(p/m)$.

The total extension $tr^\dagger(p/m)$ evaluates a node n to $tr(p/m)(n)$ if n belongs to the domain of $tr(p/m)$, and to n otherwise. Intuitively, the variables in $\boldsymbol{x}$ are assigned to the matched items[2] of the left-hand side of the derivation, and the resulting mapping is composed with the trace of the derivation to get rid of item renaming (no renaming is needed instead for $\boldsymbol{x'}$). In addition, we require the new interface items in $\mathbb{H}$ to be different from those occurring in $\mathtt{n}_\rho(\psi)$: this ensures the new items to be fresh with respect to the formula and its environment.

Boolean connectives and item comparisons have the expected meaning, and, since the denotation is for closed formulae, the interpretation of the terms generated by θ is obvious. Note however that as in [1] we consider environments ρ that might map variables into items that are indexed over the graph they belong. Thus, a formula like $x = y$ is satisfied by a graph in environment ρ if $\rho(x) = \rho(y)$, independently on whether or not $\rho(x)$ or $\rho(y)$ are nodes of the graph.

The main difference with the approach introduced in [1] is the semantics of the modal operator. Indeed, in order for the formula $p(\boldsymbol{x}, \boldsymbol{x'})\psi$ to hold in an environment ρ we require the existence of a direct derivation from $\mathbb{G}$ into a graph $\mathbb{H}$ via rule p and match m, such that (1) new items are fresh with respect to $\mathtt{n}_\rho(\psi)$; and (2) $\mathbb{H}$ fulfills ψ in an environment ρ' that is obtained from ρ with the application of the trace of the derivation p/m (to get rid of item renaming) and with the addition of the mapping of the variables occurring in the vectors $\boldsymbol{x}$, $\boldsymbol{x'}$ to the items of the left-hand and right-hand side of the match m of rule p. In that way, one can express not only the possibility of applying a graph transformation rule, but we can bind variables with the items involved in the transformation which we can use in the residual formula.

8 From Spatial to Graph Logic

We can now finally turn our attention to the encoding $[\cdot] : \mathcal{SF} \to \mathcal{GF}$, mapping spatial formulae into graph formulae over the GTS $\mathcal{G}_\phi$ defined in Section 6. Our

[2] Abusing of notation we assume here that such items are ordered for each rule.

$$\begin{aligned}
\exists y \in Y.\psi &\equiv \exists y.(y \in Y \wedge \psi) \\
in(x,y) &\equiv s(y) = x \vee t[0](y) = x \vee t[1](y) = x \vee t[2](y) = x \\
I(x) &\equiv \neg \exists y.in(x,y) \\
C(Y) &\equiv \forall y.(y \in Y \rightarrow i(p) = s(y) \vee \exists y' \in Y.s(y') = t[0](y)) \\
R(Y,Y') &\equiv \forall y.(y \in Y \wedge \tau(y) = \nu \rightarrow \neg \exists y' \in Y'.in(t[0](y),y')) \\
P(Y,Y') &\equiv \forall y.(y \in Y \leftrightarrow y \notin Y') \\
\langle\langle p(\boldsymbol{x_1},\boldsymbol{x_1'})\rangle\rangle\psi &\equiv \langle p(\boldsymbol{x},\boldsymbol{x'})\rangle(\boldsymbol{x} = \boldsymbol{x_1} \wedge \boldsymbol{x'} = \boldsymbol{x_1'} \wedge \psi\{^{\boldsymbol{x}}/_{\boldsymbol{x_1}}, ^{\boldsymbol{x'}}/_{\boldsymbol{x_1'}}\})
\end{aligned}$$

Fig. 7. Auxiliary graph formulae

goal is to define a complete and sound encoding such that for any process P we have that $P \in [\![\phi]\!]$ iff $[\![P]\!]_{\mathtt{fn}(P)} \in [\![[\phi]]\!]$.

For the sake of readability, for each modal operator we consider only those arguments that are relevant for the encoding. So, $p_\pi(x_1, x_2)$ (resp. $p'_\pi(x)$) binds x_1 and x_2 (resp. x) with the items 1 and 2 of the left-hand side of the rule p_π (resp. p'_π) depicted in Fig. 4, i.e., the channel on which synchronization occurs and the sent name (and the same occurs for p'). These nodes are relevant for the encoding since they might become isolated and thus need to be garbage collected. Similarly, $p_n(x_a)$ binds x_a with the item a of the right-hand side of the rule p_n (see Fig. 5, right), i.e., the new interface node; $p_{\exists_i}(x_a)$ binds x_a with the item a of the rule $p_{\exists_i}$ (see Fig. 5, center), i.e., the checked for interface node $p_r(x_a)$ binds x_a with the item a of the right-hand side of rule p_r (see Fig. 5, right), i.e., of the deleted interface node; and $p_g(x_a)$ binds x_a with the item a of the right-hand side of rule p_g (see Fig. 6), i.e., the revealed interface node.[3]

Fig. 7 summarizes some additional abbreviations that provide a more readable and concise presentation of the encoding. First, as a shorthand, $\exists y \in Y.\psi$ quantifies over the edges of an edge set Y, while $in(x, y)$ is a shorthand for the formula expressing the occurrence of the node x in either the source or the target of edge y. Since the type graph considers at most three targets, the formula considers only up to the third target. Formula $I(x)$ is used to identify useless nodes. It states that x is not the source or target of any edge, thus characterizing isolated nodes. Another property is that a set of edges (in an acyclic graph, as those representing processes) is connected: In words, $C(Y)$ requires each edge of set Y to occur consecutively to another edge of Y unless it has the root of the graph (the image of p) as source. Then $R(Y, Y')$ states the confinement of the target of a restriction operator, i.e., the target of each ν edge of Y cannot be used in Y'. We also use a formula $P(Y, Y')$ to express that two sets of edges Y, Y' are disjoint and complementary, i.e., they partition the set of edges. Another abbreviation is that we sometimes want to express the fact that a rule p can be applied for a certain match, denoted by $\langle\langle p(\boldsymbol{x_1}, \boldsymbol{x_1'})\rangle\rangle\phi$.

Furthermore, $\{\psi\}^Y$ denotes the formula ψ relativized to the set of edges Y, i.e., $\{\exists y.\psi\}^Y \equiv \exists y \in Y.\{\psi\}^Y$, $\{\exists Y'.\psi\}^Y \equiv \exists Y'.(\forall y.y \in Y' \rightarrow y \in Y) \wedge \{\psi\}^Y$,

[3] Note that, except for the reduction rules $p_\pi(x_1, x_2)$ and $p'_\pi(x)$ and for the rules $p_{g_b}(x)$ and $p_{g_r}(x_1, x_2)$ performing a sort of garbage collection, all the operators bind interface nodes, since they are used for checking name properties.

$\{\exists x.\psi\}^Y \equiv \exists x.\exists y \in Y.in(i(x), y) \wedge \{\psi\}^Y$ and $\{\langle p(\boldsymbol{x}, \boldsymbol{x'})\rangle.\psi\}^Y \equiv \langle p(\boldsymbol{x}, \boldsymbol{x'})\rangle(\bigwedge_{x\in\boldsymbol{x}} \exists y \in Y.(in(x, y) \vee in(i(x), y)) \wedge \{\psi\}^Y)$, just to define the most significant cases for ψ (the rest are recursively defined in a straightforward way).

We finally present our encoding of spatial formulae into graph formulae.

Definition 19 (logics encoding). *Let ϕ be a spatial formula. The* logics encoding $[\phi]$, *mapping a spatial formula ϕ into a graph formula, is defined by structural induction according to the rules in Fig. 8.*

The encoding of boolean connectives (b1) and fixpoints (μ1) is trivial.

Regarding the encoding of name equalities, it is worth noticing that the encoding works with interface nodes rather than with their images. Because the input morphism is injective in any process encoding, we can safely encode name comparison as the comparison of the corresponding interface nodes (n1).

$$
\begin{aligned}
[\mathbf{0}] &= \forall y.F && (v1)\\
[T] &= T \qquad [\neg\phi] = \neg[\phi] \qquad [\phi_1 \vee \phi_2] = [\phi_1] \vee [\phi_2] && (b1)\\
[\eta_1 = \eta_2] &= (\eta_1 = \eta_2) && (n1)\\
[Z] &= Z \qquad [\mu Z.\phi] = \mu Z.[\phi] && (\mu 1)\\
[\text{И}x.\phi] &= \langle p_n(x)\rangle[\phi] && (f1)\\
[\exists x.\phi] &= \quad [\text{И}x.\phi] && (e1)\\
&\vee \langle p_{\exists_i}(x)\rangle[\phi] && (e2)\\
&\vee \exists x.[\phi] && (e3)\\
[\eta \circledR \phi] &= \quad (\langle\langle p_{\exists_i}(\eta)\rangle\rangle I(i(\eta)) \wedge \langle\langle p_r(\eta)\rangle\rangle[\phi]) && (r1)\\
&\vee (\neg\langle\langle p_{\exists_i}(\eta)\rangle\rangle T \wedge \langle p_n(x')\rangle\langle\langle p_r(x')\rangle\rangle[\phi\{^{x'}/_{\eta}\}]) && (r2)\\
[\Diamond\phi] &= \langle p_\pi(x, x')\rangle G(x, x', [\phi]) \;\vee\; \langle p'_\pi(x)\rangle G(x, x, [\phi]) && (a1)\\
[\phi_1|\phi_2] &= \exists Y.\exists Y'.P(Y, Y') && (c1)\\
&\qquad \wedge C(Y) \wedge C(Y') && (c2)\\
&\qquad \wedge R(Y, Y') \wedge R(Y', Y) && (c3)\\
&\qquad \wedge \{[\phi_1]\}^Y \wedge \{[\phi_2]\}^{Y'} && (c4)
\end{aligned}
$$

Fig. 8. The encoding of spatial formulae into graph formulae

The encoding of an empty process is the graph $\mathtt{0}_p$ depicted in Fig. 3, i.e., it is a graph with just one node and no edges. Moreover, no other GWDI modeling a process has an empty set of edges. Thus, the encoding of **0** is a graph formula that characterizes graphs without edges (v1).

The encoding of the freshness quantifier exploits Gabbay-Pitts property [14]: it suffices to consider just one fresh name, neither occurring in ϕ nor previously in the process. We obtain such a name via the freshness rule p_n, which binds the variable x as the fresh name it is introduced. The rule ensures that x will be effectively fresh for the process and the formula (f1).

Also the encoding of $\exists x.\phi$ relies on Gabbay-Pitts property. Indeed, to check if ϕ holds for some name x it suffices to consider (e1) a fresh name (thus relying on the encoding of freshness quantification), (e2) all the free names of the process (the nodes of its interface) and (e3) the nodes in $n_\sigma([\phi])$ (i.e., all the names of ϕ plus the least support of $\sigma(Z)$ for any open propositional variable Z in ϕ).

The encoding of $\eta \circledR \phi$ distinguishes two cases: either (r1) the node η occurs in the interface and its image is isolated or (r2) it is not in the interface. The first turns out to be true when η has been introduced by the application of the freshness rule p_r. In other words, $\eta \circledR \phi$ was nested in a freshness quantification on η (which is a variable rather than a constant). In this case the encoding considers the revelation of a restricted node as η using rule p_r. If η does not occur in the interface, then η is not a free name, hence, we introduce it in the interface via rule p_r and proceed as in the first case.

The action modality requires either the rules p_π or the rule p'_π to be applicable. The resulting graph must then satisfy ϕ, but we may need to garbage collect those nodes involved in the synchronization, i.e., names and restrictions might become useless. This is performed by the formula denoted by G: its presentation is neglected, since it is both intuitive and lengthy, as it consists of two large disjunctions where each disjunct corresponds to one different case depending on whether the synchronization name is free/bound and whether the synchronization and communicated names result used/useless/uselessly restricted. For instance, if after the application of p_π the node x (the synchronization channel) becomes isolated (i.e, useless), we have to apply either the rule p_{g_i} or the rule p_{g_b} (depending if x is an input) before evaluating the encoding ϕ: the resulting formula is $\langle\langle p_{g_i}(x)\rangle\rangle[\phi] \vee \langle\langle p_{g_b}(x)\rangle\rangle[\phi]$.

Finally, consider the encoding of composition. The encoding of the parallel composition P of two processes is done via the parallel composition $\otimes$ of the corresponding graphical encodings. The resulting graph $\llbracket P \rrbracket$ is a tree with the image of p as root, from where several edges depart. Some of them represent subprocesses and the rest correspond to name restrictions. Thus, the encoding of $\phi_1 | \phi_2$ is a graph formula that states whether there is a *correct* decomposition of a graph into two components, one satisfying ϕ_1 and the other satisfying ϕ_2 (c4). A correct decomposition requires (c1) to find two complementary and mutually disjoint sets of edges; each set must form (c2) a connected graph including at least an edge whose source is the image of p; and (c3) any restriction edge has to belong to the right set.

The theorem below states that the proposed encoding is correct.

Theorem 1. *Let P be a process and let ϕ be a closed spatial formula. Then, $P \in \llbracket \phi \rrbracket$ iff $\llbracket P \rrbracket_{\mathtt{fn}(P)} \in \llbracket [\phi] \rrbracket$.*

9 Conclusions and Future Work

We have extended our graph-based technique for the verification of spatial properties of π-calculus specifications. In previous works we proposed a graphical representation of the π-calculus [15] and an algorithm for the verification of (recursion-free) spatial properties of finite processes in such a graphical representation [16]. In the present paper we tackled possibly recursive processes and formulae. However, instead of providing a new algorithm, we defined an

encoding of spatial logic [3] into a modal graph logic. Our first intention was to reuse existing logics and tools for the verification of graph transformation systems, but the approaches we considered turned out not to be sufficiently expressive.

This lack of tools gave rise to the need of a new modal graph logic. With respect to other approaches [1,5,12,21], the main novelty of our logic is a modal operator that binds variables with the items involved in a graph derivation and, in addition, ensures the items created by the rule to be new with respect to the environment in which the formula is interpreted. This operator generalizes node quantification and this is the key to encode spatial ingredients like the revelation of restricted nodes and the creation of fresh names.

Our approach enjoys indeed a few benefits. Besides being intuitively appealing, the graphical presentation offers canonical representatives for abstract processes, since two processes are structurally congruent exactly when they are mapped to the same graph with interfaces (up to interface preserving isomorphism). The encoding has a unique advantage with respect to most of the approaches to the graphical implementation of calculi with name mobility (such as *bigraphs* [19]): it allows for the reuse of standard graph transformation theory and tools for simulating the reduction semantics of the calculus [15].

In addition, the proposed graph logic is very expressive and flexible, as it is parametric with respect to the graph transformation system under consideration. We believe that this can easily provide encodings for various spatial logics and process calculi (in suitable graphical representation). Of course, the model checking problem for our logic is undecidable in general, as the encoded problem of model checking the spatial logic of [3] is decidable only for π-calculus specifications with bounded processes. We shall, thus, identify interesting decidable fragments of our logic and study implementation strategies for some verification problems. The main difficulty relies in the representation of recursive processes as infinite graphs, but we might use the alternative representation of recursion by means of so-called *process expression*, since constant invocations can be encoded as graph derivations [15]. The path was not chosen here since it results in a more cumbersome encodings of both the processes and the logic.

Nevertheless, setting aside any consideration on the efficiency and usability of our approach, we believe that a main contribution of our paper is a further showcase of the usefulness of graphical techniques as a unifying, intuitive, suitable and flexible formalism for the design and validation of concurrent systems.

References

1. Baldan, P., Corradini, A., König, B., Lluch Lafuente, A.: A temporal graph logic for verification of graph transformation systems. In: Fiadeiro, J.L., Schobbens, P.-Y. (eds.) Proceedings of WADT '06 (Workshop on Algebraic Development Techniques). LNCS, vol. 4409, pp. 1–20. Springer, Heidelberg (2007)
2. Bruni, R., Gadducci, F., Montanari, U.: Normal forms for algebras of connections. Theor. Comp. Sci. 286(2), 247–292 (2002)

3. Caires, L.: Behavioral and spatial observations in a logic for the π-calculus. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 72–87. Springer, Heidelberg (2004)
4. Caires, L., Cardelli, L.: A spatial logic for concurrency (part I). Information and Computation 186(2), 194–235 (2003)
5. Cardelli, L., Gardner, P., Ghelli, G.: A spatial logic for querying graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 597–610. Springer, Heidelberg (2002)
6. Cardelli, L., Fiore, M., Winskel, G.: Manipulating trees with hidden labels. In: Gordon, A.D. (ed.) ETAPS 2003 and FOSSACS 2003. LNCS, vol. 2620, pp. 216–232. Springer, Heidelberg (2003)
7. Cardelli, L., Gordon, A.D.: Ambient logic. Forthcoming issue of Mathematical Structures in Computer Science (2007)
8. Conforti, G., Macedonio, D., Sassone, V.: Spatial logics for bigraphs. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 766–778. Springer, Heidelberg (2005)
9. Corradini, A., Gadducci, F.: An algebraic presentation of term graphs, via gs-monoidal categories. Applied Categorical Structures 7(4), 299–331 (1999)
10. Corradini, A., Montanari, U., Rossi, F.: Graph processes. Fundamenta Informaticae 26(3/4), 241–265 (1996)
11. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation I: Basic concepts and double pushout approach. In [24], pp. 163–245
12. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In [24], pp. 313–400
13. Drewes, F., Habel, A., Kreowski, H.-J.: Hyperedge replacement graph grammars. In [24], pp. 95–162
14. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing 13(3-5), 341–363 (2002)
15. Gadducci, F.: Term graph rewriting and the π-calculus. In: Ohori, A. (ed.) APLAS 2003. LNCS, vol. 2895, pp. 37–54. Springer, Heidelberg (2003)
16. Gadducci, F., Lluch Lafuente, A.: Graphical verification of a spatial logic for the π-calculus. In: Heckel, R., König, B., Rensink, A. (eds.) Graph Transformation for Verification and Concurrency. El. Notes in Theor. Comp. Sci, Elsevier, Amsterdam (2007)
17. Kozioura, V., König, B.: AUGUR: A tool for the analysis of graph transformation systems. Bulletin of EATCS 87, 126–137 (2005)
18. Milner, R.: Communicating and Mobile Systems: The π-calculus. Cambridge University Press, Cambridge (1992)
19. Milner, R.: Bigraphical reactive systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 16–35. Springer, Heidelberg (2001)
20. Rensink, A.: The GROOVE simulator: A tool for state space generation. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 479–485. Springer, Heidelberg (2004)
21. Rensink, A.: Towards model checking graph grammars. In: Leuschel, M., Gruner, S., Lo Presti, S., (eds.) Automated Verification of Critical Systems, vol. DSSE–TR–2003–2 of University of Southampton Technical Reports, pp. 150–160. University of Southampton (2003)
22. Reif, J., Sistla, A.P.: A multiprocess network logic with temporal and spatial modalities. Journal of Computer and System Sciences 30(1), 41–53 (1985)

23. Reynolds, J.: Separation logic: A logic for shared mutable data structures. In: Logic in Computer Science, pp. 55–74. IEEE Computer Society Press, Los Alamitos (2002)
24. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1, World Scientific (1997)
25. Varró, D.: Automated formal verification of visual modeling languages by model checking. Software and Systems Modeling 3(2), 85–113 (2004)

Higher Dimensional Trees, Algebraically

Neil Ghani[1] and Alexander Kurz[2,⋆]

[1] University of Nottingham
[2] University of Leicester

Abstract. In formal language theory, James Rogers published a series of innovative papers generalising strings and trees to higher dimensions. Motivated by applications in linguistics, his goal was to smoothly extend the core theory of the formal languages of strings and trees to higher dimensions.

Rogers' definitions rely on a specific representation of higher dimensional trees. This paper presents an alternative approach which focusses more on their universal properties and is based upon category theory, algebras, coalgebras and containers. Our approach reveals that Rogers' trees are canonical constructions which are also particularly beautiful. We also provide new theoretical results concerning higher dimensional trees. Finally, we provide evidence for our devout conviction that clean mathematical theories provide the basis for clean implementations by indicating how our abstract presentation will make computing with higher dimensional trees easier.

1 Introduction

Strings occur in the study of formal languages where they are used to define complexity classes such as those of regular expressions, context free languages, context sensitive languages etc. Trees also play a multitude of different roles and are often thought of as 2-dimensional strings. For instance, there is a clear and well defined theory of tree automata, of tree transducers and other analogues of string-theoretic notions. Indeed, the recent interest in XML and its focus on 2-dimensional data has brought the formal language theory of trees to a wider audience.

In a series of innovative papers (see [11] and references therein), James Rogers asked how one can formalise, and hence extend, the idea that trees are two-dimensional strings to higher dimensions. The desire to *go up a dimension* is very natural - for example a parser will turn a string into a tree. Thus higher dimensional trees will certainly arise when parsing 2-dimensional trees and, more generally, when trees are considered not as part of the meta-theory of the formal languages of strings, but as objects worthy of their own study. Rogers came from a background in both formal languages and natural languages and his motivation to study higher dimensional trees was rooted in the use of the latter to study the former. For example, his paper discusses applications to Tree Adjoining Grammars, Government Binding Theory, and Generalised Phrase Structure Grammars.

Rogers' work was highly imaginative and he certainly had great success in generalising formal language theory from strings and trees to higher dimensions. However, his approach to higher dimensional trees is very concrete. For example, Rogers defines

⋆ Partially supported by EPSRC EP/C014014/1.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 226–241, 2007.
© Springer-Verlag Berlin Heidelberg 2007

a tree as a *tree domain*, ie a set of paths satisfying the left-sibling and ancestor properties. Similarly, he defines higher dimensional trees to be sets of higher dimensional paths satisfying higher dimensional versions of the ancestor and left-sibling properties. These conditions are notationally quite cumbersome at the two dimensional level and this complexity is magnified at higher dimensions. This has practical consequences as it is our belief that clean mathematical foundations are required for clean implementations of both higher dimensional trees as data structures and the algorithms which manipulate them. In particular, implementing higher dimensional trees as higher dimensional tree domains involves the (potential) requirement to regularly verify that algorithms preserve the well-formedness condition of the set of higher dimensional paths in a higher dimensional tree domain.

We provide a more abstract treatment of higher dimensional trees where the fundamental concept is not the path structure of tree domains but rather the notion of fixed point and initial algebra. When viewed through this categorical prism, Rogers' definitions and constructions become very succinct and elegant. This is a tribute to both the sophistication of category theory in capturing high level structure and also to Rogers' insight in recognising these structures as being of fundamental mathematical and computational interest. The overall contributions of this paper are thus as follows:

- We provide a categorical reformulation of the definition of Rogers higher dimensional trees. Remarkably, the central construction in our reformulation is the hitherto unused quadrant of the space whose other members are the free monad, the completely iterative monad, and the cofree comonad.
- To demonstrate that this research has both practical as well as theoretical insight, we use this reformulation to show that classical results of Arbib and Manes on 'Machines in a category' apply to higher-dimensional automata. In particular, this gives procedures of determinisation and minimisation.
- In a similar vein, we show that while clearly being comonadic, higher dimensional trees are also monadic in nature. This is an example of the kind of result that is both fundamental and would be missed without the abstract categorical formulation.
- We justify our belief that clean mathematical foundations leads to a clean computation structure by implementing higher dimensional trees in the Haskell programming language.

Our intention with this research is to synthesise our abstract approach with the intuitions and applications of Rogers. This paper is just the beginning, clarifying some algebraic and categorical aspects, before turning to applications of higher dimensional trees in language theory. We believe that this paper provides an interesting application of category theory, especially of algebras and coalgebras, taming the apparent complexity which Rogers' definitions possess at first sight and allowing us to transfer known results to higher dimensional automata.

The paper is structured as follows. *Section 2* follows parts of Rogers [11] and presents his notions of higher-dimensional trees and automata. *Section 3* presents our reformulation of Rogers' notions using fixed-point equations and coalgebras. *Section 4* shows that Rogers' higher dimensional trees are examples of *containers* which allows us to deduce several useful meta-theoretic results needed later. *Section 5* uses our abstract

reformulation to show that the classical theorems of determinisation and minimisation from automata theory hold.

2 Rogers's Higher Dimensional Trees

The most pervasive definition of (finitely branching) trees is via the notion of a tree domain. A tree domain is an enumeration of the paths in a tree - since a path is a list of natural numbers, a tree domain is a subset of lists of natural numbers. However, there should be two conditions on sets of paths reflecting the fact that i) if a node has an $n+1$'th child, then there should be an n'th child; and ii) all nodes apart from the root have a parent. Thus tree domains are defined as follows

Definition 2.1 (Tree Domains). *A tree domain $T \subseteq \mathbb{N}^*$ is a subset of lists of natural numbers such that*

- *(LS): If $w.(n+1) \in T$, then $w.n \in T$*
- *(A): If $w.n \in T$, then $w \in T$*

We use . for the concatenation of a list with an element. We call the first condition the left-sibling property (LS) and we call the second condition the ancestor property (A). Notice how tree domains, by focusing on paths, will inevitably lead to a process of computation dominated by the creation and consumption of sets of paths satisfying (LS) and (A). As we shall see later, tree domains and the paths in them can be treated more abstractly, and in a cleaner fashion, by the shapes and positions of the container reformulation of tree domains.

However, for now, we want to ask ourselves how the tree domains given above can be generalised from being 2-dimensional structures to n-dimensional structures. In the 2-dimensional case we had a notion of path as a list of natural numbers and then a tree domain consisted of a set of paths satisfying the properties (LS) and (A). Rogers defines n-dimensional tree domains by first defining what an n-dimensional path is and then defining an n-dimensional tree domain to be a set of n-dimensional paths satisfying higher dimensional variants of (LS) and (A). So what is an n-dimensional path? Notice that a natural number is a list of 1s and hence a list of natural numbers is a list of lists of 1s. Thus

Definition 2.2 (Higher Dimensional Paths [11, Def 2.1]). *The n-dimensional paths form a $\mathbb{N}$-indexed set P with $P_0 = 1$ (the one element set) and with P_{n+1} defined to be the least set satisfying*

- $[] \in P_{n+1}$
- *If $[x_1, .., x_m] \in P_{n+1}$ and $x \in P_n$, then $[x_1, .., x_m, x] \in P_{n+1}$*

A simpler definition would be that $P_n = \mathsf{List}^n 1$ but we wanted to give Roger's definition to highlight its concreteness. Having defined the n-dimensional paths we can define the n-dimensional tree domains as follows

Definition 2.3 (Higher Dimensional Tree domains [11, Def 2.2]). *Let $T_0 = \{\emptyset, 1\}$. The set T_{n+1} of $n + 1$-dimensional tree domains consists of those subsets $T \subseteq P_{n+1}$ such that*

– *(HDLS): If $s \in P_{n+1}$, then $\{w \in P_n | s.w \in T\} \in T_n$*
– *(HDA): If $s.w \in T$, then $s \in T$*

The first condition is the higher dimensional left sibling property (HDLS). It is slightly tricky as, in higher dimensions, there is no unique left sibling and so one cannot simply say that if a node has an $n+1$'th child then the node has an n'th child. (HDLS) solves this problem by saying the immediate children of a node in an $n+1$-dimensional tree domain form an n-dimensional tree domain. In the two dimensional case, (HDLS) is thus the requirement that the children of a node in a tree form a list. (HDA) is a straightforward generalisation of the 2-dimensional ancestor property (A). The reader may wish to check that a one dimensional tree domain is a set of lists over 1 closed under prefixes, that is, T_1 is bijective to $\mathsf{List}(1)$. There are two zero dimensional tree domains which correspond to the empty tree and to the tree which just contains one node and no children.

The notion of automata is central in formal language theory and generalises to higher dimensions in a straightforward way. Firstly, we must extend tree domains so that higher dimensional trees can actually store data - this is done by associating to each path in a tree domain, a piece of data to be stored there.

Definition 2.4 (Labelled tree domains [11, Def 2.3]). *A Σ-labelled tree domain is a mapping $T \to \Sigma$, where T is a tree domain and Σ a set (called the alphabet). We denote the set of n-dimensional Σ-labelled tree domains by $T_n(\Sigma)$.*

Definition 2.5 (n-Automaton [11, Def 2.9]). *An $(n+1)$-dimensional automaton over an alphabet Σ and a finite set of states Q is a finite set of triples (σ, q, T) where $\sigma \in \Sigma$, $q \in Q$ and T is a Q-labelled tree domain of dimension n.*

Rogers goes on to define when an (n+1)-automaton licenses (or accepts) an $n+1$-dimensional tree as follows. A (Σ-labelled) local tree is an element of $\Sigma \times T_n(\Sigma)$. An (n+1)-dimensional grammar over Σ is a finite subset of $\Sigma \times T_n(\Sigma)$, ie a finite set of local trees. An element $\lambda : T \to \Sigma$ in $T_{n+1}(\Sigma)$ is licensed by a grammar if for all $s \in T$, the pair $(\lambda(s), \lambda' : T' \to \Sigma)$ is in the grammar, where $T' = \{w | s.w \in T\}$ and $\lambda'(w) = \lambda(s.w)$. In other words, a tree is licensed by a grammar if it is constructed from the local trees of the grammar. Note that, forgetting the alphabet Σ, an automaton can be seen as a grammar over Q. An element in $T_{n+1}(\Sigma)$ is now licensed by an automaton if it is an image of a Q-labelled tree licensed by the grammar in which the the label of the root of each local tree has been replaced with a symbol in Σ associated with that local tree in the automaton [11].

We will see in Section 5 that acceptance is more easily defined via the unique morphism from the initial algebra of trees. For coalgebraists let us note here already that automata are coalgebras. First, the notion of labelling means that n-dimensional tree domains form a functor $T_n : \mathsf{Set} \to \mathsf{Set}$. In particular $T_0(X) = 1 + X$ and $T_1(X) = \mathsf{List}(X)$. Now, an $n+1$-dimensional automata over Σ is just a finite set Q and a function $Q \to \mathcal{P}(\Sigma \times T_n(Q))$. Automata and their accepted languages will be discussed in detail in Section 5, but let us look at two familiar examples already.

Example 2.6. A 1-automaton is essentially the standard notion of a non-deterministic string automata — that is a function $Q \to \mathcal{P}(\Sigma \times (1+Q))$ where each state can perform a Σ-transition and either terminate or arrive at another state.

Example 2.7. A 2-automaton is a coalgebra $\delta : Q \to \mathcal{P}(\Sigma \times T_1(Q))$, that is, a relation $\delta \subseteq Q \times (\Sigma \times \mathsf{List}(Q))$ which can be understood as a non-deterministic tree automata (see eg [6]): Given a state q and a tree $\sigma(t_1, \dots t_n)$ the automaton tries to recognise the tree by guessing a triple $(q, \sigma, [q_1, \dots q_n]) \in \delta$ and continuing this procedure in the states q_i with trees t_i. Whereas this coalgebraic definition has a top-down flavour, the accepted language is most easily defined in an algebraic (bottom-up) fashion as follows. The relation δ gives rise to a set of Q-labelled terms (or bottom-up $c$omputations) C via

$$\frac{(q, \sigma, []) \in \delta}{q\sigma \in C} \qquad \frac{(q, \sigma, [q_1, \dots q_n]) \in \delta, \quad q_i t_i \in C}{q\sigma(t_1, \dots t_n) \in C}.$$

where $q\sigma \in C$ means the automata recognises the σ-labelled tree starting from the state q. One then defines, wrt a set of accepting states Q_0, that the automaton accepts a tree t iff $qt \in C$ and $q \in Q_0$.

3 Higher Dimensional Trees, Algebraically

Despite being a natural generalisation of a 2-dimensional tree domain to an n-dimensional tree domain, Definition 2.3 is very concrete. For example, formalising the notion of licensing (following Definition 2.5 above) is tedious. We will show that a more abstract approach to the definition of tree domains is possible. In particular, the 1-dimensional tree domains are just the usual lists while the non-empty two-dimensional tree domains are known in the functional programming community as rose trees with a simple syntax and semantics. That is, categorically one may define $\mathsf{Rose}\, X = \mu Y.X \times \mathsf{List}\, Y$ and derive from this the equally simple Haskell implementation

```
data Rose a = Node a [Rose a]
```

What is really pleasant about this categorical/functional programming presentation of tree domains is that initial algebra semantics provides powerful methods for writing and reasoning about programs. In particular, it replaces fascination with the detailed representation of the structure of paths and the (LS) and (A) properties with the more abstract universal property of being an initial algebra. That is not to say paths are not important, just that they ought to be (in our opinion) a derived concept. Indeed, we show later in Theorem 4.6 how to derive the path algebra from the initial algebra semantics.

The natural question is whether we can give an initial algebra semantics for higher dimensional trees. The answer is not just yes, but yes in a surprisingly beautiful and elegant manner. As remarked earlier, the immediate children of a node in an $(n+1)$-dimensional tree should form an n-dimensional tree. This is formalised in

Definition 3.1. *Define a family of functors by*

$$\begin{aligned} R_{-1}X &= 0 \qquad & T_n X &= 1 + R_n X \quad (n \geq -1) \\ R_{n+1}X &= \mu Y.X \times T_n Y & & \end{aligned}$$

Note that we intend $R_{n+1}X$ to be the set of *non-empty* $n{+}1$-dimensional X-labelled tree domains while $T_{n+1}X$ is intended to be the set of *empty or non-empty* $n{+}1$-dimensional

X-labelled tree domains. Thus $R_{n+1}X$ should consist of an element of X to be stored at the root of the tree and a potentially empty n-dimensional tree domain labelled with further tree domains. While one could start indexing at 0 by defining $R_0X = X$, there is no harm in starting one step before with the definition of -1-dimensional trees. As expected, calculations show that

n	R_nX	T_nX
-1	0	1
0	X	$1 + X$
1	$\mathsf{List}^+(X)$	$\mathsf{List}(X)$
2	$\mathsf{Rose}(X)$	$1 + \mathsf{Rose}(X)$

where $\mathsf{List}^+(X)$ are the non-empty lists over X.

In fact, one can go further and not just define a sequence of functors R_n and T_n, but a higher order functor which maps a functor F to the functor sending X to $\mu Y.X \times FY$. We find this particularly interesting for both theoretical and practical reasons. At the theoretical level, we note that this construction of a functor from a functor is the final piece of the jigsaw remarked upon in [9] and summarised in

	Monads	Comonads
Initial Algebras	$\mu Y. X + FY$	$\boldsymbol{\mu Y. X \times FY}$
Final Coalgebras	$\nu Y. X + FY$	$\nu Y. X \times FY$

In [9], the three other higher order functors were remarked upon as follows:

- The map sending a functor to F to the functor $X \mapsto \mu Y.X + FY$ is the free monad construction
- The map sending a functor to F to the functor $X \mapsto \nu Y.X + FY$ is the free completely iterative monad construction
- The map sending a functor to F to the functor $X \mapsto \nu Y.X \times FY$ is the cofree comonad construction

Higher dimensional tree functors provide—to our knowledge—the first naturally arising instance of the remaining quadrant of the table above. From [9], we have

Theorem 3.2. *For any functor F, the map $X \mapsto \mu Y.X \times FY$ is a comonad.*

At a practical level, this higher order functor translates into the following simple definition of higher dimensional trees in Haskell, the canonical recursion combinator arising from the initiality of higher dimensional trees and their comonadic structure. In the following, `Maybe` is Haskell implementation of the monad sending X to $1 + X$.

```
data Rose f a = Rose a (Maybe (f (Rose f a)))
type Tree f a = Maybe (Rose f a)

data Rose0 a = Rose0 a      -- = \X -> X
type Rose1   = Rose Rose0   -- = \X -> List^+(X)
```

```
type Rose2  = Rose Rose1   -- = \X -> Rose(X)
type Rose3  = Rose Rose2

cata :: Functor f => (a -> Maybe (f b) -> b) -> Rose f a -> b
cata g (Rose x xs) = g x (fmap (fmap (cata g)) xs)

instance Functor Rose0 where
 fmap f (Rose0 a) = Rose0 (f a)

instance Functor f => Functor (Rose f)
 where fmap f = cata act where act a t = Rose (f a) t

class Comonad f where
 root   :: f a -> a
 comult  :: f a -> f (f a)

instance Comonad Rose0 where
 root  (Rose0 x)  = x
 comult (Rose0 x)  = Rose0 (Rose0 x)

instance Functor f => Comonad (Rose f) where
 root (Rose x xs)  = x
 comult (Rose x xs) = Rose (Rose x xs) (fmap (fmap comult) xs)
```

As we have seen, higher dimensional trees are instances of canonical constructions which always produce comonads. It is also well-known that List^+ and List are monads. Less well known is that Rose is a monad. Clearly R_0 is also a monad. Indeed we have

Theorem 3.3. *For all $n \geq 0$, R_n is a monad.*

Space prevents us from detailing the proof of this theorem. However, it is important because it allows computation with higher dimensional trees to be further simplified via the use of the monadic notation available in Haskell to structure common patters of computation. For example, parsing and filtering become particularly simple.

To summarise, we depart from Rogers in not defining higher dimensional trees in terms of paths, but via the more abstract categorical notion of initial algebras. As a result, we take the *functor* T_n as primary as opposed to the *set* of tree domains which one may then label. This cleaner mathematical foundation reveals higher dimensional trees to be related to the fundamental constructions of the free monad, free completely iterative monad and cofree comonad. It also leads to a simple implementation of higher dimensional trees in Haskell.

4 Containers

Containers [8] are designed to represent those functors which are concrete data types and those natural transformations which are polymorphic functions between such concrete data types. Such data types include lists, trees etc, but not solutions of mixed

variance recursive domain equations such as $\mu X.(X \to X) + \mathbb{N}$. Containers take as primitive the idea that concrete data types consist of its general form or *shapes* and, given such a shape, a set of *positions* where data can be stored. Since Rogers' n-dimensional trees certainly store data at the nodes of the n-dimensional tree, it is natural to ask whether these trees are indeed containers. In this section, we see that the functors T_n and R_n are indeed containers and point out the following theoretical and practical consequences:

- Many properties of n-dimensional trees can be deduced from the fact that they are containers. As just one example, our transformation of a non-deterministic automata into a deterministic one requires n-dimensional trees to preserve weak pullbacks. This follows from the fact that n-dimensional trees are containers.
- While we choose not to take paths and tree domains as primitive in our treatment of higher dimensional trees, paths are nevertheless important. We want a capability to compute with them but do not want the burden of verifying the (HDLS) and (HDA) properties. In particular, we want a purely inductive definition of tree domains and paths and, remarkably, find that the shapes and positions of the container T_n provide that.

Containers are semantically equivalent to *normal functors* and a special case of *analytic functors*. However, while containers talk about the different shapes a data structure can assume, analytic functors talk about the number of structures of a given size and hence there is no clear, simple and immediate connection between tree domains and paths on the one hand and analytic functors on the other hand. Thus we use containers rather than analytic functors to represent higher dimensional trees. In the rest of this section, we introduce containers and recall some of the closure properties of containers. This proves sufficient to then show that all Rogers' trees are indeed containers. While the theory of containers can be developed in any locally cartesian closed category with W-types and disjoint coproducts, we restrict to the category of Set to keep things simple.

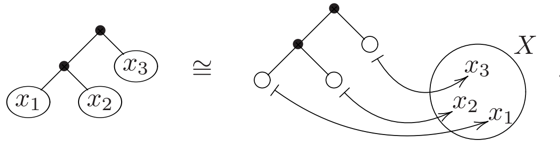
The simplest example of a data type which can be represented by a container is that of lists. Indeed, any element of the type $\mathsf{List}(X)$ of lists of X can be uniquely written as a natural number n given by the length of the list, together with a function $\{0, \ldots, n-1\} \to X$ which labels each position within the list with an element from X. Thus

$$\mathsf{List}(X) = \coprod_{n \in \mathbb{N}} \{0 \ldots n-1\} \to X$$

More generally, we consider data types given by i) *shapes* which describe the *form* of the data type; and ii) for each shape, $s \in S$, there is a set of positions $P(s)$. Thus we define

Definition 4.1 (Container). *A container (S, P) consists of a set S and an S-indexed family P of sets, ie a function $P : S \to \mathsf{Set}$.*

As suggested above, lists can be presented as a container with shapes $\mathbb{N}$ and positions defined by $P(n) = \{0, \ldots, n-1\}$. Similarly, any binary tree can be uniquely described by its underlying shape (which is obtained by deleting the data stored at the leaves) and a function mapping the positions in this shape to the data thus:



The extension of a container is an endofunctor defined as follows:

Definition 4.2 (Extension of a Container). *Let (S, P) be a container. Its extension, is the functor $T_{(S,P)}$ defined by*

$$T_{(S,P)}(X) = \coprod_{s \in S} P(s) \to X$$

Thus, an element of $T_{(S,P)}(X)$ is a pair (s, f) where $s \in S$ is a shape and $f : P(s) \to X$ is a labelling of the positions of s with elements from X. The action of $T_{(S,P)}$ on a morphism $g : X \to Y$ sends the element (s, f) to the element $(s, g \cdot f)$. If F is a functor that is the extension of a container, then the shapes of that container can simply be calculated as $F1$ — that is $S = T_{(S,P)}1$. This corresponds to erasing the data in a data structure to reveal the underlying shape. Containers have many good properties, in particular, many constructions on functors specialise to containers. These closure properties are summarised below.

Theorem 4.3 (Closure properties of Containers [8]). *The following are true*

- *The identity functor is the extension of the container with one shape and one position.*
- *The constantly A valued functor has shapes A and positions given by $Pa = 0$.*
- *Let (S_1, P_1) and (S_2, P_2) be containers. Then the functor $T_{(S_1,P_1)} + T_{(S_2,P_2)}$ is the extension $T_{(S,P)}$ of the container (S, P) defined by*

$$S = S_1 + S_2 \qquad P(\mathsf{inl}(s)) = P_1 s \qquad P(\mathsf{inr}(s)) = P_2 s$$

- *Let (S_1, P_1) and (S_2, P_2) be containers. Then the functor $T_{(S_1,P_1)} \times T_{(S_2,P_2)}$ is the extension $T_{S,P}$ of the container (S, P) defined by*

$$S = S_1 \times S_2 \qquad P(s_1, s_2) = P_1 s_1 + P_2 s_2$$

In order to show that containers are closed under fixed points, we need to introduce the notion of a n-ary container to represent n-ary functors. For the purposes of our work, we only need bifunctors and so we restrict ourselves to binary containers.

Definition 4.4 (Bi-Containers). *A bi-container consists of two containers with the same underlying shape. That is a set S and a pair of functions $P_1, P_2 : S \to \mathsf{Set}$. The extension of a binary container is a bifunctor given by*

$$T_{(S,P_1,P_2)}(X, Y) = \coprod_{s \in S} (P_1 s \to X) \times (P_2 s \to Y)$$

Given a bi-container (S, P_1, P_2), the functor $X \mapsto \mu Y.F(X, Y)$ is a container as demonstrated by the following theorem.

Theorem 4.5 (Fixed Points of Containers [8]). *Let (S, P_1, P_2) be a bi-container and let $F(X, Y) = T_{(S,P_1,P_2)}(X, Y)$ be its extension. Then the functor $\mu Y.F(X, Y)$ is a container with shapes given by*

$$S = \mu Y.T_{(S,P_2)}(Y)$$

and positions given by

$$P(s, f) = P_1 s + \coprod_{p \in P_2 s} P(fp)$$

To understand this theorem, think of an element of $\mu Y.F(X, Y)$ as a tree with a top F-layer which stores elements from X at the X positions in this F-layer and further elements of $\mu Y.F(X, Y)$ at the Y-positions of this F-layer. We know that the shapes of the functor $\mu Y.F(X, Y)$ must be this functor at 1, ie $\mu Y.F(1, Y)$. More concretely, a shape for $\mu Y.F(X, Y)$ must thus be an F-shape for the top layer of a tree and, for each Y-position of that shape, we must have a shape of $\mu Y.F(X, Y)$ to represent the tree recursively stored at that position. As for the positions for storing data of type X in a tree with shape (s, f) where $s \in S$ and $f : P_2 s \to \mu Y.F1Y$, these should be either the positions for storing X-data in the top layer given by $P_1 s$ or, for each position in $p \in P_2 s$, a position in the subtree stored at that position. Since that subtree has shape fp, we end up with the formula above.

Applying these closure properties, we derive the following

Theorem 4.6. *Rogers' n-dimensional non-empty tree functor R_n is the extension of a container. That is, $R_n = T_{(S_n^+, P_n^+)}$ where*

$$S_{-1}^+ = 0$$

$$\begin{aligned} S_{n+1}^+ &= \mu Y.1 + R_n Y \\ P_{n+1}^+(\mathsf{inl}*) &= 1 \\ P_{n+1}^+(\mathsf{inr}(s, f)) &= 1 + \coprod_{p \in P_n s} P_{n+1}^+(fp) \end{aligned}$$

As a corollary, T_n is also the extension of a container. That is $T_n = T_{(S_n,P_n)}$ where $S_n = 1 + S_n^+$, $P_n(\mathsf{inl}*) = 0$ and $P_n(\mathsf{inr}s) = P_n^+ s$. What is particularly nice about the container presentation of T_n is that the shapes S_n are in bijection with the tree domains while the paths in any tree domain are in bijection with the positions of the equivalent shape. Further, the paths are given by a purely inductive definition.

5 Automata, (Co)Algebraically

We show that the classical automata-theoretic results about determinisation and minimisation extend to the higher-dimensional automata of Rogers. Using our reformulation of Rogers' structures in Section 3 and the container-technology of Section 4, these results become special cases of the classical results about automata as algebras for a

functor, a theory initiated by Arbib and Manes [2,3,4]. We also extend Rogers' work by appropriate notions of signature and deterministic automata.

We should like to point out that none of the constructions or proofs in this section requires the explicit manipulation of trees or tree domains.

Before starting on the topic of the section, we review the situation for string and tree automata. Ignoring initial and accepting states, the situation is depicted in

(strings)	non-det	det
top-down	$Q \to \mathcal{P}(A \times Q)$	$Q \to Q^A$
bottom-up	$A \times Q \to \mathcal{P}Q$	$A \times Q \to Q$

(trees)	non-det	det
top-down	$Q \to \mathcal{P}(FQ)$	—
bottom-up	$FQ \to \mathcal{P}Q$	$FQ \to Q$

For both string and tree automata, the relationship between non-deterministic top-down automata (=coalgebras in the Kleisli-category of $\mathcal{P}$) and non-deterministic bottom-up automata (=algebras in the Kleisli-category of $\mathcal{P}$) is straightforward: both $Q \to \mathcal{P}(FQ)$ and $FQ \to \mathcal{P}Q$ are just two different ways of denoting a relation $\subseteq Q \times FQ$. The relationship between deterministic top-down automata (=coalgebras) and deterministic bottom-up automata (=algebras) is given in the string case by the adjunction $A\times - \dashv (-)^A$ (this situation is generalised and studied in [3]). In the tree case, F is an arbitrary functor on Set and so has in general no right-adjoint.[1] It is still possible to describe deterministic top-down tree automata but they are strictly less expressive [6, Chapter 1.6].

The familiar move from non-deterministic to deterministic string automata can be summarised as follows. Any non-deterministic transition structure $f : Q \to \mathcal{P}(A \times Q)$ can be lifted to a map $\bar{f} : \mathcal{P}Q \to \mathcal{P}(A \times Q)$ given by $\bar{f}(S) = \bigcup_{q\in S} f(q)$. Using $\mathcal{P}(A \times Q) \cong (\mathcal{P}Q)^A$, $\bar{f}$ is a deterministic transition structure $\mathcal{P}Q \to (\mathcal{P}Q)^A$ on $\mathcal{P}Q$. Determinisation for tree automata will be discussed below.

5.1 Signatures

Rogers' automata of Definition 2.5 do not associate arities to the symbols in the alphabet Σ. For example, in the tree automata of Example 2.6, one σ may appear in two triples $(q, \sigma, l_1), (q, \sigma, l_2) \in \delta$ where l_1 and l_2 are lists of different lengths. Thus the same 'function symbol' σ may have different arities and the Σ-labelled trees are not exactly elements of a term algebra.

To rectify this situation, we must ask ourselves what is the appropriate notion of arity if operations take as input higher dimensional trees. In the two-dimensional case arities are natural numbers: the arity of a function symbol σ is the number of its arguments. But, in container terminology, $\mathbb{N}$ is just the the set of shapes of $T_1 = \mathsf{List}$. Thus, when operations of a signature are consuming higher dimensional trees, their arities should be the shapes of trees one dimension lower. This leads to

Definition 5.1 (**$(n+1)$-dimensional signature).** *An $(n+1)$-dimensional signature is a set Σ with a map $\Sigma \to T_n(1)$.*

Example 5.2. 1. A 1-dimensional signature is a map $r : \Sigma \to \{0, 1\}$, due to the isomorphism $T_0(1) \cong \{0, 1\}$. We will see below (Example 5.4) that 0 specifies nullary operations and 1 specifies unary operations.

[1] In fact, if $F : \mathsf{Set} \to \mathsf{Set}$ has a right-adjoint, then $F = A \times -$ for $A = F1$.

2. A 2-dimensional signature is a signature in the usual sense, due to the isomorphism $T_1(1) \cong \mathbb{N}$ that maps a list to its length.

The next step is to associate to each signature a functor in such a way that the initial algebra for the functor contains the elements of the language accepted by an automaton. The simplest and most elegant way to do this is to construct a container and use its extension. Recalling that $P_n : S_n \to \mathsf{Set}$ is the container whose extension is T_n and that $S_n = T_n(1)$, we can turn any signature $r : \Sigma \to T_n(1)$ into the container (Σ, P_r) as follows

$$P_r : \Sigma \xrightarrow{r} S_n \xrightarrow{P_n} \mathsf{Set}. \quad (1)$$

Definition 5.3 **(F_Σ).** *The functor $F_{(\Sigma,r)}$, or briefly F_Σ, associated to a signature $r : \Sigma \to T_n(1)$ is the extension $T_{(\Sigma,P_r)}$ of the container (1), that is, $F_\Sigma(X) = \coprod_{\sigma\in\Sigma} P_n(r(\sigma)) \to X$.*

Example 5.4. 1. A 1-dimensional signature $r : \Sigma \to \{0, 1\}$ gives rise to the functor $F_\Sigma(X) = \Sigma_0 + \Sigma_1 \times X$ where $\Sigma_i = r^{-1}(i)$.
2. A 2-dimensional signature $r : \Sigma \to \mathbb{N}$ gives rise to the functor $F_\Sigma(X) = \coprod_{\sigma\in\Sigma} X^{r(\sigma)}$ usually associated with a signature.

The next two propositions, which one might skip as a pedantic technical interlude, make the relation between an alphabet Σ' and a signature $\Sigma \to T_n(1)$ precise. The first proposition says that trees for the signature $\Sigma \to T_n(1)$ (ie elements of the initial F_Σ-algebra) are also trees over the alphabet Σ (ie elements of $T_{n+1}(\Sigma)$). The second proposition says that trees over the alphabet Σ' are the same as trees over the signature $\Sigma' \times T_n(1) \to T_n(1)$.

Proposition 5.5. *For each $(n+1)$-dimensional signature $\Sigma \to T_n(1)$, there is a canonical F_Σ-algebra structure on $T_{n+1}(\Sigma)$. Moreover, the unique algebra morphism from the initial F_Σ-algebra to $T_{n+1}(\Sigma)$ is injective.*

Proof. The carrier of the initial F_Σ-algebra is $\mu Y.F_\Sigma(Y)$ and $R_{n+1}(\Sigma)$ is $\mu Y.\Sigma \times T_n(Y)$. The injective morphism in question arises from the injective map of type $F_\Sigma(Y) \to \Sigma \times T_n(Y)$, that is of type $(\coprod_{\sigma\in\Sigma} P(r(\sigma)) \to Y) \to \Sigma \times (\coprod_{s\in S_n} P(s) \to Y)$, which maps pairs $(\sigma, f) \in F_\Sigma(Y)$ to $(\sigma, (r(\sigma), f))$.

Proposition 5.6. *Let Σ' be a set (called an alphabet) and Σ be the signature given by the projection $r : \Sigma' \times T_n(1) \to T_n(1)$. Then $R_{n+1}(\Sigma')$ is isomorphic to the (carrier of the) initial F_Σ-algebra.*

Proof. The carrier of the initial F_Σ-algebra is $\mu Y.F_\Sigma(Y)$ and $R_{n+1}(\Sigma')$ is $\mu Y.\Sigma' \times T_n(Y)$. But $\Sigma' \times T_n(Y) = \Sigma' \times (\coprod_{s\in S_n} P(s) \to Y) \cong \coprod_{(\sigma,s)\in\Sigma'\times S_n} P(s) \to Y = \coprod_{\sigma\in\Sigma} P(r(\sigma)) \to Y = F_\Sigma(Y)$.

5.2 Higher Dimensional Automata

Before giving a coalgebraic formulation of Rogers' automata (Definition 5.10), we introduce the corresponding notion of deterministic automaton (Definition 5.7), which has a particularly simple definition of accepted language and is used in the next subsection on determinisation and minimisation. (Recall Definition 5.3 of F_Σ.)

Definition 5.7. *A deterministic $(n+1)$-dimensional automaton for the signature $\Sigma \to T_n(1)$ is a function*

$$F_\Sigma Q \to Q.$$

Example 5.8. 1. To obtain the usual string automata over an alphabet A we consider a 1-dimensional signature Σ consisting of the elements of A as unary operation symbols plus one additional nullary operation symbol (see Example 5.4.1). $F_\Sigma(Q)$ is then $1 + A \times Q$.
2. A 2-dimensional automaton is the usual deterministic bottom-up tree automaton [6].

Definition 5.9. *A state q in a deterministic $(n+1)$-dimensional automaton for the signature $\Sigma \to T_n(1)$ accepts an $(n+1)$-dimensional tree t if the unique morphism from the initial F_Σ-algebra maps t to q.*

We adapt Rogers' definition of automata given in Definition 2.5:

Definition 5.10. *An $(n+1)$-dimensional automaton for the signature $\Sigma \to T_n(1)$ is a function*

$$Q \to \mathcal{P}(F_\Sigma(Q)).$$

Example 5.11. 1. In the case of string automata, $F_\Sigma(Q)$ is $1{+}A{\times}Q$ and an automaton becomes $Q \to \mathcal{P}(1{+}A{\times}Q) \cong 2{\times}(\mathcal{P}Q)^A$. The map $Q \to 2$ encodes the accepting states and the map $Q \to (\mathcal{P}Q)^A$ gives the transition structure.
2. Comparing with the previous definition, a 2-dimensional automaton $\delta : Q \to \mathcal{P}(F_\Sigma(Q))$ can still be considered as a set of triples $\delta \subseteq Q \times (\Sigma \times \mathsf{List}(Q))$, but not all such triples are allowed: for $(q, \sigma, \langle q_1, \ldots q_n \rangle) \in \delta$ it has to be the the case that the arity of σ is n. This coincides with the notion of a non-deterministic top-down tree automaton as in [6].

We have indicated how to define the accepted language of a (non-deterministic) automaton in Example 2.7. In particular, we found it natural to give a bottom-up formulation. We will now generalise this definition. The basic idea is as follows. We first observe that we cannot use the final coalgebra for the functor $\mathcal{P}F_\Sigma$ since this coalgebra would take the branching given by $\mathcal{P}$ into account. Instead, the correct idea is to consider a non-deterministic automaton as a F_Σ-coalgebra in the category of relations. We first note the following proposition which follows from F_Σ being the extension of a container.

Proposition 5.12. *F_Σ preserves weak pullbacks.*

Now let Rel denote the category of sets and relations.

Definition 5.13. *Given a functor F on Set we define $\bar{F}$ to map sets X to $\bar{F}X = FX$ and to map relations $X \overset{\pi_0}{\leftarrow} R \overset{\pi_1}{\to} Y$ to $\bar{F}R = F(\pi_0)^\circ; F(\pi_1)$ where $(-)^\circ$ denotes relational converse and ';' relational composition .*

Barr [5] showed that $\bar{F}$ is a *functor* on Rel if and only if F preserves weak pullbacks. A theorem of de Moor [7, Theorem 5] and Hasuo et al [10, Theorem 3.1] then guarantees that the initial F-algebra $i : FI \to I$ in Set gives rise to the final $\bar{F}$-coalgebra $i^\circ : I \to \bar{F}I$ in Rel. This gives a 'coinductive' definition of the accepted language of a non-deterministic automaton:

Definition 5.14. *The language accepted by a state q of an $(n+1)$-dimensional automaton $Q \to \mathcal{P}(F_\Sigma(Q))$ is given by the unique arrow (in the category* Rel*) into the final $\bar{F}_\Sigma$-coalgebra .*

Note that this definition associates to q a subset of the carrier I of the initial F_Σ-algebra.

It is clear from the constructions that every deterministic automaton can be considered as a non-deterministic automaton, and that the two notions of accepted language agree. We make this precise with the following definition and proposition.

Definition 5.15. *The non-deterministic automaton corresponding to the deterministic automaton $f : F_\Sigma Q \to Q$ is given by $f^\circ : Q \to \mathcal{P}F_\Sigma Q$ (where f° is again the converse relation of (the graph of) f).*

Proposition 5.16. *The deterministic automaton $F_\Sigma Q \to Q$ accepts t in q if and only if the corresponding non-deterministic automaton $Q \to \mathcal{P}F_\Sigma Q$ has t in the language of q.*

5.3 Determinisation and Minimisation

This section follows the work by Arbib and Manes [2,3,4] on automata as algebras for a functor on a category.

Determinisation. First observe that the elementship relation $\ni \subseteq \mathcal{P}X \times X$ can be lifted to $\bar{F}(\ni) \subseteq F\mathcal{P}X \times FX$, which can be written as

$$F\mathcal{P}X \xrightarrow{\tau_X} \mathcal{P}FX \tag{2}$$

τ_X is well-known to be natural in X whenever F preserves weak pullbacks. Now, given a non-deterministic automaton

$$Q \to \mathcal{P}F_\Sigma Q \tag{3}$$

we first turn it from top-down to bottom-up by going to the converse relation

$$F_\Sigma Q \to \mathcal{P}Q \tag{4}$$

and then lift it from $F_\Sigma Q$ to $\mathcal{P}F_\Sigma Q$ and precompose with τ to obtain

$$F_\Sigma \mathcal{P}Q \to \mathcal{P}F_\Sigma Q \to \mathcal{P}Q \tag{5}$$

Remark 5.17. The step from (4) to (5) is a special case of [4, Lemma 7] (where $\mathcal{P}$ can be an arbitrary monad on a base category).

Theorem 5.18. *Given an $(n+1)$-dimensional automaton $Q \to \mathcal{P}F_\Sigma Q$ (Definition 5.10) with accepting states $Q_0 \subseteq Q$, the state Q_0 in the corresponding deterministic automaton (5) accepts the same language.*

Minimisation. A deterministic automaton with a set of accepting states is a structure

$$F_\Sigma Q \xrightarrow{\delta} Q \xrightarrow{\alpha} 2 \tag{6}$$

We denote by $F_\Sigma I \to I$ the initial F_Σ-algebra and by $\rho : I \to Q$ the unique morphism given by initiality. The map $\beta = \alpha \circ \rho$ is called the *behaviour* of (6) because $\beta(t)$ tells us for any $t \in I$ whether it belongs to the accepted language or not. Note that the automata (6) form a category, denoted DAut, which has as morphism $f : (\delta, \alpha) \to (\delta', \alpha')$ those algebra morphism $f : \delta \to \delta'$ satisfying $\alpha' \circ f = \alpha$.

Definition 5.19 ([2, Section 4]). *Let $\iota : F_\Sigma I \to I$ be the initial F_Σ-algebra. The automaton (6) is* reachable *if the algebra morphism $\iota \to \delta$ is surjective and it is a* realisation *of $\beta : I \to 2$ iff there is a morphism $(\iota, \beta) \to (\delta, \alpha)$ in* DAut. *Moreover, (6) is a minimal realisation of β iff for all reachable realisations (δ', α') of β there is a unique surjective* DAut*-morphism $f : (\delta', \alpha') \to (\delta, \alpha)$.*

Different minimal realisation theorems can be found in Arbib and Manes [2,3,4] and Adámek and Trnková [1]. The theorem below follows [1, V.1.3].

Theorem 5.20. *Let Σ be an $(n+1)$-dimensional signature, F_Σ the corresponding functor and $F_\Sigma I \to I$ the initial F_Σ-algebra. Then every map $\beta : I \to 2$ has a minimal realisation.*

Proof. Let $e_i : (\iota, \beta) \to (\delta_i, \alpha_i)$ be the collection of all surjective DAut-morphisms with domain (ι, β). Let f_i be the multiple pushout of e_i in Set and $g = f_i \circ e_i$. The universal property gives us α with $\alpha \circ g = \beta$. Being a container F_Σ is finitary and, therefore [1, V.1.5], preserves the multiple pushout. Hence there is δ with $\delta \circ F_\Sigma g = g \circ \iota$. Since F_Σ preserves, like any set-functor, surjective maps, δ is uniquely determined. We have constructed an automaton (δ, α) that realises β. It is minimal because any other reachable realisation appears as one of the e_i.

6 Conclusion

This paper applies (co)algebraic and categorical techniques to Rogers' recent work in linguistics on higher dimensional trees. In particular, we have given an algebraic formulation of Rogers' higher dimensional trees and automata. Our analysis shows that, just as ordinary trees, the higher dimensional trees organise themselves in an initial algebra for a set-functor. This allowed us to use Arbib and Manes' theory of automata as algebras for a functor, yielding simple definitions of accepted language and straightforward constructions of determinisation and minimisation.

More importantly, as we have only been able to hint at, our algebraic formulation gives us the possibility to write programs manipulating the trees in functional programming languages like Haskell that support polymorphic algebraic data types. Future work will be needed to substantiate our claim that, in fact, our abstract categorical treatment is very concrete in the sense that it will give rise to simple implementations of algorithms manipulation higher dimensional trees. A good starting point could be Rogers' characterisation of non-strict tree adjoining grammars as 3-dimensional automata [11, Thm 5.2].

Acknowledgements. The 2nd author wishes to thank Ichiro Hasuo for helpful discussions.We are also grateful to the referees for their numerous comments that helped us to improve the presentation.

References

1. Adámek, J., Trnková, V.: Automata and Algebras in Categories. Kluwer Academic Publishers, Dordrecht (1990)
2. Arbib, M.A., Manes, E.G.: Machines in a category: An expository introduction. SIAM Review 16 (1974)
3. Arbib, M.A., Manes, E.G.: Adjoint machines, state-behaviour machines, and duality. Journ. of Pure and Applied Algebra 6 (1975)
4. Arbib, M.A., Manes, E.G.: Fuzzy machines in a category. Bull. Austral. Math. Soc. 13 (1975)
5. Barr, M.: Relational algebras. LNM 137 (1970)
6. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (1997) Available online.
7. de Moor, O.: Inductive data types for predicate transformers. Information Processing Letters 43(3), 113–118 (1992)
8. Ghani, N., Abbott, M., Altenkirch, T.: Containers - constructing strictly positive types. Theoretical Computer Science 341(1), 3–27 (2005)
9. Ghani, N., Lüth, C., de Marchi, F, Power, J.: Dualizing initial algebras. Mathematical Structures in Computer Science 13(1), 349–370 (2003)
10. Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace theory. In: International Workshop on Coalgebraic Methods in Computer Science (CMCS 2006). Elect. Notes in Theor. Comp. Sci. vol. 164, pp. 47–65. Elsevier, Amsterdam (2006)
11. Rogers, J.: Syntactic structures as multi-dimensional trees. Research on Language and Computation 1(3-4), 265–305 (2003)

A Semantic Characterization of Unbounded-Nondeterministic Abstract State Machines

Andreas Glausch and Wolfgang Reisig

Humboldt-Universität zu Berlin
Institut für Informatik
{glausch,reisig}@informatik.hu-berlin.de

Abstract. Universal algebra usually considers and examines algebras as static entities. In the mid 80ies Gurevich proposed *Abstract State Machines* (ASMs) as a computation model that regards algebras as dynamic: a state of an ASM is represented by a freely chosen algebra which may change during a computation. In [8] Gurevich characterizes the class of *sequential ASMs* in a purely semantic way by five amazingly general and elegant axioms. In [9] this result is extended to *bounded-nondeterministic ASMs*.

This paper considers the general case of *unbounded-nondeterministic ASMs*: in each step, an unbounded-nondeterministic ASM may choose among unboundedly many (sometimes infinitely many) alternatives. We characterize the class of unbounded-nondeterministic ASMs by an extension of Gurevich's original axioms for sequential ASMs. We apply this result to prove the reversibility of unbounded-nondeterministic ASMs.

1 Introduction

Abstract State Machines (ASMs) have been introduced as a "computational model that is more powerful and more universal than the standard computational models" by Yuri Gurevich in 1985 [6]. This is achieved by a combination of two classic notions of computer science and mathematics: *transition systems* and *algebras*.

Transition systems play a fundamental role in theoretical computer science. Usually, the operational semantics of a discrete computational model is specified in terms of a transition system, consisting of a set of *states* and a *next-state relation*. Examples are the transition systems as generated by Turing machines, λ-expressions, Petri nets, etc. The computational model of ASMs also fits into this setting, but differs from classical computational models in its general notion of states: each state of an ASM is an algebra.

As usual, an algebra A comprises a nonempty set U_A (its *universe*) together with finitely many functions defined over U_A, each with a fixed arity. No additional properties are required. In fact, *every* algebra may serve as a state of an ASM. As a consequence, a state of an ASMs may naturally include any mathematical data structure that can be described in terms of logic, e.g. sets,

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 242–256, 2007.
© Springer-Verlag Berlin Heidelberg 2007

real numbers, abstract geometrical objects, vector spaces, or even uncomputable functions. In classical computational models, such concepts usually require a particular encoding, if possible at all.

The seamless integration of arbitrary data structures make ASMs particularly well suited for abstract and natural modeling of algorithms and systems in general. Accordingly, ASMs have been extended to a full-fledged design and analysis methodology [5,4]. By stepwise refinement and composition of ASMs, large-scale real-world systems have been formally modeled and analyzed [10,12].

2 Scope and Contribution of This Paper

Classically, ASMs employ a pseudo-code like program syntax to describe the updates to be applied to a state A. This syntax is based on Σ-terms defined over the signature Σ of A and some additional elementary control structures. During the last decade, special attention towards a *syntax-independent* characterization of ASMs has been paid. Such characterizations are valuable in order to compare the expressive power of different variants of ASMs with classical computation models, and help to identify subtle properties of ASMs.

As stated above, ASMs fit into the general setting of transition systems. Correspondingly, a class $\mathcal{C}$ of ASMs may be characterized by answering the following question:

Which transition systems can be represented by the ASMs in $\mathcal{C}$?

In [8] Gurevich answers this question for the class of *sequential small-step ASMs* in a surprisingly elegant way: he defines a class of transition systems which he calls *sequential algorithms* by three general and semantic axioms, and proves this class to be equivalent to sequential ASMs (c.f. also [11]). Later, Blass and Gurevich identified similar characterizations for other variants of ASMs, including bounded-nondeterministic, parallel, and interactive versions [9,1,2,3].

In this paper we contribute to this work by a corresponding result for the general case of *unbounded-nondeterministic small-step ASMs* (nondeterministic ASMs for short). Nondeterministic ASMs have been introduced in [7] as "ASMs with qualified choose". We define the class of *nondeterministic algorithms* by purely semantic axioms, and prove that this class is equivalent to nondeterministic ASMs. This result shows that nondeterministic ASMs capture a surprisingly large class of transition systems.

We exemplify the profit of this result by proving the reversibility of nondeterministic ASMs: for each nondeterministic ASM M there exists a nondeterministic ASM M^{-1} executing M in reverse order.

The rest of this paper is organized as follows. In the next section we axiomatize the class of nondeterministic algorithms. In Sect. 4 we exemplify and define nondeterministic ASMs. Finally, we prove a theorem stating the equivalence of both notions, and apply this theorem in order to show the reversibility of nondeterministic ASMs.

3 Nondeterministic Algorithms

In [8] Gurevich defines the class of *sequential algorithms* by the following (slightly rearranged) five axioms which are very general, nevertheless simple and intuitive:

1. A sequential algorithm consists of a set of states $\mathcal{S}$, a set of initial states $\mathcal{I} \subseteq \mathcal{S}$, and a next-state function $\tau : \mathcal{S} \to \mathcal{S}$.
2. Each state $A \in \mathcal{S}$ is an algebra.
3. τ preserves the universe of states.
4. $\mathcal{S}$ and $\mathcal{I}$ are closed under isomorphism, and τ preserves isomorphism.

The decisive fifth axiom of sequential algorithms is *bounded exploration*. Intuitively, the bounded exploration axiom reads:

5. A finite set of ground terms is sufficient to characterize τ.

Gurevich shows in [8] that each sequential algorithm can be represented syntactically by a corresponding sequential ASM.

In this section we define the class of *nondeterministic algorithms* by five likewise simple axioms. The first four axioms are a canonical nondeterministic extension of Gurevich's original axioms. This extension has been presented in [9] already. The fifth axiom is entirely new: intuitively, it only requires a nondeterministic algorithm to perform a bounded amount of work in each step. In the following subsections, we present the axioms and justify their reasonability.

3.1 A Nondeterministic Algorithm Constitutes a Transition System

As indicated in the introduction already, the operational behaviour of a discrete algorithm specified in a particular computation model is usually given by a transition system. Therefore, we assume that the behaviour of every nondeterministic algorithm can naturally be represented by a nondeterministic transition system.

Axiom N1 (states and transitions). *A nondeterministic algorithm $\mathcal{N}$ consists of*

- *a set of states $S_{\mathcal{N}}$,*
- *a set of initial states $I_{\mathcal{N}} \subseteq S_{\mathcal{N}}$*
- *a next-state relation $\to_{\mathcal{N}} \subseteq S_{\mathcal{N}} \times S_{\mathcal{N}}$.*

Each pair $(A, A') \in \to_{\mathcal{N}}$ is a *step of* $\mathcal{N}$. As usual, a *run of* $\mathcal{N}$ is a sequence $A_0A_1A_2\ldots$ of states with $A_0 \in I_{\mathcal{N}}$ and $A_i \to_{\mathcal{N}} A_{i+1}$ for all indices i.

3.2 A State of a Nondeterministic Algorithm Is an Algebra

The huge experience on algebraic specification confirms that algebras are general enough to faithfully describe any static mathematical entity on any level of abstraction. Consequently, it is legitimate to assume that every state of every conceivable algorithm can be naturally described by an algebra. The second axiom formalizes this idea.

As usual, a *signature* Σ is used to address the functions of an algebra: Σ consists of finitely many function symbols $\mathtt{f}_1, \ldots, \mathtt{f}_k$, each $\mathtt{f}_i$ with its arity n_i. An algebra A is a Σ*-algebra* if A determines for each n-ary function symbol $\mathtt{f}$ a unique n-ary function $\mathtt{f}_A$.

As an algorithm always has a finite syntactical representation, an algorithm addresses only a finite set of functions. Hence, a single signature (with a finite set of symbols) suffices for all states. This leads to the second axiom:

Axiom N2 (states are algebras). *For a nondeterministic algorithm $\mathcal{N}$, all states in $S_\mathcal{N}$ are algebras over the same signature $\Sigma_\mathcal{N}$.*

Due to this axiom, we use the notions *state* and *algebra* interchangeably in this paper. As a running example we consider the following algebra Q, with universe $U_Q = \{1, 2, 3\}$, consisting of two nullary functions $\mathtt{a}_Q$ and $\mathtt{b}_Q$, and two unary functions $\mathtt{v}_Q$ and $\mathtt{next}_Q$:

$$\begin{array}{llll} & & \mathtt{v}_Q(1) = 1 & \mathtt{next}_Q(1) = 2 \\ \mathtt{a}_Q = 1 & \mathtt{b}_Q = 2 & \mathtt{v}_Q(2) = 2 & \mathtt{next}_Q(2) = 3 \\ & & \mathtt{v}_Q(3) = 3 & \mathtt{next}_Q(3) = 1 \end{array}$$

Hence, the signature Σ_Q of Q consists of the nullary function symbols $\mathtt{a}$ and $\mathtt{b}$, and the unary function symbols $\mathtt{v}$ and $\mathtt{next}$.

3.3 Steps of a Nondeterministic Algorithm Preserve the Universe

The universe U_A of an algebra A comprises all elementary semantic objects of A. In addition, A defines relationships between these objects in terms of functions over U_A. In this sense, the elements of U_A are atomic and foundational objects that cannot be decomposed, destroyed, or created. Only the functions of A are modified. As an example consider the Euclidian algorithm which computes the greatest common divisor of two given integers. The states of the Euclidian algorithm are built over the universe of all integers. A computation of the Euclidian algorithm does neither add nor remove integers from this universe. It merely computes new relationships such as “3 is the greatest common divisor of 12 and 27”. Consequently, the third axiom reads:

Axiom N3 (universe preservation). *For a nondeterministic algorithm $\mathcal{N}$ the following holds: for each step (A, A') of $\mathcal{N}$, A and A' have the same universe.*

3.4 Steps of a Nondeterministic Algorithm Preserve Isomorphisms

As usual, an isomorphism i between two Σ-algebras A and B is a bijective mapping $i : U_A \to U_B$ that preserves the functions of A. Isomorphic algebras only differ in their concrete representation of the universe, whereas the functions of both algebras are essentially the same.

For an algorithm the concrete representation of the universe is inessential. For example, the Euclidean algorithm computes the greatest common divisor regardless whether the integers are represented by some transistors on a chip

or by ink on paper. In general, an algorithm does not distinguish isomorphic states, but computes isomorphic next-states at isomorphic states. This insight is formalized by the fourth axiom:

Axiom N4 (isomorphism preservation). *For a nondeterministic algorithm* $\mathcal{N}$ *the following holds:*

(i) $\mathcal{S}_\mathcal{N}$ *and* $\mathcal{I}_\mathcal{N}$ *are closed under isomorphism.*
(ii) Let (A, A') *be a step of* $\mathcal{N}$ *and let* $B \in \mathcal{S}_\mathcal{N}$ *with an isomorphism* $i : A \to B$*. Then there is a step* (B, B') *of* $\mathcal{N}$ *such that* $i : A' \to B'$ *is an isomorphism.*

Alternatively, (ii) may be formulated based on the well-known notion of *simulation relation*: each bijective mapping i induces a simulation relation $\simeq$ on the states of $\mathcal{N}$ that is defined as $A \simeq B$ iff i is an isomorphism from A to B. Hence, isomorphic states simulate each other.

3.5 Steps of a Nondeterministic Algorithm Perform Bounded Work

The axioms N1–N4 are merely Gurevich's classical first four axioms to sequential algorithms, adjusted to the nondeterministic case. The fifth axiom presented next is new and requires some additional notions.

A real-world processor (e.g. a computer, an organization, or a human being) executing an algorithm performs only a *bounded* amount of work in each step (algorithms that allow for *unbounded* amount of work in each step are considered in [1]). Therefore, it is quite natural to require an algorithm to limit the amount of work to be done in each step. In the following we formalize this vague idea.

According to the previous axioms, a step of a nondeterministic algorithm preserves the signature and the universe of a state. That is, for a step (A, A'), A and A' share the same function symbols and the same function arguments, and differ only in their function values. In order to represent such differences formally, it is useful not to consider A as a collection of functions, but as a set of location-value-triples. A *location* of A consists of a n-ary function symbol $\texttt{f}$ and a n-ary argument tuple $\bar{a}$. For example $(\texttt{v}, [1])$ is a location of the state Q (we enclose the argument tuple in square brackets for the sake of readability). Each location $(\texttt{f}, \bar{a})$ of A defines a unique value $v = \texttt{f}_A(\bar{a})$. The triple $(\texttt{f}, \bar{a}, v)$ represents a small component of A which we call a *molecule of* A. For example, $(\texttt{v}, [1], 1)$ is a molecule of the state Q. Intuitively, the molecule $(\texttt{v}, [1], 1)$ states that "the function denoted by $\texttt{v}$ maps the argument tuple $[1]$ to the value 1".

A state A is completely described by its set of molecules. For example, the above state Q is represented by the following set of molecules:

$$\begin{aligned} Q = \{\ & (\texttt{a}, [], 1), (\texttt{b}, [], 2), \\ & (\texttt{v}, [1], 1), (\texttt{v}, [2], 2), (\texttt{v}, [3], 3), \\ & (\texttt{next}, [1], 2), (\texttt{next}, [2], 3), (\texttt{next}, [3], 1)\ \}. \end{aligned}$$

Calling them "updates", Gurevich employed molecules already in [7] to describe differences between algebras.

As announced above we intend to formalize the idea of "performing bounded work in each step": only a bounded part of a state A should contribute to a step, whereas the rest of A remains unaffected. The representation of A by a set of molecules permits a simple formalization of "a part of A": each subset $M \subseteq A$ is a *substate of* A. As an example, the set

$$M_Q =_{\text{def}} \{ (\mathtt{a}, [], 1), (\mathtt{v}, [2], 2) \}.$$

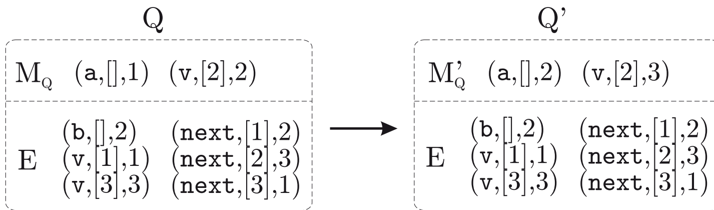
is a substate of Q.

Substates are used to describe steps that involve only a bounded part of a state: a *substep* changes a substate M by updating the values of the molecules in M. For instance, with

$$M'_Q =_{\text{def}} \{ (\mathtt{a}, [], 2), (\mathtt{v}, [2], 3) \}$$

the pair (M_Q, M'_Q) is a substep which changes the substate M_Q to the substate M'_Q. In general, a substep is a pair of substates (M, M') such that the locations of the molecules of M and M' coincide (i.e. M and M' differ only in the values of their molecules).

We employ substeps to capture the "amount of work" performed by a step of $\mathcal{N}$: each step (A, A') is decomposed into a substep (M, M') and a substate E disjoint from M and M' such that $(A, A') = (M \cup E, M' \cup E)$. Intuitively, the substep (M, M') describes the actual state change performed by the step whereas E describes the part of the state that is unaffected by the step (we use the letter "E" for "external"). In this case, we call the step (A, A') a *completion of* (M, M'). The following figure shows a step (Q, Q') which is a completion of the substep (M_Q, M'_Q):



A "bounded amount of work" then is captured by a substep bounded in size: For a natural number k, a substep (M, M') is *k-bounded* iff $|M| \leq k$ (which is equivalent to $|M'| \leq k$). We are now able to formulate the final axiom stating that the steps of a nondeterministic algorithm perform only bounded substeps:

Axiom N5 (bounded work). *For a nondeterministic algorithm* $\mathcal{N}$ *there exists a constant* $k \in \mathbb{N}$ *and a set* $\mathcal{W}$ *of* k*-bounded substeps such that for all states* A, A' *of* $\mathcal{N}$ *the following holds:* (A, A') *is a step of* $\mathcal{N}$ *iff* (A, A') *is an completion of a substep in* $\mathcal{W}$.

As $\mathcal{W}$ witnesses the fact that $\mathcal{N}$ performs only a bounded amount of work in each step, we call $\mathcal{W}$ a *bounded-work witness* of $\mathcal{N}$.

Though every substep in $\mathcal{W}$ is bounded, the set $\mathcal{W}$ itself may be infinite, even uncountably infinite. For example, consider a function symbol `r` holding a real number: $\mathcal{W}$ may contain uncountably many substeps that change the value of `r` to a different real number. This points out a decisive difference to Gurevich's original bounded-exploration axiom: a finite set of ground terms is not sufficient to characterize such rich behaviour.

We believe that the Axioms N1–N5 are intuitively convincing requirements to discrete nondeterministic algorithms. However, we do not demand any further requirements: we call *any* entity satisfying the Axioms N1–N5 a nondeterministic algorithm.

4 Nondeterministic Abstract State Machines

In the previous section we introduced the class of nondeterministic algorithms in a purely semantic and declarative way. This may appear strange, as algorithms usually are represented in an explicit and syntactical form, e.g. by program code. This raises the question whether a given nondeterministic algorithm can be represented in a syntactical way at all: is there a language expressive enough to describe *any* nondeterministic algorithm? In the following we answer this questions positively by presenting the operational computation model of *nondeterministic ASMs*, which has been introduced in [7] already.

We start by introducing the syntax and semantics of *nondeterministic ASM rules* which divide into four rule types: *assignment rules*, *conditional rules*, *parallel rules*, and *choice rules*. The syntax of these rules is simple and intuitive, and reminds of pseudo-code. However, in contrast to pseudo-code, nondeterministic ASM rules have a formal semantics. Nondeterministic ASM rules form the syntactical basis of nondeterministic ASMs introduced afterwards.

4.1 Assignment Rules

As usual, *Σ-terms* are constructed inductively from a signature Σ and a set of variables V. Given a Σ-algebra A and a variable assignment $\alpha : V \to U_A$, each Σ-term t is evaluated to a unique value $t_{A,\alpha} \in U_A$. We may skip the index α in case t is ground (i.e. t contains no variables).

Terms are used to form *assignment rules* which update a single function value of an algebra. An example built from signature Σ_Q is the assignment rule ExAssign:

$$\texttt{v(a):=next(b)}.$$

Executing rule ExAssign at state Q assigns to the function symbol `v` at the argument $a_Q = 1$ the value $\texttt{next(b)}_Q = 3$ (as all terms in ExAssign are ground, a variable assignment α is not required). The result is a new state Q' equal to Q except for the value at location $(\texttt{v}, [1])$.

The general form of an assignment rule Assign is

$$\texttt{f}(t_1, \ldots, t_n)\texttt{:=}t'$$

where $t_1, \ldots, t_n$ and t' are Σ-terms, and `f` is a n-ary function symbol of Σ. Applied at a state A and a variable assignment α, ASSIGN updates the value of the function symbol `f` at the argument $\bar{a} =_{\text{def}} ({t_1}_{A,\alpha}, \ldots, {t_n}_{A,\alpha})$ by the value $v =_{\text{def}} {t'}_{A,\alpha}$. This update is represented by an *update molecule* $(\mathtt{f}, \bar{a}, v)$.

In general, ASM rules may update more than a single location of A. Multiple updates are represented by a *set* of update molecules, which is called an *update set*. The update set of the assignment rule ASSIGN at state A and variable assignment α is defined as the singleton set

$$\text{ASSIGN}_{A,\alpha} \;\; =_{\text{def}} \;\; \{\, (\mathtt{f}, \bar{a}, v) \,\}$$

with $\bar{a}$ and v as defined above. Such an update set is applied to A by changing the function values of A according to the update molecules in the update set. The formal definition is straightforward and will be given in Sect. 4.3.

4.2 Conditional Rules

An assignment rule may be guarded by a condition, which is represented by a *conditional rule*. For example, the conditional rule EXCOND

```
if (a=b) then a:=next(a)
```

executes the assignment rule at a state A only if the condition `a=b` holds in A.

The condition may be an arbitrary Boolean formula. In technical terms, a Boolean formula ϕ consists of several *term equations* of the form t_1=t_2 connected by the usual Boolean operations $\neg$, $\wedge$, and $\vee$. For a given state A and a variable assignment α, the truth value of ϕ is computed in the obvious way.

The general form of a conditional rule COND is

$$\texttt{if } \phi \texttt{ then } \text{ASSIGN}$$

where ϕ is a Boolean formula and ASSIGN is an assignment rule. Its semantics is obvious: ASSIGN is executed if the condition ϕ is satisfied by A and α. Otherwise the state A is left unchanged. Hence, the update set of COND at A and α is defined as

$$\text{COND}_{A,\alpha} \;\; =_{\text{def}} \;\; \begin{cases} \text{ASSIGN}_{A,\alpha} & \text{, if } \phi \text{ is satisfied by } A \text{ and } \alpha \\ \emptyset & \text{, otherwise.} \end{cases}$$

For technical convenience, we assume every assignment rule as a special conditional rule whose condition holds in every state.

4.3 Parallel Rules

Several conditional rules may be executed simultaneously, which is represented by a *parallel rule*. A simple example is the parallel rule EXPAR

```
par a:=b b:=a endpar.
```

ExPar simultaneously executes both assignment rules (which are special conditional rules, as explained above). Executing ExPar at a state Q yields a new state Q' where the values of `a` and `b` are swapped.

The general form of a parallel rule Par is

$$\texttt{par}\ \text{Cond}_1 \dots \text{Cond}_n\ \texttt{endpar.}$$

where $\text{Cond}_1, \dots, \text{Cond}_n$ are conditional rules. Executing Par at a state A and variable assignment α will result in a simultaneous execution of the updates performed by $\text{Cond}_1, \dots, \text{Cond}_n$. Formally, the update set of Par is defined as

$$\text{Par}_{A,\alpha} \ =_{\text{def}}\ {\text{Cond}_1}_{A,\alpha} \cup \dots \cup {\text{Cond}_n}_{A,\alpha}.$$

An update set Δ (such as $\text{Par}_{A,\alpha}$) then is applied to a state A by changing the function values according to the update molecules in Δ: for each update molecule $(\mathtt{f}, \bar{a}, v) \in \Delta$, the value of `f` at argument $\bar{a}$ is changed to v. The resulting state is denoted by $A \oplus \Delta$. Hence, the functions of $A \oplus \Delta$ are defined as

$$\mathtt{f}_{A\oplus\Delta}(\bar{a}) = \begin{cases} v & \text{, for } (\mathtt{f}, \bar{a}, v) \in \Delta \\ \mathtt{f}_A(\bar{a}) & \text{, otherwise} \end{cases}$$

for each n-ary function symbol `f` and each n-ary argument tuple $\bar{a}$.

Note that $A \oplus \Delta$ is undefined in case Δ is *inconsistent*, i.e. Δ contains two molecules $(\mathtt{f}, \bar{a}, v)$ and $(\mathtt{f}, \bar{a}, v')$ with $v \neq v'$. For example, executing the parallel rule ExPar2

```
par f(x):=u f(y):=v endpar
```

at a state A with $\mathtt{x}_A = \mathtt{y}_A$ and $\mathtt{u}_A \neq \mathtt{v}_A$ yields an inconsistent update set. In that case ExPar2 yields no next-state.

For technical convenience, we assume every conditional rule Cond as a parallel rule `par` Cond `endpar`.

4.4 Choice Rules

Choice rules allow nondeterministic choice of elements of the universe of a state. An example of a choice rule built over the signature Σ_Q is the rule ExChoice:

$$\texttt{choose}\ v\ \texttt{do a:=}v.$$

Executed at state Q, ExChoice chooses a value v from the universe of Q and assigns it to `a`. Hence, ExChoice yields three different possible next-states, one for each of the values 1, 2, and 3. At a state A with an infinite universe U_A, ExChoice yields an infinite number of next-states, one for each element of U. Therefore, choice rules introduce *unbounded nondeterminism*.

A slightly advanced example of a choice rule is ExChoice2:

$$\texttt{choose}\ x, y\ \texttt{with v(}x\texttt{)}\neq y\ \texttt{do v(}x\texttt{):=}y.$$

ExChoice2 assigns the nondeterministically chosen value y to the function symbol v at the nondeterministically chosen argument x. The additional condition $\mathtt{v}(x) \neq v$ restricts the possible values for x and y. In this case, the condition ensures that the newly assigned value differs from the old one.

In general terms, a choice rule introduces quantified variables as known from first-order logic. The general form of a choice rule Choice is

$$\mathtt{choose}\ x_1, \ldots, x_n\ \mathtt{with}\ \phi\ \mathtt{do}\ \textsc{Par}$$

where $x_1, \ldots, x_n$ are variables, ϕ is a Boolean formula, and Par is a parallel rule such that ϕ and Par contain only variables from $\{x_1, \ldots, x_n\}$ (i.e. all variables in Choice are bounded). For a given state A, Choice first nondeterministically chooses a variable assignment α such that ϕ is satisfied by A and α. Then Par is executed by use of the variable assignment α. Consequently, Choice has the potential for infinitely many possible update sets at a state A. Formally, this set of possible update sets is defined as

$$\textsc{Choice}_A \ =_{\text{def}} \ \{ \ \textsc{Par}_{A,\alpha} \mid \exists \alpha : \phi \text{ is satisfied by } A \text{ and } \alpha \ \}.$$

The semantics of a choice rule Choice built over a signature Σ is then defined in terms of a next-state relation $\rightarrow_{\textsc{Choice}}$: for two Σ-algebras A, A' holds $A \rightarrow_{\textsc{Choice}} A'$ iff there is a consistent update set $\Delta \in \textsc{Choice}_A$ such that $A' = A \oplus \Delta$.

4.5 Generalized Syntax and Semantics

The syntax of ASM rules presented above is rather restricted: a choice rule merely contains a collection of simultaneously executed assignment rules. Such a restriction definitely would hamper the practical application of ASM as a specification language.

To cope with this problem, the syntax and semantics of ASM rules may be extended to allow arbitrary nesting of conditional rules, parallel rules, and choice rules. In fact, ASM rules classically are defined to allow such arbitrary nesting [7,5]. An example of such a nested ASM rule over signature Σ_Q is the following ASM rule Nested:

```
par
  if (a=v(a)) then par
    choose x do a:=x
    choose y do b:=y
  endpar
  if (¬a=v(a)) then b:=inc(b)
endpar.
```

In this paper we stick to the simple un-nested version of ASM rules in order to keep the technical details as low as possible. Nevertheless, this restriction is not critical. The expressive power of ASM rules as presented in this paper does not increase by allowing arbitrary nesting: each nested ASM rule such as

NESTED can be canonically transformed to an equivalent un-nested choice rule by shifting `choose`-statements towards the beginning of the rule (we skip the formal construction here). This corresponds to an analogy from first-order logic: every first-order formula built from $\exists$, $\wedge$ and $\vee$ can be transformed to its prenex normal form containing only a single occurrence of $\exists$.

According to their syntax, ASM rules merely seem to be yet another programming language. But in contrast to classical programs, an ASM rule RULE built over a signature Σ may be executed on *arbitrary* Σ-algebras. Therefore, as explained in the introduction, RULE may compute on arbitrary mathematical objects such as vectors and real numbers.

4.6 Nondeterministic ASMs

A nondeterministic ASM resembles a nondeterministic algorithm except for the next-state relation which is explicitly given by a choice rule. More precisely, a nondeterministic ASM M consists of

- a signature Σ_M,
- a set of Σ_M-algebras $\mathcal{S}_M$, closed under isomorphism (the states of M),
- a set $\mathcal{I}_M \subseteq \mathcal{S}_M$, closed under isomorphism (the initial states of M),
- a choice rule CHOICE built over the signature Σ_M.

The next-state relation of M, denoted by $\rightarrow_M$, is the restriction of $\rightarrow_{\text{CHOICE}}$ to the states of M. Analogously to nondeterministic algorithms, a *run of M* is a sequence $A_0 A_1 A_2 \ldots$ of states of M with $A_0 \in \mathcal{I}_M$ and $A_i \rightarrow_M A_{i+1}$ for all indices i.

5 The Equivalence Theorem

In this section we present the main result of this paper. The following theorem states that the class of nondeterministic algorithms (as introduced in Section 3) and the class of nondeterministic ASMs (as introduced in Section 4) are equivalent:

Theorem 1 (Equivalence Theorem). *Nondeterministic algorithms and nondeterministic ASMs describe the same set of transition systems.*

We present the proof in the next section.

Theorem 1 confirms that the notion of nondeterministic ASMs and the notion of nondeterministic algorithms may be used interchangeably. In particular, interesting properties of nondeterministic ASMs can be identified by examining nondeterministic algorithms. As an example, for each nondeterministic algorithm $\mathcal{N}$, reverting the next-state relation $\rightarrow_\mathcal{N}$ yields another nondeterministic algorithm. Hence, nondeterministic algorithms are reversible. This fact is proven by verifying the Axioms N1–N5 for the reverse of $\mathcal{N}$. This task is quite simple, as the Axioms N1–N5 are highly symmetric with respect to the next-state relation $\rightarrow_\mathcal{N}$.

According to the Theorem 1, the following corollary follows immediately:

Corollary 1 (Reversibility). *For each nondeterministic ASM M there exists a nondeterministic ASM M^{-1} such that $\rightarrow_{(M^{-1})} = (\rightarrow_M)^{-1}$.*

Alternatively, Corollary 1 can also be proven without Theorem 1 by constructing for each ASM rule its reverse rule. For instance, the reverse of the rule "`a:=next(a)`" is the rule "`choose` x `with a=next(`x`) do a:=`x". However, the proof based on Theorem 1 is considerably simpler, as there are no such syntactical constructions involved. This points out the profit of Theorem 1: the notion of nondeterministic algorithms allows to examine nondeterministic ASMs without any syntactic overhead.

6 Proof of the Equivalence Theorem

In this section we present the proof of Theorem 1. The proof divides into two parts: firstly, we show that every nondeterministic ASM represents a nondeterministic algorithm. Secondly, we show that every nondeterministic algorithm can be represented by a nondeterministic ASM.

The first part is fairly simple. Let $\mathcal{M}$ be a nondeterministic ASM with CHOICE its choice rule. One only needs to verify that $\mathcal{M}$ satisfies the axioms N1–N5: Axioms N1 and N2 are satisfied by the definition of nondeterministic ASMs. Axiom N3 and N4 are properties of the semantics of CHOICE that are easily verified. Axiom N5 holds due to the fact that in each step (A, A') of $\mathcal{M}$, CHOICE accesses and modifies only a bounded substate of A.

The second part of the proof requires considerably more effort. For the rest of this section, let $\mathcal{N}$ be a nondeterministic algorithm, and let $\mathcal{W}$ be a bounded-work witness for $\mathcal{N}$ (see axiom N5). In the following we show that there exists a nondeterministic ASM $\mathcal{M}$ that represents $\mathcal{N}$. Due to the significant difference between Gurevich's bounded exploration axiom and our bounded work axiom, the proof employs a couple of new argumentations.

We start with a lemma presenting a close connection between completions of substeps and the $\oplus$-operator:

Lemma 1. *Let $(M, M') \in \mathcal{W}$ and let $A, A' \in \mathcal{S}_{\mathcal{N}}$. Then (A, A') is a completion of (M, M') iff $M \subseteq A$ and $A' = A \oplus M'$.*

Proof. ($\Rightarrow$) Let E be a substate disjoint from M and M' such that $(A, A') = (M \cup E, M' \cup E)$. Obviously $M \subseteq A$. Further show that $A \oplus M' = (A \setminus M) \cup M'$. As $A' = E \cup M' = (A \setminus M) \cup M'$, this implies $A' = A \oplus M'$. ($\Leftarrow$) For $E =_{\text{def}} A \setminus M$, show that E and M' are disjoint and that $(A, A') = (M \cup E, M' \cup E)$. □

The next lemma states that parallel rules preserve isomorphisms between states. For a parallel rule PAR, let $\tau(\text{PAR}, A, \alpha) =_{\text{def}} A \oplus \text{PAR}_{A,\alpha}$ denote the next state computed by PAR at state A and variable assignment α.

Lemma 2. *Let* PAR *be a parallel rule over a signature Σ, let A, B be Σ-algebras with an isomorphism $i : A \rightarrow B$, and let α be a variable assignment with values in U_A. Then $i : \tau(\text{PAR}, A, \alpha) \rightarrow \tau(\text{PAR}, B, i \circ \alpha)$ also is an isomorphism.*

Proof. Follows by examining the semantics of PAR. □

The following Lemma presents the main part of the proof: for each step (A, A') of $\mathcal{N}$, a choice rule CHOICE can be constructed such that (A, A') is a step of CHOICE, and all other steps of CHOICE are also steps of $\mathcal{N}$.

Lemma 3. *Let (A, A') be a step of $\mathcal{N}$. Then there is a choice rule* CHOICE *such that $A \to_{\text{CHOICE}} A'$, and $B \to_{\text{CHOICE}} B'$ implies $B \to_{\mathcal{N}} B'$ for all states B, B' of $\mathcal{N}$.*

Proof. By Axiom N5, there exists a substep $(M, M') \in \mathcal{W}$ such that (A, A') is a completion of (M, M'). We use (M, M') to construct CHOICE.

Let $V \subseteq U_A$ denote the elements of the universe of A occurring in M and M'. As the size of M and M' is bounded, V is finite. For each element $v \in V$, choose a unique variable x^v. Define the Boolean formulas ϕ and ψ by

$$\phi =_{\text{def}} \bigwedge_{v \neq w \in V} x^v \neq x^w, \quad \psi =_{\text{def}} \bigwedge_{(\mathtt{f},[u_1,\ldots,u_n],v) \in M} \mathtt{f}(x^{u_1}, \ldots, x^{u_n}) = x^v.$$

Construct for each molecule $(\mathtt{f}, [u_1, \ldots, u_n], v) \in M'$ the assignment rule $\mathtt{f(}x^{u_1}, \ldots, x^{u_n}\mathtt{):=}x^v$. Combine all of these assignment rules to a single parallel rule PAR. Let $v_1, \ldots, v_m$ be the elements in V. Define CHOICE as

$$\mathtt{choose}\ x^{v_1}, \ldots, x^{v_m}\ \mathtt{with}\ \phi \wedge \psi\ \mathtt{do}\ \text{PAR}.$$

According to axiom N5, the size of M and M' is bounded by a constant k. Consequently, the size of CHOICE also is bounded by a constant c.

We first show that $A \to_{\text{CHOICE}} A'$: let α be the variable assignment defined by $\alpha(x^v) = v$ for all $v \in V$. Then ϕ and ψ are satisfied by A and α. Further, by construction of PAR, we have $\text{PAR}_{A,\alpha} = M'$. Therefore, $A \to_{\text{CHOICE}} A \oplus M'$. By Lemma 1, $A \oplus M' = A'$.

Finally, let B, B' be states of $\mathcal{N}$ such that $B \to_{\text{CHOICE}} B'$. We have to show that $B \to_{\mathcal{N}} B'$. As $B \to_{\text{CHOICE}} B'$, there is a variable assignment β such that ϕ and ψ are satisfied at B and β, and $B' = \tau(\text{PAR}, B, \beta)$.

Construct from B a isomorphic state C by bijectively replacing, for each $v \in V$, the element $\beta(x^v)$ by v, and by replacing every other element from U_B by a new element not contained in U_B. This construction is well-defined as ϕ is satisfied by B and β. Let $i : B \to C$ be the corresponding isomorphism. As ψ is satisfied by B and β, one can show that $M \subseteq C$. By construction of PAR, we have $\text{PAR}_{C,i\circ\beta} = M'$.

Let $C' =_{\text{def}} \tau(\text{PAR}, C, i \circ \beta)$. By Lemma 2, $i : B' \to C'$ is an isomorphism. As B, B' are states of $\mathcal{N}$, Axiom N4 implies that C, C' also are states of $\mathcal{N}$. As $\text{PAR}_{C,i\circ\beta} = M'$ and by the definition of $\tau(\text{PAR}, C, i\circ\beta)$, we conclude $C' = C \oplus M'$. As $M \subseteq C$, Lemma 1 implies that (C, C') is a completion of (M, M'). By Axiom N5, (C, C') is a step of $\mathcal{N}$. As $i : B \to C$ and $i : B' \to C'$ are isomorphisms, Axiom N4 implies that (B, B') also is a step of $\mathcal{N}$. □

The last lemma states that a finite set of choice rules can be united to a single choice rule:

Lemma 4. *Let* $\text{Choice}_1, \ldots, \text{Choice}_n$ *be choice rules. Then there exists a single choice rule* Union *such that* $\rightarrow_{\text{Choice}} = \rightarrow_{\text{Choice}_1} \cup \cdots \cup \rightarrow_{\text{Choice}_n}$. Choice *is the* union *of* $\text{Choice}_1, \ldots, \text{Choice}_n$.

Proof. For the sake of simplicity, we present Choice in form of a nested nondeterministic ASM rule. However, as stated in Section 4, Choice may be transformed to an equivalent un-nested choice rule. The desired rule Choice is

$$
\begin{aligned}
&\texttt{choose } x_0, \ldots, x_n \\
&\texttt{with } \bigvee_{1 \leq i \leq n} (x_0 = x_i \wedge \bigwedge_{1 \leq j \leq n, j \neq i} x_0 \neq x_j) \texttt{ do par} \\
&\quad \texttt{if } (x_0 = x_1) \texttt{ then } \text{Choice}_1 \\
&\quad \ldots \\
&\quad \texttt{if } (x_0 = x_n) \texttt{ then } \text{Choice}_n \\
&\texttt{endpar.}
\end{aligned}
$$

The choice condition ensures that the value of x_0 is equal to the value of one and only one x_i, $i = 1, \ldots, n$. Hence, each variable assignment satisfying the choice condition satisfies exactly one of the guards $(x_0 = x_1)$, $\ldots$, $(x_0 = x_n)$.

A technical remark: as the above choice condition cannot be satisfied at states with singleton universes, the above construction works only for states whose universe contains at least two elements. However, this issue can be resolved by a rather technical extension of the above construction. Due to the lack of space and due to the practical irrelevance of singleton universes we skip this extension here. □

The final proof combines the lemmata presented above:

Proof (of Theorem 1). For all steps (A, A'), derive a choice rule $\text{Choice}_{(A,A')}$ by applying Lemma 3. Let $\mathcal{C}$ be the set of all of these choice rules. By construction (see proof of Lemma 3), the size of all choice rules in $\mathcal{C}$ is bounded by a constant c. WLOG we may assume that the programs in $\mathcal{C}$ contain only finitely many different variables. Hence, as $\mathcal{C}$ contains only symbol sequences of bounded length over a finite alphabet, $\mathcal{C}$ is finite.

By Lemma 4, let Choice be the union of all choice rules in $\mathcal{C}$. Then for all states A, A' of $\mathcal{N}$, we have $A \rightarrow_{\mathcal{N}} A'$ iff $A \rightarrow_{\text{Choice}} A'$: ($\Rightarrow$) Let $A \rightarrow_{\mathcal{N}} A'$. By Lemma 3, $A \rightarrow_{\text{Choice}_{(A,A')}} A'$. By Lemma 4, $A \rightarrow_{\text{Choice}} A'$. ($\Leftarrow$) Let $A \rightarrow_{\text{Choice}} A'$. By Lemma 4, there is a step (B, B') of $\mathcal{N}$ with $A \rightarrow_{\text{Choice}_{(B,B')}} A'$. By Lemma 3, $A \rightarrow_{\mathcal{N}} A'$. □

7 Conclusion

The theory of ASMs suggests a comprehensive and quite general approach to the notion of "algorithm". The basic idea is to represent each state of an algorithm by an algebra. A number of variants of ASMs have been identified, among them sequential, nondeterministic, parallel, and interactive versions. The deeper understanding of all variants of ASMs requires their characterization independently of any concrete syntax. This has been achieved for many of them, including sequential, parallel, and interactive versions.

In this paper we characterized the class of unbounded-nondeterministic ASMs. To this end we axiomatized the class of nondeterministic algorithms and showed that this class is equivalent to unbounded-nondeterministic ASMs. Surprisingly, the definition of nondeterministic algorithms turns out to be considerably simpler than the semantics of unbounded-nondeterministic ASMs. Due to this fact, we were able to prove the reversibility of unbounded-nondeterministic ASMs with nearly no effort. We intend to identify and to prove further interesting properties (such as linear speedup [8]) in a similar way.

Acknowledgement. We are grateful to the anonymous referees for their helpful comments.

References

1. Blass, A., Gurevich, Y.: Abstract State Machines Capture Parallel Algorithms. ACM Trans. Comput. Logic 4(4), 578–651 (2003)
2. Blass, A., Gurevich, Y.: Ordinary Interactive Small-Step Algorithms, parts I, II, III ACM Trans. Comput. Logic (to appear)
3. Blass, A., Gurevich, Y., Rosenzweig, D., Rossman, B.: General Interactive Small Step Algorithms. Technical Report MSR-TR-2005-113, Microsoft Research (August 2006)
4. Börger, E.: The Origins and the Development of the ASM Method for High Level System Design and Analysis. Journal of Universal Computer Science 8(1), 2–74 (2002)
5. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, Heidelberg (2003)
6. Gurevich, Y.: A New Thesis. Abstracts, American Mathematical Society p. 317 (1985)
7. Gurevich, Y.: Evolving Algebras 1993: Lipari Guide. In: Börger, E. (ed.) Specification and Validation Methods, pp. 9–36. Oxford University Press, Oxford (1995)
8. Gurevich, Y.: Sequential Abstract State Machines Capture Sequential Algorithms. ACM Transactions on Computational Logic 1(1), 77–111 (2000)
9. Gurevich, Y., Yavorskaya, T.: On Bounded Exploration and Bounded Nondeterminism. Technical Report MSR-TR-2006-07, Microsoft Research (January 2006)
10. ITU-T. SDL Formal Semantics Definition. ITU-T Recommendation Z.100 Annex F, International Telecommunication Union (November 2000)
11. Reisig, W.: On Gurevich's Theorem on Sequential Algorithms. Acta Informatica 39(5), 273–305 (2003)
12. Stärk, R., Schmid, J., Börger, E.: Java and the Java Virtual Machine: Definition, Verification, Validation. Springer, Heidelberg (2001)

Parametric (Co)Iteration vs. Primitive Direcursion

Johan Glimming

Department of Numerical Analysis and Computer Science
Stockholm University, Sweden
glimming@kth.se

Abstract. Freyd showed that in certain CPO-categories, locally continuous functors have minimal invariants, which possess a structure that he termed dialgebra. This gives rise to a category of dialgebras and homomorphisms, where the minimal invariants are initial, inducing a powerful recursion scheme (direcursion) on a cpo. In this paper, we identify a problem appearing when translating (co)iterative functions (on a fixed parameterised datatype) to direcursion (on the same datatype), and present a solution to this problem as a recursion scheme (primitive direcursion), generalising and symmetrising primitive (co)recursion for endofunctors. To this end, we give a uniform technique for translating (co)iterative maps into direcursive maps. This immediately gives a plethora of examples of direcursive functions, improving on the situation in the literature where only a few examples have appeared. Moreover, a technical trick proposed in a POPL paper is avoided for the translated maps. We conclude the paper by applying the results to a denotational semantics of Abadi and Cardelli's typed object calculus, and linking them to previous work on higher-order coalgebra and to bisimulations.

1 Introduction

Solutions to recursive domain equations involving function spaces can be given as initial $\hat{G}$-algebras in suitable categories, where $\hat{G}$ is an endofunctor given by symmetrising G. This was shown by Freyd [12,13] (based on work by Smyth and Plotkin [30]) and later refined by Fiore [10] in a framework of enriched category theory. Initial $\hat{G}$-algebras (also called *dialgebras*) generalise usual algebras and coalgebras. Moreover, a recursion principle arises for $\hat{G}$-algebras. This principle is hereafter called *direcursion* and it is the topic of this paper. It has previously been investigated both in loc. cit. and as a tool for functional programming (with associated proof principle) (see e.g. [21,36,8]). Some notable theoretical results are: the reduction to inductive types as given by Freyd in his seminal paper [12] and the relationship to dinaturality [13], the derivation of an associated proof principle [23,24], programming examples dealing with higher-order abstract syntax [36], lambda calculus interpreters [21], and circular datatypes [8]. But direcursion remains relatively unexplored as regards termination properties and its relationships to other recursion schemes (and programming examples have so far been rather scarce). In this paper we begin to remedy this situation.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 257–278, 2007.
© Springer-Verlag Berlin Heidelberg 2007

We will here investigate the relationships between (co)iteration and direcursion for a *fixed datatype*. With (co)iteration we mean the unique homomorphisms associated to the initial (final) $\hat{G}(\mu\hat{G}, _)$-(co)algebras by Bekič's Lemma, i.e. (co)iterative maps on this particular parameterised datatype. Since the carrier of this (co)algebra coincides with the solution $\mathcal{O} = \mu\hat{G}$, we ask how these schemes compare to direcursion for same functor G. Our main result is to show that by generalising direcursion (by precomposing with an injection map or postcomposing with a projection map), we can express all such (co)iterative maps as a canonical direcursive map such that the same computation is carried out at every stage. We call this generalisation *primitive direcursion* since it is primitive recursion for symmetric functors $\hat{G}$, i.e. it is simultaneously primitive recursion and primitive corecursion for recursive types. The latter two principles have been studied in previous work by Meertens [19] and Uustalu and Vene [33]. Primitive direcursion simultaneously gives both schemes in the special case when the bifunctor is constant in its contravariant argument, but also transports those schemes to cover domain equations involving function spaces as well, i.e. to the mixed variant case, for which it has not previously been considered.

The paper is structured as follows: the first few pages survey necessary background material. Next, we develop primitive direcursion from first principles. In the same section, we give the translation of certain iterative/coiterative maps into primitive direcursive maps, which is our main result. This result can be viewed as an internalised version of Bekič's Lemma. The fourth section exemplifies the results in the setting of program semantics, and the last section gives our conclusions.

2 Mathematical Preliminaries

In this paper we are essentially considering the category $\mathsf{CPPO}_{\perp!}$ of directed complete pointed cpos and *strict* continuous maps (or subcategories with similar properties, e.g. Scott domains and strict maps), as defined in e.g. [3,31]. Such a category is used when solving domain equations, for instance in [30] based on [29]. We will in this paper assume an ambient category $\mathcal{C}$ which abstracts from $\mathsf{CPPO}_{\perp!}$ exactly the properties that we require here. These are our assumptions on $\mathcal{C}$:

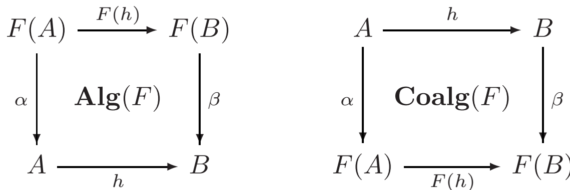
- $\mathcal{C}$ has products $\times$ and coproducts $+$.
- $\mathcal{C}$ is algebraically compact, so that each (suitably qualified) endofunctor has an initial algebra and a final coalgebra, and these are canonically isomorphic [13], i.e. the unique homomorphism from the inverse of the initial algebra to the final coalgebra is an isomorphism. In particular, this family of endofunctors is assumed to include the functors considered in this paper. It follows also that $\mathcal{C}$ has a zero object $0 \cong 1$, also known as a biterminator.
- $\mathcal{C}$ has regular initial dialgebras [12], as will be detailed below. Having regular free dialgebras is in fact a consequence of algebraic compactness [13]. In a weaker axiomatisation where we require merely free dialgebras for a class

of functors instead of algebraic compactness, this condition must however remain explicit, see [12,14] for examples.

– $\mathcal{C}$ is symmetric monoidal closed with tensor $\otimes$ and unit 1. The right adjoint to this tensor is written $\multimap$ (with the natural isomorphisms curry and $\mathsf{uncurry}$, and counit eval).
– $\mathcal{C}$ has a generator I, i.e. for all $f, g \in C(A, B)$ we have $f = g$ iff for all $i : I \to A$ we have that $f \circ i = g \circ i$.

An example of such a category is $\mathsf{CPPO}_{\perp!}$ itself, in which case the tensor $\otimes$ is smash product (with right adjoint strict function space, with curry etc). The product $\times$, on the other hand, is the cartesian product of cpos (thus we have merely *weak exponentials* in the sense of $\otimes$ being left adjoint to the function space rather than the product), and coproduct is coalesced sum (i.e. the least elements are identified in contrast to e.g. separated sum where a new one is adjoined). The generator for this category is given by the Sierpinski space $I = \{\perp, \top\}$ (so it is in particular not well pointed since $I \ncong 1$). In $\mathsf{CPPO}_{\perp!}$ the family of endofunctors considered above are the $\mathsf{CPPO}_{\perp!}$-enriched (i.e. locally continuous) functors, i.e. functors $F : \mathcal{C} \to \mathcal{C}$ given by maps $|\mathcal{C}| \to |\mathcal{C}|$ on objects (writing $|\mathcal{C}|$ for the class of objects), and arrow maps given on homsets by a *Scott-continuous* mapping $\mathcal{C}(A, B) \mapsto \mathcal{C}(FA, FB)$ for each $A, B \in |\mathcal{C}|$, such that composition and identities are preserved. Such functors are alternatively called *locally continuous*. Note that for bifunctors this means that each section is locally continuous. In particular, for mixed variant functors $F : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ we require that $\mathcal{C}(B, A) \times \mathcal{C}(A', B') \mapsto \mathcal{C}(F(A, A'), F(B, B'))$ has said property instead (see [3]).

Definition 1 (Algebra, Coalgebra)
Given a covariant functor $F : \mathcal{C} \to \mathcal{C}$ we say that an arrow $\alpha : F(A) \to A$ is an F-algebra with carrier A. The dual notion is that of F-coalgebra, i.e. reversed arrows $\alpha : A \to F(A)$. The arrows between (co)algebras are F-homomorphisms, i.e. arrows h such that the left or right diagram below commute (in the respective case):



We now recall some results concerning solutions to recursive domain equations in $\mathsf{CPPO}_{\perp!}$. These results serve to motivate our axiomatised category $\mathcal{C}$. Given a locally continuous endofunctor F, we construct a diagram by iterating the functor, beginning at the zero object 0. There are then unique morphisms $0 \to F0$ and $F0 \to 0$ so we can construct systems giving both a limit and a colimit. The limit and colimit coincide in this case, and is denoted μF. This motivates the algebraic compactness requirement for $\mathcal{C}$. Further details are given in e.g. [30], [25], [3]. One particular result (not detailed here) is that μF carries an initial

algebra and also a final coalgebra (arising from considering respectively cones and cocones, see loc. cit.). This is an important consequence since it together with the following lemma shows that in $\mathsf{CPPO}_{\bot!}$ we can solve domain equations (for locally continuous endofunctors) up to isomorphism:

Proposition 1 (Lambek's Lemma). *An initial algebra* $(\mathcal{O}_F, \iota_F)$ *is an isomorphism* $\mathcal{O}_F \cong F(\mathcal{O}_F)$. *Dually for final coalgebra* $(\mathcal{O}_F, \iota_F^\circ)$

The notation ι_F and ι_F° is used for the initial algebra and final coalgebra respectively (we drop suffixes when possible). Our next assumption for $\mathcal{C}$ is that it has regular initial dialgebras. The fact that $\mathsf{CPPO}_{\bot!}$ satisfies this condition is due to Freyd [12] (but see also pioneering work by [30] in the more concrete setting of a subcategory of embedding-projection pairs). We survey Freyd's work here, in particular by recalling that, for a (mixed-variant) functor F, an object X is called F-invariant if there is an isomorphism $\alpha : F(X) \cong X$. If *fix* $e.\, \alpha \circ F(e) \circ \alpha^{-1} \in A \to A$ is the identity, X is called special F-invariant. If it is the only idempotent map $A \to A$ for which $e \circ \alpha = \alpha \circ F(e)$, it is called *minimal invariant* [12]. Freyd [12] showed that in $\mathsf{CPPO}_{\bot!}$ there exists an F-invariant object for every locally continuous functor that is minimal in this sense. A corollary of this result is the recursion principle that we call direcursion, which relies on first generalising the notion of (co)algebra to mixed-variance functors:

Definition 2 (Dialgebra). *A* G-dialgebra *for bifunctor* $G : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ *is a quadruple* (A, B, ϕ, ψ) *of objects* A, B *and associated arrows* $\phi : G(B, A) \to A$ *and* $\psi : B \to G(A, B)$.

Note that in the case when G is dummy in its contravariant argument, i.e. an endofunctor F on $\mathcal{C}$, this definition gives precisely that (A, ϕ) is an F-algebra and, independently, that (B, ψ) is a F-coalgebra. Dialgebras for a bifunctor G form a category $Dialg(G)$ with the following morphisms:

Definition 3 (Dialgebra Map). *Given* G*-dialgebras* (A, B, ϕ, ψ) *and* (A', B', ϕ', ψ'), *a* G-homomorphism *(or* dialgebra map/dimap*) is a pair of arrows* $(h : A \to A', g : B' \to B)$ *such that the following diagrams commute:*

$$
\begin{array}{ccc}
G(B,A) & \xrightarrow{\phi} & A \\
{\scriptstyle G(g,h)}\downarrow & \equiv & \downarrow{\scriptstyle h} \\
G(B',A') & \xrightarrow[\phi']{} & A'
\end{array}
\qquad\qquad
\begin{array}{ccc}
B & \xrightarrow{\psi} & G(A,B) \\
{\scriptstyle g}\uparrow & \equiv & \uparrow{\scriptstyle G(h,g)} \\
B' & \xrightarrow[\psi']{} & G(A',B')
\end{array}
$$

An initial dialgebra for a bifunctor G is a dialgebra (A, B, ϕ, ψ) such that for any other G-dialgebra (A', B', ϕ', ψ') there is a unique dialgebra map $(h : A \to A', g : B' \to B)$. The existence of initial dialgebras in $\mathsf{CPPO}_{\bot!}$ was established by Freyd:

Theorem 1 (Existence of Initial Dialgebras [12]). $\mathsf{CPPO}_{\perp!}$ *has initial dialgebras for every locally continuous bifunctor* $G : {\mathsf{CPPO}_{\perp!}}^{op} \times \mathsf{CPPO}_{\perp!} \to \mathsf{CPPO}_{\perp!}$*. In addition, initial dialgebras in* $\mathsf{CPPO}_{\perp!}$ *are of the form* $(\mathcal{O}_G, \mathcal{O}_G, \iota_G, \iota_G^\circ)$ *where* $\iota_G \circ \iota_G^\circ = id$ *and* $\iota_G^\circ \circ \iota_G = id$*.*

Proof. See e.g. [24]. □

The second property in the theorem is what Freyd termed *regular initial dialgebra*. The existence of such dialgebras was one of the conditions we listed for $\mathcal{C}$, and it will be frequently used in this paper. Note also that usual initial algebras and final coalgebras for endofunctors follows from Freyd's condition since the difunctor can be constant in its negative argument, e.g. $G(Y,X) = 1 + X$. (In such cases the intertwined diagrams for direcursion instead become two independent diagrams for iteration and coiteration, respectively.) Moreover, Freyd's condition has the following consequence, which is the recursion principle studied in this paper:

Definition 4 (Direcursion [12])
Let $(\mathcal{O}, \mathcal{O}, \iota_G, \iota_G^\circ)$ *be the initial* G*-dialgebra and suppose* (A, B, ϕ, ψ) *is some other* G*-dialgebra. Then there exists unique morphisms* $g : \mathcal{O} \longrightarrow A$ *and* $h : B \longrightarrow \mathcal{O}$ *such that the following diagrams commute:*

$$\begin{array}{ccc} G(\mathcal{O},\mathcal{O}) & \xrightarrow{\iota_G} & \mathcal{O} \\ {\scriptstyle G(h,g)}\downarrow & \equiv & \downarrow{\scriptstyle g} \\ G(B,A) & \xrightarrow[\phi]{} & A \end{array} \qquad \begin{array}{ccc} \mathcal{O} & \xrightarrow{\iota_G^\circ} & G(\mathcal{O},\mathcal{O}) \\ {\scriptstyle h}\uparrow & \equiv & \uparrow{\scriptstyle G(g,h)} \\ B & \xrightarrow[\psi]{} & G(A,B) \end{array} \qquad \text{(direc-Prop)}$$

We introduce the notation $(\![\phi,\psi]\!)_G \stackrel{def}{=} g$ *and* $[\!(\phi,\psi)\!]_G \stackrel{def}{=} h$ *whenever the conditions for* ϕ *and* ψ *are satisfied.*

There are a number of standard properties that can be easily established. We survey them here:

Lemma 1 (Basic properties of direcursion [21]). *Let* $(\mathcal{O}, \mathcal{O}, \iota_G, \iota_G^\circ)$ *be the initial* G*-dialgebra and suppose* (A, B, ϕ, ψ) *is some other* G*-dialgebra. Then the following is true:*

$$(\![\phi,\psi]\!) \circ \iota_G = \phi \circ G([\!(\phi,\psi)\!], (\![\phi,\psi]\!)) \qquad \text{(direc-Self)}$$
$$\iota_G^\circ \circ [\!(\phi,\psi)\!] = G((\![\phi,\psi]\!), [\!(\phi,\psi)\!]) \circ \psi$$

$$id = (\![\iota_G, \iota_G^\circ]\!) \qquad \text{(direc-Refl)}$$
$$id = [\!(\iota_G, \iota_G^\circ)\!]$$

$$A = B \text{ and } \psi \circ \phi = id_{G(A,A)} \text{ implies } [\!(\phi,\psi)\!] \circ (\![\phi,\psi]\!) = id_{\mathcal{O}} \qquad \text{(direc-Retract)}$$

Proof. The two first properties follow directly. The third one follows by pasting the right square for direcursion below the left one, and vice versa. □

Lemma 2 (Direcursion Fusion [21]). *Let* $(\mathcal{O}, \mathcal{O}, \iota_G, \iota_G^\circ)$ *be the initial G-dialgebra. Suppose that* $(A, B, \phi : G(B, A) \to A, \psi : B \to G(A, B))$ *and* $(A', B', \phi' : G(B', A') \to A', \psi' : B' \to G(A', B'))$ *are both G-dialgebras. For every dimap* $g : A \to A', h : B' \to B$ *such that*

$$g \circ \phi = \phi' \circ G(h, g)$$

and

$$\psi \circ h = G(g, h) \circ \psi'$$

we have the following property

$$g \circ (\![\phi, \psi]\!) = (\![\phi', \psi']\!) \text{ and } [\![(\phi, \psi)]\!] \circ h = [\![(\phi', \psi')]\!] \qquad \text{(direc-Fusion)}$$

2.1 Symmetric Functors, Bekič's Lemma and Parameterised (Co)Algebras

As noted by Freyd [12] (and further detailed in [9]), we can by introducing symmetric endofunctors $\mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}^{op} \times \mathcal{C}$, view a dialgebra as an algebra on the product category. We have in particular the following result:

Lemma 3 ([12]). *There is a bijective correspondence between functors* $F : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ *and symmetric functors* $\hat{F} : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}^{op} \times \mathcal{C}$*, where*

$$\begin{aligned} \hat{F}(X, Y) &= (F(Y, X), F(X, Y)) \\ \hat{F}(f, g) &= (F(g, f), F(f, g)) \end{aligned}$$

Proof. This is established e.g. using the notion of involutions (self-dual functors) and a category $\hat{\mathcal{C}}$ of involutory objects (due to John Power, see Fiore [9] for details). □

In particular, this implies that direcursion can alternatively be formulated using $\hat{F}$, in which case the maps $f = (\![\phi, \psi]\!)$ and $g = [\![(\phi, \psi)]\!]$ are equivalently and more abstractly defined as follows in the category $\mathcal{C}^{op} \times \mathcal{C}$ (see [9]):

$$\begin{array}{ccc} \hat{G}(\mathcal{O}, \mathcal{O}) & \xrightarrow[(\iota_G, \iota_G^\circ)]{} & (\mathcal{O}, \mathcal{O}) \\ {\scriptstyle \hat{G}(f,g)} \big\downarrow & \equiv & \big\downarrow {\scriptstyle (f,g)} \\ \hat{G}(A, B) & \xrightarrow[(\phi, \psi)]{} & (A, B) \end{array}$$

Given a bifunctor $F : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ and an object $X \in |\mathcal{C}|$ we have an endofunctor $F(X, _) : \mathcal{C} \to \mathcal{C}$. This means that we can consider two different equation systems as follows (where we already have shown that the initial F-dialgebra gives a solution for the left-hand side (lhs) system).

$$\begin{cases} X \cong F(Y, X) = \pi_1(\hat{F}(X, Y)) \\ Y \cong F(X, Y) = \pi_2(\hat{F}(X, Y)) \end{cases} \qquad \begin{cases} X \cong F(\mu F(X, _), X) \\ Y \cong \mu F(X, _) \end{cases}$$

The solution to the rhs system is said to be parameterised, and it is not immediately clear if it has the same solutions as the lhs one. However, we have:

Lemma 4 (Bekič's Lemma [4]). *Let $F : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ be a locally continuous endofunctor with initial dialgebra $(\mathcal{O}, \mathcal{O}, \iota, \iota^\circ)$. Then we have that $\mathcal{O} \cong \mu F(\mathcal{O}, _)$.*

Proof. E.g. [3,9]. □

The endofunctors of the form $F(A, _)$ in this way have initial $F(A, _)$-algebras where A is the parameter.

2.2 Primitive (Co)Recursion and (Co)Iteration

A datatype within functional programming is typically modelled by an initial algebra for suitable functor F, and this work has treated also the functor as a parameter of programs. Since each functor F on $\mathcal{C}$ has an initial algebra $\mathcal{O}$, there is a unique homomorphism into $\mathcal{O}$ from any other object A for which there is a structure $\phi : FA \to A$. We call these F*-iterative maps*, written $(\![\phi]\!)_F$ for easy reference. The dually constructed maps are called F*-coiterative maps*, and are written $[\!(\psi)\!]_F$. From these, Meertens [19] constructed F-primitive recursion:

Theorem 2 (F-primitive recursion). *Suppose $(\mu F, \iota_F)$ is the initial F-algebra. For every morphism $\phi : F(\mu F \times A) \to A$ there exists a unique morphism h (called* F-primitive recursion*) such that the following diagram commutes:*

$$
\begin{array}{ccc}
F(\mu F) & \xrightarrow{\iota_F} & \mu F \\
\Big\downarrow{\scriptstyle \langle id, h\rangle} & \equiv & \Big\downarrow{\scriptstyle h} \\
F(\mu F \times A) & \xrightarrow[\phi]{} & A
\end{array}
$$

Proof. For existence define a map $\phi' : F(\mu F \times A) \to \mu F \times A$ by $\phi' = \langle \iota \circ F(\pi_1), \phi \rangle$. Hence there is a coiterative map $(\![\phi']\!)_F$ which satisfies the diagram when post-composed with π_2. For uniqueness suppose $h = \phi \circ \langle id, h \rangle \circ \iota_F^\circ$. But then $\langle \pi_1 \circ (\![\phi']\!)_F, h \rangle$ is a homomorphism into $\mu F \times A$, and hence, by properties of pairing in $\mathcal{C}$ together with initiality of $(\mu F, \iota_F)$, we have $h = \pi_2 \circ (\![\phi']\!)_F$. □

This result dualises into a scheme useful for coalgebraic datatypes, as was shown by Uustalu et al [33,34]. The definition of such F*-primitive corecursion* is dual to the above construction, and so is the associated proof.

3 Primitive Direcursion

We will in this section consider a symmetrised version of primitive (co)recursion, and prove a number of basic result for this recursion principle, followed by our

main result. We provide a quite detailed exposition here, and develop primitive direcursion from first principles. Many proofs can alternatively be viewed as a special case of primitive recursion with the ambient category $\mathcal{C}^{op} \times \mathcal{C}$, i.e. $\hat{F}$-primitive recursion, and are therefore omitted. Note that direcursion itself by a similar argument is a special case of iteration, if one moves to the symmetrised category $\mathcal{C}^{op} \times \mathcal{C}$, as we mentioned in a previous section. However, we include some such proofs to highlight which properties of $\mathcal{C}$ they rely on (including the regularity assumption).

Theorem 3 (Primitive Direcursion). *Let* $(\mathcal{O}, \mathcal{O}, \iota_G, \iota_G^\circ)$ *be the initial* G*-dialgebra. Let* A *and* B *be two objects and* $\phi : G(\mathcal{O} + B, O \times A) \to A$ *and* $\psi : B \to G(\mathcal{O} \times A, \mathcal{O} + B)$ *two morphisms. Then there exist* $g : \mathcal{O} \longrightarrow A$ *and* $h : B \longrightarrow \mathcal{O}$ *such that the following diagrams commute:*

$$\begin{array}{ccccccc} G(\mathcal{O},\mathcal{O}) & \xrightarrow{\iota_G} & \mathcal{O} & & \mathcal{O} & \xrightarrow{\iota_G^\circ} & G(\mathcal{O},\mathcal{O}) \\ {\scriptstyle G([id,h],\langle id,g\rangle)}\downarrow & \equiv & \downarrow{\scriptstyle g} & & {\scriptstyle h}\uparrow & \equiv & \uparrow{\scriptstyle G(\langle id,g\rangle,[id,h])} \\ G(\mathcal{O}+B,\mathcal{O}\times A) & \xrightarrow[\phi]{} & A & & B & \xrightarrow[\psi]{} & G(\mathcal{O}\times A,\mathcal{O}+B) \end{array} \quad \text{(prim-Prop)}$$

Proof. Let ϕ and ψ be given as in the antecedent of the theorem. We instantiate direcursion with $A' = \mathcal{O} \times A$ and $B' = \mathcal{O} + B$ and define $\phi' : G(\mathcal{O}+B, \mathcal{O}\times A) \to \mathcal{O} \times A$ and $\psi' : \mathcal{O} + B \to G(\mathcal{O} \times A, \mathcal{O} + B)$ by $\phi' = \langle \iota \circ G(\mathsf{inl}, \pi_1), \phi\rangle$ and $\psi' = [G(\pi_1, \mathsf{inl}) \circ \iota^\circ, \psi]$. From these two maps, we define $g = \pi_2 \circ (\![\phi', \psi']\!)$ and $h = [\![\phi', \psi']\!] \circ \mathsf{inr}$. Finally, we verify that each square commutes (omitting the reasoning for the right square as the following dualises):

$$\begin{array}{lll} & g \circ \iota & \\ = & \pi_2 \circ (\![\phi', \psi']\!) \circ \iota & \text{by assumption} \\ = & \pi_2 \circ \langle \iota \circ G(\mathsf{inl}, \pi_1), \phi\rangle \circ G([\![\phi', \psi']\!], (\![\phi', \psi']\!)) & \text{by (direc-Self)} \\ = & \phi \circ G([\![\phi', \psi']\!], (\![\phi', \psi']\!)) & \text{by (co)pairing} \\ = & \phi \circ G([[\![\phi', \psi']\!] \circ \mathsf{inl}, [\![\phi', \psi']\!] \circ \mathsf{inr}], \langle \pi_1 \circ (\![\phi', \psi']\!), \pi_2 \circ (\![\phi', \psi']\!)\rangle) & \text{by surjective pairing} \\ = & \phi \circ G([[\![\iota, \iota^\circ]\!], h], \langle (\![\iota, \iota^\circ]\!), g\rangle) & \text{by (direc-Fusion)} \\ = & \phi \circ G([id, h], \langle id, g\rangle) & \text{by (direc-Refl)} \end{array}$$

□

The previous theorem in fact defines a unique pair of morphism:

Theorem 4 (Primitive Direcursion Characterisation). *Let* $(\mathcal{O}, \mathcal{O}, \iota_G, \iota_G^\circ)$ *be the initial* G*-dialgebra. Suppose*

$$\begin{array}{ll} (a) & \phi' = \langle \iota \circ G(inl, \pi_1), \phi\rangle, \text{ and} \\ (b) & \psi' = [G(\pi_1, inl) \circ \iota^\circ, \psi]. \end{array}$$

Then the following two statements are equivalent:

$$(i)\ g \circ \iota_G = \phi \circ G([id, h], \langle id, g\rangle)$$
$$\iota_G^\circ \circ h = G(\langle id, g\rangle, [id, h]) \circ \psi,\ \textit{and}$$
$$(ii)\ g = \pi_2 \circ (\!|\phi', \psi'|\!)\ \textit{and}\ h = [\![\phi', \psi']\!] \circ \mathit{inr}. \qquad \text{(prim-Charn)}$$

We introduce the notation $\langle\!|\phi, \psi|\!\rangle_G \stackrel{def}{=} g$ *and* $|\!\rangle\phi, \psi\langle\!|_G \stackrel{def}{=} h$ *for any given pair of (well-typed) maps* ϕ *and* ψ.

Proof. Straightforward. □

The following basic property follows directly:

Corollary 1. *Let* $(\mathcal{O}, \mathcal{O}, \iota_G, \iota_G^\circ)$ *be the initial* G*-dialgebra. Every function given by* G*-primitive direcursion can also be given by* G*-direcursion up to a certain pre/post-composed map:*

$$\begin{aligned}\langle\!|\phi, \psi|\!\rangle &= \pi_2 \circ (\!|\langle \iota \circ G(\mathit{inl}, \pi_1), \phi\rangle, [G(\pi_1, \mathit{inl}) \circ \iota^\circ, \psi]|\!) \\ |\!\rangle\phi, \psi\langle\!| &= [\![\langle \iota \circ G(\mathit{inl}, \pi_1), \phi\rangle, [G(\pi_1, \mathit{inl}) \circ \iota^\circ, \psi]]\!] \circ \mathit{inr}\end{aligned} \qquad \text{(prim-Direc)}$$

Proof. Straightforward. □

Note here that primitive direcursive functions can generally not be defined by purely direcursive functions, although the previous corollary establishes a close relationship between the two notions.

Lemma 5. *Let* $(\mathcal{O}, \mathcal{O}, \iota_G, \iota_G^\circ)$ *be the initial* G*-dialgebra. Primitive direcursion satisfies the following cancellation and reflection laws:*

$$\langle\!|\phi, \psi|\!\rangle \circ \iota_G = \phi \circ G([id, |\!\rangle\phi, \psi\langle\!|], \langle id, \langle\!|\phi, \psi|\!\rangle\rangle) \qquad \text{(prim-Self)}$$
$$\iota_G^\circ \circ |\!\rangle\phi, \psi\langle\!| = G(\langle id, \langle\!|\phi, \psi|\!\rangle\rangle, [id, |\!\rangle\phi, \psi\langle\!|]) \circ \psi$$

$$\forall\psi \quad id = \langle\!|\iota \circ G(\mathit{inl}, \pi_1), \psi|\!\rangle \qquad \text{(prim-Refl)}$$
$$\forall\phi \quad id = |\!\rangle\phi, G(\pi_1, \mathit{inl}) \circ \iota^\circ\langle\!|$$

Proof. The cancellation law follows immediately from (prim-Charn) by just chasing the diagram given in (prim-Prop). We show the first reflection law (the other is dual):

	$\langle\!|\iota \circ G(\mathsf{inl}, \pi_1), \psi|\!\rangle$	
$=$	$\langle\!|\iota \circ G(\mathsf{inl}, \pi_1), \psi|\!\rangle \circ \iota \circ \iota^\circ$	by regularity
$=$	$\iota \circ G(\mathsf{inl}, \pi_1) \circ G([id, |\!\rangle\phi, \psi\langle\!|], \langle id, \langle\!|\phi, \psi|\!\rangle\rangle) \circ \iota^\circ$	by (prim-Self)
$=$	$\iota \circ G([id, |\!\rangle\phi, \psi\langle\!|] \circ \mathsf{inl}, \pi_1 \circ \langle id, \langle\!|\phi, \psi|\!\rangle\rangle) \circ \iota^\circ$	by composition
$=$	$\iota \circ G(id, id) \circ \iota^\circ$	by (co)pairing
$=$	$\iota \circ \iota^\circ$	by functor property
$=$	id	by regularity

□

The previous lemma, albeit a direct consequence of the construction, is of importance since it gives a more efficient implementation of primitive direcursion in a lazy functional programming language. The final basic property, the fusion law, takes a particular form for primitive direcursion:

Corollary 2 (Primitive Direcursion Fusion). *Let $(\mathcal{O}, \mathcal{O}, \iota_G, \iota_G^\circ)$ be the initial G-dialgebra. Suppose that $(A, B, \phi : G(\mathcal{O}+B, \mathcal{O}\times A) \to A, \psi : B \to G(\mathcal{O}\times A, \mathcal{O}+B))$ and $(A', B', \phi' : G(\mathcal{O}+B', \mathcal{O}\times A') \to A', \psi' : B' \to G(\mathcal{O}\times A', \mathcal{O}+B'))$ are both G-dialgebras. For every dimap $g : A \to A', h : B' \to B$ such that*

$$g \circ \phi = \phi' \circ G(id + h, id \times g)$$

and

$$\psi \circ h = G(id \times g, id + h) \circ \psi'$$

we have the following property

$$g \circ \langle\!|\phi, \psi|\!\rangle = \langle\!|\phi', \psi'|\!\rangle \text{ and } \rangle\!|\phi, \psi|\!\langle \circ h = \rangle\!|\phi', \psi'|\!\langle \qquad \text{(prim-FUSION)}$$

We establish another relationship between direcursion and primitive direcursion, showing that primitive direcursion generalises direcursion in the following sense:

Lemma 6. *Let $(O, O, \iota, \iota^\circ)$ be the initial G-dialgebra. For any $\phi : G(B, A) \to A$ and $\psi : B \to G(A, B)$, the following equalities hold:*

$$\begin{array}{l} (\![\phi, \psi]\!) = \langle\!|\phi \circ G(\mathit{inr}, \pi_2), G(\pi_2, \mathit{inr}) \circ \psi|\!\rangle \\ [\!(\phi, \psi)\!] = \rangle\!|\phi \circ G(\mathit{inr}, \pi_2), G(\pi_2, \mathit{inr}) \circ \psi|\!\langle \end{array} \qquad \text{(direc-PRIM)}$$

That is, every direcursive function is also a primitive direcursive function.

Proof. Straightforward using (direc-FUSION). □

The (prim-SELF) law generalises into a statement that primitive direcursion is universal in the sense of e.g. Proposition 4.3 in [6]:

Lemma 7. *Let $(O, O, \iota, \iota^\circ)$ be the initial G-dialgebra. Any morphism $f : \mathcal{O} \to A$ satisfies the following identity, for any $\psi : B \to G(\mathcal{O}\times A, \mathcal{O}+B)$:*

$$f = \langle\!|f \circ \iota \circ G(\mathit{inl}, \pi_1), \psi|\!\rangle$$

Furthermore, any morphism $f' : B \to \mathcal{O}$ satisfies the following identity, for any $\phi : G(\mathcal{O}+B, A\times\mathcal{O}) \to A$:

$$f' = \rangle\!|\phi, G(\pi_1, \mathit{inl}) \circ \iota^\circ \circ f'|\!\langle$$

Proof. We first convince the reader that the equalities are well-typed:

$$\begin{array}{ccc} G(\mathcal{O}+B, \mathcal{O}\times A) & \longrightarrow & A \\ \Big\downarrow {\scriptstyle G(\mathsf{inl},\pi_1)} & \equiv & \Big\uparrow {\scriptstyle f} \\ G(\mathcal{O}, \mathcal{O}) & \xrightarrow{\ \iota\ } & \mathcal{O} \end{array} \qquad \begin{array}{ccc} B & \longrightarrow & G(\mathcal{O}\times A, \mathcal{O}+B) \\ {\scriptstyle f'} \Big\downarrow & \equiv & \Big\uparrow {\scriptstyle G(\pi_1,\mathsf{inl})} \\ \mathcal{O} & \xrightarrow{\ \iota^\circ\ } & G(\mathcal{O}, \mathcal{O}) \end{array}$$

Let $\phi_0 = f \circ \iota \circ G(\mathsf{inl}, \pi_1)$. For the first equality we reason as follows:

$$
\begin{array}{lll}
 & \llparenthesis f \circ \iota \circ G(\mathsf{inl}, \pi_1), \psi \rrparenthesis & \\
= & \iota \circ \iota^\circ \circ \llparenthesis f \circ \iota \circ G(\mathsf{inl}, \pi_1), \psi \rrparenthesis & \text{by regularity} \\
= & f \circ \iota \circ G(\mathsf{inl}, \pi_1) \circ G([id, \rrparenthesis \phi_0, \psi \llparenthesis], \langle id, \llparenthesis \phi_0, \psi \rrparenthesis \rangle) \circ \iota^\circ & \text{by (prim-Self)} \\
= & f \circ \iota \circ G([id, \rrparenthesis \phi_0, \psi \llparenthesis] \circ \mathsf{inl}, \pi_1 \circ \langle id, \llparenthesis \phi_0, \psi \rrparenthesis \rangle) \circ \iota^\circ & \text{by composition} \\
= & f \circ \iota \circ G(id, id) \circ \iota^\circ & \text{by (co)pairing} \\
= & f & \text{by regularity}
\end{array}
$$

Let $\psi_0 = G(\pi_1, \mathsf{inl}) \circ \iota^\circ \circ f'$. For the second equality dually reason as follows:

$$
\begin{array}{lll}
 & \rrparenthesis \phi, f \circ \iota \circ G(\mathsf{inl}, \pi_1) \llparenthesis & \\
= & \rrparenthesis \phi, f \circ \iota \circ G(\mathsf{inl}, \pi_1) \llparenthesis \circ \iota \circ \iota^\circ & \text{by regularity} \\
= & \iota \circ G(\langle id, \llparenthesis \phi, \psi_0 \rrparenthesis \rangle, [id, \rrparenthesis \phi, \psi_0 \llparenthesis]) \circ G(\pi_1, \mathsf{inl}) \circ \iota^\circ \circ f' & \text{by (prim-Self)} \\
= & \iota \circ G(\pi_1 \circ \langle id, \llparenthesis \phi, \psi_0 \rrparenthesis \rangle, [id, \rrparenthesis \phi, \psi_0 \llparenthesis] \circ \mathsf{inl}) \circ \iota^\circ \circ f' & \text{by composition} \\
= & \iota \circ G(id, id) \circ \iota^\circ \circ f' & \text{by (co)pairing} \\
= & f' & \text{by regularity}
\end{array}
$$

□

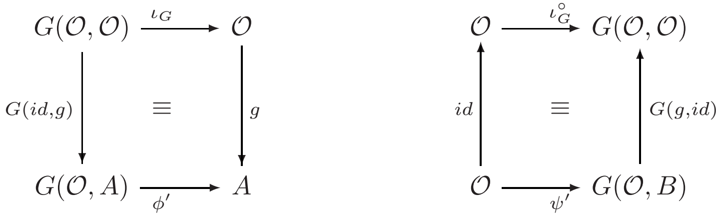
We now turn to our main result. Suppose g is an iterative map as follows:

$$
\begin{array}{ccc}
G(\mathcal{O}, \mathcal{O}) & \xrightarrow{\ \iota\ } & \mathcal{O} \\
{\scriptstyle G(id,g)} \downarrow & \equiv & \downarrow {\scriptstyle g} \\
G(\mathcal{O}, A) & \xrightarrow[\ \phi\ ]{} & A
\end{array}
$$

We ask: is g definable using direcursion (which here means that it belongs to a class of functions defined inductively, closed under composition, and including elementary functions such as (co)pairing, projections/injections, constants, id, as well as any function defined by direcursion with parameters ϕ, ψ in this class)? Can we, for example, choose A, B, ϕ', ψ' such that the following diagrams commute with this given g as a solution?

$$
\begin{array}{ccc}
G(\mathcal{O}, \mathcal{O}) & \xrightarrow{\ \iota_G\ } & \mathcal{O} \\
{\scriptstyle G(h,g)} \downarrow & \equiv & \downarrow {\scriptstyle g} \\
G(B, A) & \xrightarrow[\ \phi'\ ]{} & A
\end{array}
\qquad\qquad
\begin{array}{ccc}
\mathcal{O} & \xrightarrow{\ \iota_G^\circ\ } & G(\mathcal{O}, \mathcal{O}) \\
{\scriptstyle h} \uparrow & \equiv & \uparrow {\scriptstyle G(g,h)} \\
B & \xrightarrow[\ \psi'\ ]{} & G(A, B)
\end{array}
$$

To simulate iteration, we wish to force $h = id$ in this definition. In other words, we must specialise the above definition as follows:



We conclude that for $h = id$ we require the following condition:

$$G(g, id) \circ \psi' = \iota^\circ$$

If g has a right inverse (g^{-1}), then we can solve this equation:

$$\psi' = G(g^{-1}, id) \circ G(g, id) \circ \psi' = G(g^{-1}, id) \circ \iota^\circ$$

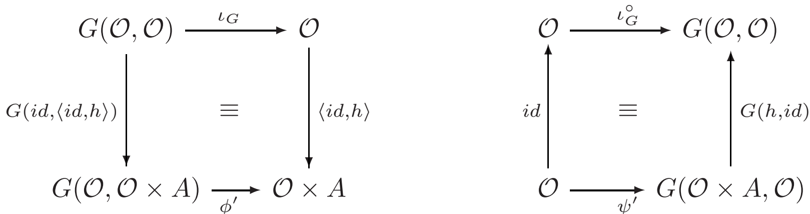
We have therefore arrived at a problem: a sufficient condition is that the iterative map g has a right inverse g^{-1}, but this does not hold in many cases, for example not for all those iterative maps that fail to be surjective. For instance, if A is a standard lazy list datatype (as a cpo) then we are forced to exclude maps without e.g. the empty lists (or infinite lists etc) in their image. We therefore would like to have a more generally applicable solution. Since we have merely identified a sufficient condition, we are not in a hopeless situation. It turns out that primitive direcursion provides a solution to this problem:

Theorem 5 (Iteration as direcursion). *Let G be a locally continuous functor $\mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ and suppose that $\mathcal{O}$ carries the initial G-dialgebra. Suppose that $h = (\![\phi]\!)_{G(\mathcal{O},_)}$. Then $h = \pi_2 \circ (\![\phi', \psi']\!)$ with $\phi' : G(\mathcal{O}, \mathcal{O} \times A) \to \mathcal{O} \times A$ and $\psi' : \mathcal{O} \to G(\mathcal{O} \times A, \mathcal{O})$ given by*

$$\phi' = (\iota \times \phi) \circ \langle G(id, \pi_1), G(id, \pi_2)\rangle, \text{ and} \tag{1}$$

$$\psi' = G(\pi_1, id) \circ \iota^\circ. \tag{2}$$

Proof. We begin by asking when the following diagrams will commute where we force $g = \langle id, h\rangle$.



For the right hand square we must find a canonical ψ' satisfying the following property:

$$G(\langle id, h\rangle, id) \circ \psi' = \iota^\circ$$

But we can take $\psi' = G(\pi_1, id) \circ \iota^\circ$ and show that it satisfies this property:

$$G(\langle id, h\rangle, id) \circ G(\pi_1, id) \circ \iota^\circ = G(\pi_1 \circ \langle id, h\rangle, id) \circ \iota^\circ = G(id, id) \circ \iota^\circ = \iota^\circ$$

It remains to show that we can always define also ϕ' such that the left hand square also commutes. To define ϕ' we use that h is $G(\mathcal{O}, _)$-iterative, i.e.

$$h \circ \iota = \phi \circ G(id, h).$$

From this property we will then prove that $\langle id, h\rangle \circ \iota = \phi' \circ G(id, \langle id, h\rangle)$, by defining a canonical ϕ' from ϕ as follows:

$$\phi' = (\iota \times \phi) \circ \langle G(id, \pi_1), G(id, \pi_2)\rangle$$

We can now establish the result for parametric iterative maps:

$$\begin{aligned}
\langle id, h\rangle \circ \iota &= (\iota \times \phi') \circ \langle G(id, \pi_1), G(id, \pi_2)\rangle \circ G(id, \langle id, h\rangle)) && (3)\\
&= (\iota \times \phi') \circ \langle G(id, \pi_1) \circ G(id, \langle id, h\rangle), G(id, \pi_2)\rangle \circ G(id, \langle id, h\rangle)) && (4)\\
&= (\iota \times \phi') \circ \langle G(id, id), G(id, h)\rangle && (5)\\
&= (\iota \times \phi') \circ \langle id, G(id, h)\rangle && (6)\\
&= \langle \iota, \phi' \circ G(id, h)\rangle && (7)\\
&= \langle \iota, h \circ \iota\rangle = \langle id, h\rangle \circ \iota && (8)
\end{aligned}$$

Note how we in (7) used that h is iterative. □

This result immediately dualises:

Theorem 6 (Coiteration as direcursion). *Let G be a locally continuous functor $\mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ and suppose that $\mathcal{O}$ carries the initial G-dialgebra. Suppose that $h = [\![\psi]\!]_{G(\mathcal{O},_)}$. Then $h = [\![\phi', \psi']\!] \circ \mathsf{inr}$ with $\phi' : G(\mathcal{O} + B, \mathcal{O}) \to \mathcal{O}$ and $\psi' : \mathcal{O} + B \to G(\mathcal{O}, \mathcal{O} + B)$ given by*

$$\begin{aligned}
\phi' &= \iota \circ G(\mathsf{inl}, id), \text{ and} && (9)\\
\psi' &= [G(id, \mathsf{inl}), G(id, \mathsf{inr})] \circ (\iota^\circ + \psi). && (10)
\end{aligned}$$

We consider the first theorem again. Can we eliminate also the postcomposed projection in the construction? For this we seek a dialgebra homomorphism $(\pi_2, id) : (\mathcal{O} \times A, \mathcal{O}, \phi'_\phi, \psi'_\phi) \to (A, \mathcal{O}, \phi'', \psi'')$ (in order to use fusion), i.e. we require (for ϕ' and ψ' as in the theorem):

$$\begin{array}{ccc}
G(\mathcal{O}, \mathcal{O} \times A) & \xrightarrow{\phi'} & \mathcal{O} \times A \\
\big\downarrow{\scriptstyle G(id,\pi_2)} & \equiv & \big\downarrow{\scriptstyle \pi_2} \\
G(\mathcal{O}, A) & \xrightarrow[\phi'']{} & A
\end{array}
\qquad
\begin{array}{ccc}
\mathcal{O} & \xrightarrow{\psi'} & G(\mathcal{O} \times A, \mathcal{O}) \\
\big\uparrow{\scriptstyle id} & \equiv & \big\uparrow{\scriptstyle G(\pi_2,id)} \\
\mathcal{O} & \xrightarrow[\psi'']{} & G(A, \mathcal{O})
\end{array}$$

That is, we wish to find another dialgebra (ϕ'', ψ'') such that the unique homomorphism from the initial G-dialgebra into that dialgebra factors through the maps given in this diagram. The right-hand diagram forces the following property (using the definition of ψ'):

$$G(\pi_2, id) \circ \psi'' = \psi' = G(\pi_1, id) \circ \iota_G^\circ$$

For the left-hand side to commute we require:

$$\phi'' \circ G(id, \pi_2) = \pi_2 \circ \phi' = \phi \circ G(id, \pi_2)$$

We conclude that in general it will not be possible to eliminate the postcomposed projection, and dually not the precomposed injection. However, a sufficient condition is that the iterative map g is split epic with right inverse g^{-1} as initially remarked above. We close the section by summarising the development:

Corollary 3 (Main Result). *Suppose $G : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ is a locally continuous bifunctor and that $(\mathcal{O}, \mathcal{O}, \iota, \iota^\circ)$ is the initial G-dialgebra. Then the following is true for arbitrary maps $\phi : G(\mathcal{O}, A) \to A$ and $\psi : B \to G(\mathcal{O}, B)$ and α, β:*

$$(\![\phi]\!)_{G(\mathcal{O},_)} = \langle\!| \phi \circ G(id, \pi_2), \alpha |\!\rangle_G, \text{ and} \tag{11}$$

$$[\!(\psi)\!]_{G(\mathcal{O},_)} = |\!\rangle \beta, G(id, \mathit{inr}) \circ \psi \langle\!|_G. \tag{12}$$

In particular, we can take α and β to be the unique maps that factors through the zero object for a suitably chosen object, e.g. zero $0 \cong 1$ itself:

$$\alpha = \bot_{1,G(\mathcal{O}\times A,\mathcal{O})} : 1 \to G(\mathcal{O} \times A, \mathcal{O})$$
$$\beta = \bot_{G(\mathcal{O}+B,\mathcal{O}),1} : G(\mathcal{O} + B, \mathcal{O}) \to 1$$

4 Example: Application to Object Calculus Semantics

Direcursion arises naturally in self-application semantics [18] of typed object calculus based on recursive types, which has recently been subject to some research [26,28,17,16]. In this section, we will consider the interpretation in Glimming et al [17], but for concreteness we work in $\mathsf{CPPO}_{\bot!}$ rather than a category of partial maps. For example, $\mathbb{B}_\bot$ is a flat cpo with underlying set $\{\mathtt{tt}, \mathtt{ff}, \bot\}$, and all maps are strict. We conclude the paper by giving some examples. Note that all of these examples (and others) have been implemented in a lazy functional programming language:

Example 1 (Object-Based Natural Numbers). Abadi and Cardelli [1] model object-based "natural numbers" essentially by defining a type $\sigma = Obj(X)[\text{pred} : X, \text{zero} : \text{Bool}]$, and then giving suitable terms to represent numbers (see loc. cit.). Under self-application semantics, this object type is modelled as a solution to the domain equation $\mathcal{O} \cong \mathcal{O} \multimap (\mathcal{O}_\bot \times \mathbb{B}_\bot)$. Writing $(\mathcal{O}, \mathcal{O}, \iota, \iota^\circ)$ for the initial dialgebra arising from the induced mixed variant functor G, we can define a

map $\mathsf{sapp} : \mathcal{O} \to (\mathcal{O}_\perp \times \mathbb{B}_\perp)$ by $\mathsf{sapp} = \mathsf{eval} \circ \langle \iota^\circ, id \rangle$. Now we have for example $\iota(\lambda x.(\perp, \perp)), \iota(\lambda x.(x, \perp)) \in \mathcal{O}$ and also $zero = \lceil 0 \rceil = \iota(\lambda x.(x, \mathtt{tt})) \in \mathcal{O}$. Moreover, define $\lceil n+1 \rceil = \iota(\lambda x.(\lceil n \rceil, \mathtt{ff})) \in \mathcal{O}$ for $n \in \mathbb{N}$. Note that in general the "methods" need not be constant functions, but can depend on the current "self" in a non-trivial way. Now the extraction of a natural number in $\mathbb{N}_\perp$ (given by a flat cpo) from an "object" in $\mathcal{O}$ can be defined as a $G(\mathcal{O}, _)$-iterative function as follows:

$$\begin{array}{ccc} G(\mathcal{O},\mathcal{O}) & \xrightarrow{\iota} & \mathcal{O} \\ {\scriptstyle G(id,h)}\downarrow & \equiv & \downarrow{\scriptstyle h} \\ G(\mathcal{O},\mathcal{O}\times\mathbb{N}_\perp) & \xrightarrow[\phi]{} & \mathcal{O}\times\mathbb{N}_\perp \end{array}$$

In the diagram, ϕ is the following algebra map for taking one step during the extraction:

$$\phi = f \circ \langle \pi_2, eval \rangle \circ \langle G(1, {\pi_1}_\perp \circ \pi_1), \iota \circ G(1, \langle {\pi_1}_\perp \circ \pi_1, \pi_2 \rangle) \rangle$$

where $f : \mathcal{O} \times (\mathbb{N}_\perp)_\perp \to \mathcal{O} \times \mathbb{N}_\perp$ is defined as follows (for usual strict addition $+$):

$$f(o,n) = \begin{cases} (o,0), & \text{if } \pi_2 \circ \mathsf{sapp}(o) = \mathtt{tt} \\ (\pi_1 \circ \mathsf{sapp}(o), m+1), & \text{if } n = m_\perp \\ (o, \perp), & \text{otherwise} \end{cases}$$

Note that the first case applies when the zero method evaluates to $\mathtt{tt}$, and that the second gives back the predecessor in the first component. It can now be inferred that parametric iterative maps can be useful for defining functions on objects, since more involved examples can be constructed similar to this simplified one for "natural numbers". We have shown in this paper that the map h can equivalently be defined as $h = \pi_1 \circ k$ where $k = (\!|\phi', \psi'|\!)_G$ for a suitable dialgebra (ϕ', ψ') as detailed in previous sections. The resulting definition is shown in the following diagrams:

$$\begin{array}{ccc} G(\mathcal{O},\mathcal{O}) & \xrightarrow{\iota} & \mathcal{O} \\ {\scriptstyle G(id,k)}\downarrow & \equiv & \downarrow{\scriptstyle k} \\ G(\mathcal{O},\mathcal{O}\times(\mathcal{O}\times\mathbb{N}_\perp)) & \xrightarrow[\phi']{} & \mathcal{O}\times(\mathcal{O}\times\mathbb{N}_\perp) \end{array} \qquad \begin{array}{ccc} \mathcal{O} & \xrightarrow{\iota^\circ} & G(\mathcal{O},\mathcal{O}) \\ {\scriptstyle g=id}\uparrow & \equiv & \uparrow{\scriptstyle G(k,id)} \\ \mathcal{O} & \xrightarrow[\psi']{} & G(\mathcal{O}\times(\mathcal{O}\times\mathbb{N}_\perp),\mathcal{O}) \end{array}$$

Note that we need two different $\mathcal{O}$ here, since we cannot invert sapp. Our main result states that we have $\phi' = (\iota \times \phi) \circ \langle G(id, \pi_1), G(id, \pi_2) \rangle$ and $\psi' = G(\pi_1, id) \circ \iota^\circ$. (We can alternatively and equivalently define $h = \langle\!| \phi \circ G(id, \pi_2), \perp |\!\rangle_G$ according to Corollary 3.)

Example 2 (Constructors). We next define a direcursive function δ which creates an "object" when given a natural number, i.e. a "constructor". In this case, we consider the type $\sigma' = Obj(X)[\mathrm{pred} : X, \mathrm{zero} : \mathrm{Bool}, \mathrm{succ} : X]$, which allows a method for computing the successor of the number stored in an object as well. A domain $\mathcal{O}'$ arises from the equation $\mathcal{O}' \cong \mathcal{O}' \multimap (\mathcal{O}'_\perp \times \mathbb{B}_\perp \times \mathcal{O}'_\perp)$. We dub the induced mixed variant functor H, and define δ first using full direcursion:

$$
\begin{array}{ccc}
H(\mathcal{O}',\mathcal{O}') & \xrightarrow{\iota_H} & \mathcal{O}' \\
{\scriptstyle H(\delta,\gamma)}\downarrow & \equiv & \downarrow{\scriptstyle \gamma} \\
H(\mathbb{N},1) & \xrightarrow[\perp]{} & 1
\end{array}
\qquad
\begin{array}{ccc}
\mathcal{O}' & \xrightarrow{\iota_H^\circ} & H(\mathcal{O}',\mathcal{O}') \\
{\scriptstyle \delta}\uparrow & \equiv & \uparrow{\scriptstyle H(\gamma,\delta)} \\
\mathbb{N} & \xrightarrow[\delta^-]{} & H(1,\mathbb{N})
\end{array}
$$

The unique solution arises by providing δ^- as follows:

$$\delta^-(n) = \lambda x.(n-1, n \equiv 0, n+1)$$

Note that the positive part of the diagram trivialises in this definition. In fact, there is a more natural parametric coiterative definition:

$$
\begin{array}{ccc}
\mathcal{O}' & \xrightarrow{\iota_H} & H(\mathcal{O}',\mathcal{O}') \\
{\scriptstyle \delta'}\uparrow & \equiv & \uparrow{\scriptstyle H(id,\delta')} \\
\mathcal{O}' \times \mathbb{N}_\perp & \xrightarrow[\delta'^-]{} & H(\mathcal{O}',\mathcal{O}' \times \mathbb{N}_\perp)
\end{array}
$$

Here, we require the following $H(\mathcal{O}, _)$-coalgebra:

$$\delta'^-(n) = \lambda o.((o, n-1), n \equiv 0, (o, n+1))$$

Using Corollary 3 we have $\delta' = [\!(\perp, H(id, \mathsf{inr}) \circ \delta'^-)\!]_H$. The general problem of proving that two maps such as δ' and δ compute the same "objects", requires suitable bisimulations (see example 5 and 6 below).

Example 3 (Subtyping). Using direcursion, we will define an embedding-projection (ep-) pair $(\alpha, \beta) : \mathcal{O} \to \mathcal{O}'$. Here β serves as a *coercion function* for subtyping, whereas α gives an *approximation function* from a type into the given supertype. The pair (α, β) is the unique solution to the following diagrams:

$$
\begin{array}{ccc}
G(\mathcal{O},\mathcal{O}) & \xrightarrow{\iota} & \mathcal{O} \\
{\scriptstyle G(\beta,\alpha)}\downarrow & \equiv & \downarrow{\scriptstyle \alpha} \\
G(\mathcal{O}',\mathcal{O}') & \xrightarrow[\alpha^+]{} & \mathcal{O}'
\end{array}
\qquad
\begin{array}{ccc}
\mathcal{O} & \xrightarrow{\iota^\circ} & G(\mathcal{O},\mathcal{O}) \\
{\scriptstyle \beta}\uparrow & \equiv & \uparrow{\scriptstyle G(\alpha,\beta)} \\
\mathcal{O}' & \xrightarrow[\beta^-]{} & G(\mathcal{O}',\mathcal{O}')
\end{array}
$$

In these diagrams, we have chosen α^+ and β^- as follows:

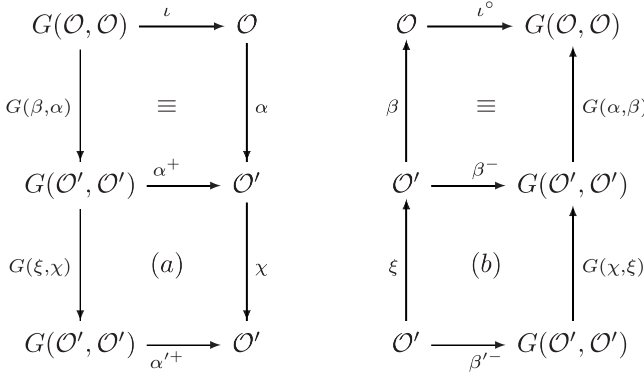
$$\alpha^+ = \iota \circ \langle \pi_1, \pi_2, \bot \rangle^{id}$$
$$\beta^- = \langle \pi_1, \pi_2 \rangle^{id} \circ \iota^\circ$$

Here $\bot : \mathcal{O}'_\bot \times \mathbb{B}_\bot \to \mathcal{O}'_\bot$ is the constantly undefined map. It is straightforward to show that $\beta^- \circ \alpha^+ = id$ and moreover that $\alpha^+ \circ \beta^- \sqsubseteq_{\mathcal{O}'} id$. (Recall that $\sqsubseteq_{\mathcal{O}'}$ is the coordinatewise order inherited from the infinitary product which determines $\mathcal{O}'$.) It follows that (α, β) is an ep-pair since G is locally continuous. Note that since subtyping uses full direcursion, we can use a constructor that works with both $\mathcal{O}'$ and $\mathcal{O}$, since the (prim-Fusion) rule can be used, once both maps are given in direcursive form using our main result.

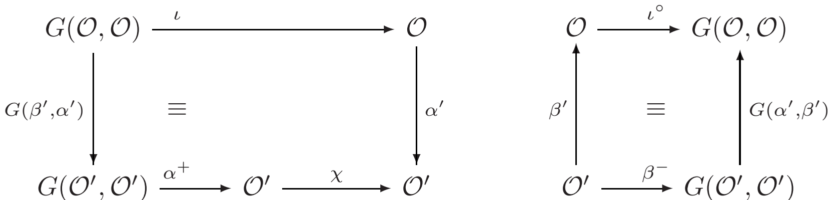
Example 4 (Inheritance). We continue with the previous example by also considering the following function:

$$\chi(o) = \iota(\lambda p.(\pi_1 \circ \iota^\circ(o)(p), \pi_2 \circ \iota^\circ(o)(p), \iota(\lambda q.(p, \mathtt{ff}, \iota^\circ(p)(q)))))$$

Note how χ serves to interpret the successor method from Abadi et al [1]. It takes an element in $\mathcal{O}'$ and equips it with the successor method. The definition involves both using method updates to copy the previous predecessor and zero state, but also the third component which adds "succ" such that it becomes constantly defined in the recursive structure. Now consider χ together with (α, β):



To unveil *inheritance* as a direcursive map, all we need to do is to find a dialgebra (α'^+, β'^-) and a map ξ making (χ, ξ) a dimap into this new G-dialgebra. Since this is a "standard" update (i.e. of the form Abadi and Cardelli considers in typed object calculi), it is constant in the recursive structure. This means that the fusion law gives us a unique dimap (α', β') for inheritance with $\xi = id$ and $\beta'^- = \beta^-$:



(This gives again an ep-pair.) We would like to use the parametric (co)iterative operations from example 1 and 2 also after we have inherited some of $\mathcal{O}$ into the new $\mathcal{O}'$. To use the fusion rule we must first apply Corollary 3 to the parametric coiterative map. We "rotate" the diagrams for the ep-pair and paste them to the definition of extraction provided in example 2, after which the (prim-Fusion) rule can readily be used once again, illustrating the usefulness of our results. However, if we attempt this for example 1 instead (without moving to H-dialgebras), we arrive at these seemingly awkward diagrams:

$$
\begin{array}{ccc}
G(\mathcal{O}',\mathcal{O}') & \xrightarrow{\alpha^+} & \mathcal{O}' \\
\downarrow{\scriptstyle G(\alpha,\beta)} & \equiv & \downarrow{\scriptstyle \beta} \\
G(\mathcal{O},\mathcal{O}) & \xrightarrow{\iota} & \mathcal{O} \\
\downarrow{\scriptstyle G(id,k)} & \equiv & \downarrow{\scriptstyle k} \\
G(\mathcal{O},\mathcal{O}\times(\mathcal{O}\times\mathbb{N}_\perp)) & \xrightarrow[\phi']{} & \mathcal{O}\times(\mathcal{O}\times\mathbb{N}_\perp)
\end{array}
\qquad
\begin{array}{ccc}
\mathcal{O}' & \xrightarrow{\beta^-} & G(\mathcal{O}',\mathcal{O}') \\
\uparrow{\scriptstyle \alpha} & \equiv & \uparrow{\scriptstyle G(\beta,\alpha)} \\
\mathcal{O} & \xrightarrow{\iota^\circ} & G(\mathcal{O},\mathcal{O}) \\
\uparrow{\scriptstyle g} & \equiv & \uparrow{\scriptstyle G(k,id)} \\
\mathcal{O} & \xrightarrow[\psi']{} & G(\mathcal{O}\times(\mathcal{O}\times\mathbb{N}_\perp),\mathcal{O})
\end{array}
$$

The diagrams that arise generalise a so-called "hylomorphism" (well-studied in purely functional program algebra, see e.g. [22,20]), i.e. that of a coiterative map followed by a iterative map. Here, both maps are instead direcursive, but the diagrams are combined such that the negative part of direcursion is followed by the positive part of another direcursive map. Ongoing work investigates if e.g. the so-called shift law [20] generalises to this setting using dinatural transformations.

Example 5 (Generalised Bisimulations). Tews [32] in his PhD thesis based a notion of higher-order bisimulation on a certain kind of morphism between what he termed generalised coalgebras. These maps, together with an associated notion of bisimulation, were in his thesis demonstrated as useful in a number of practical examples. However, Tews found that almost all important closure properties failed to hold in the category of sets unless strong restrictions were imposed. Via Corollary 3, we have now linked Tews' work to Freyd's direcursion principle. To see this, let h be a generalised coalgebra map in Tews' sense, i.e. let the following diagram commute:

$$
\begin{array}{ccccc}
A & \xrightarrow{\alpha} & G(A,A) & & \\
 & & & \searrow{\scriptstyle G(id_A,h)} & \\
\downarrow{\scriptstyle h} & & & & G(A,B) \\
 & & & \nearrow{\scriptstyle G(h,id_B)} & \\
B & \xrightarrow[\beta]{} & G(B,B) & &
\end{array}
$$

One readily identifies h as a parameterised $G(A, _)$-coalgebra map from (A, α) to $(B, G(h, id_B) \circ \beta)$ so our results apply to these maps. Existence of final generalised coalgebras appears to be an open problem [32]. As Tews' remarks, a coiteration scheme (as well as a proof principle) of potential practical utility [32] would follow, if an affirmative answer is found. The results presented in this paper do not close this problem but rather reopen it, and set the stage for further investigations. Moreover, the ambient category that we have used circumvents Tews' counterexample.

Example 6 (Self-Applicative Bisimulations). Note that sapp can be made to carry a $G(\mathcal{O}, _)$-coalgebra. This is achieved by defining $\mathcal{O} \to G(\mathcal{O}, \mathcal{O})$ by $K \circ \mathsf{sapp}$ where K is the usual combinator (we abuse notation slightly and call this map sapp as well). Hence there is a *parametric coiterative* function $[\![\mathsf{sapp}]\!]_{G(\mathcal{O},_)}$ from $\mathcal{O}$ into the final $G(\mathcal{O}, _)$-coalgebra. The kernel $\sim_{\mathsf{sapp}}$ of this coalgebra map is a bisimulation equivalence [27]. Thus $\mathcal{O}/\sim_{\mathsf{sapp}}$ is a well-defined quotient (on the underlying set). This bisimulation collapses object-based programs into class-based programs. The results of the present paper allow us to view this map as a dialgebra homomorphism $\langle\!|\mathsf{sapp} \circ G(id, \pi_2), \perp|\!\rangle_G$ (by Corollary 3), so that it can be combined with the above examples (or more complex future ones). Note that $\sim_{\mathsf{sapp}}$, although giving the link to class-based languages, is not the most suitable bisimulation for identifying object-based programs (and a type-indexed set of similar domains $\mathcal{O}_\sigma$ do not give a fully abstract model of object calculus, since Viswanathan's counterexample [35] can readily be applied). For this we require instead a finer notion, which we term OBP-bisimulation. It requires a structure $\beta : \mathcal{O} \to G(\mathcal{O} \times \mathcal{L}^+, \mathcal{O})$, which combines both mup and sapp behaviour, where $\mathsf{mup} : \mathcal{O} \times \mathcal{L} \times \mathcal{O} \to \mathcal{O}$ copies given method $\ell \in \mathcal{L}$ from the first argument to the third. Here $\beta = \mathsf{sapp} \circ \mathsf{mup}^+$ where mup^+ is a generalisation of method update which copies all the labels listed in $\mathcal{L}^+$ before the update. ($\mathcal{L}^+$ is a finite sequence of labels to be updated.) Note that β is not a coalgebra map. We therefore regard β as the negative part of a dialgebra map, i.e. require full direcursion. This outlines our current direction for future work. We emphasise that our main result already shows how $\sim_{\mathsf{sapp}}$ (and any other bisimulation arising in this way) can be combined with Freyd's direcursion so that a previous gap has been closed.

5 Conclusions and Further Work

In the 1990s, Freyd demonstrated in two well-known papers [12,13] a principle for defining functions on recursive types. However, he did not discuss how parametric (co)iterative maps can be written using his scheme. We have shown here that all such maps *can* in fact easily be defined, after we found and developed a certain variation of Freyd's principle that we termed primitive direcursion (generalising primitive recursion for endofunctors). Moreover, we have established some elementary algebraic properties of primitive direcursion. Taken together, our result can be viewed as a set of corollaries of Bekič's Lemma, giving a link between direcursion and parameterised datatypes via a recursion scheme.

A consequence of our result is that we demonstrate that many direcursive maps (previously known as difficult to exemplify [21]) arise by translating (more common) (co)iterative maps into Freyd's principle. As an additional consequence, functions defined on the parameterised initial algebra (and dually final coalgebra) can be combined (as per the usual "fusion laws") with more involved functions (e.g. the interpreters from [21]) which require full direcursion. In such situations, we can now use the reasoning principles of direcursion, even if the (co)iterative function is not surjective (not injective), and we in such cases avoid having to provide an "inverse-like" function which was previously identified as a problem [8]. We have given some examples from the semantics of object calculi, which demonstrates a situation where "fusion laws" can be used, showing that our results could be practically useful. Current work aims to take this further and establish an algebra of object-based programs in the spirit of Bird et al [5].

The author is presently developing bisimulations in the setting of denotational models of object calculus, in the vein of the work of Abramsky [2], with a goal to combine these with "fusion laws". Indeed the developments of Fiore [10] show that applicative bisimulations can be internalised into $\mathcal{C}$. Notably, such bisimulations use merely the parameterised final coalgebra since the contravariant argument remains fixed.

Another topic for further investigation is the expressivity of (primitive) direcursion. One would like to know if it is possible to naturally capture known classes of functions given by circular definitions (in the spirit of e.g. [6], but see also [15,7]). The author is particularly interested in characterising the total direcursive maps.

Finally, taking the two kinds of maps given in Corollary 3 together (with two suitable instantiations), we have the following (as in one of our examples):

$$\pi_2 \circ \langle\!| \phi, G(id, \pi_2), \alpha |\!\rangle \circ |\!\rangle \beta, G(id, \mathsf{inr}) \circ \psi \langle\!| \circ \mathsf{inl} : A \to B$$

For future work, we ask: when can direcursive maps be split into a pair of parametric (co)iterative maps of this form essentially computing results in two separate stages (like e.g. *fix* or quicksort)? Note that we can allow two different (di)naturally related mixed-variant functors, and choose A, B as well as an intermediate $\mathcal{O}$ into which all partial results can be embedded. We hope that such an analysis could further our understanding of direcursion.

Acknowledgements

I wish to express my gratitude in particular to John Power, Bernhard Reus, and to Viggo Stoltenberg-Hansen, for encouragement and helpful discussions on the research reported here, and to the anonymous referees for their comments on an earlier version of this paper.

References

1. Abadi, M., Cardelli, L.: A Theory of Objects. Springer, Heidelberg (1996)
2. Abramsky, S.: The lazy lambda calculus. In: Turner, D.A. (ed.) Research Topics in Functional Programming, pp. 65–116. Addison-Welsey, Reading, MA (1990)
3. Abramsky, S., Jung, A.: Domain theory. In: Abramsky, S., Gabbay, D., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science, vol. 3, pp. 1–168. Oxford University Press, Oxford (1994)
4. Bekič, H.: Definable operation in general algebras, and the theory of automata and flowcharts. In: Programming Languages and Their Definition - Hans Bekic (1936-1982) pp. 30–55, London, UK, Springer, Heidelberg (1984)
5. Bird, R.S., de Moor, O.: Algebra of Programming. Prentice-Hall, Englewood Cliffs (1997)
6. Cancila, D., Honsell, F., Lenisa, M.: Generalized coiteration schemata. Electronical Notes in Computer Science, 82(1) (2003)
7. Danielsson, N.A., Hughes, J., Jansson, P., Gibbons, J.: Fast and loose reasoning is morally correct. In: POPL '06: Symposium on Principles of Programming Languages, pp. 206–217. ACM Press, New York (2006)
8. Fegaras, L., Sheard, T.: Revisiting catamorphisms over datatypes with embedded functions (or, programs from outer space). In: Conf. Record 23rd ACM SIGPLAN/SIGACT Symp. on Principles of Programming Languages, POPL'96, St. Petersburg Beach, FL, USA, January 21-24, 1996, pp. 284–294. ACM Press, New York (1996)
9. Fiore, M.P.: Axiomatic Domain Theory in Categories of Partial Maps. Distinguished Dissertations in Computer Science. Cambridge University Press, Cambridge (1996)
10. Fiore, M.P.: A coinduction principle for recursive data types based on bisimulation. Information and Computation 127(2), 186–198 (1996)
11. Fokkinga, M.M.: Law and order in algorithmics. Ph.D. thesis, Technical University Twente, The Netherlands (1992)
12. Freyd, P.J.: Recursive types reduced to inductive types. In: Proceedings 5th IEEE Annual Symp. on Logic in Computer Science, LICS'90, pp. 498–507. IEEE Computer Society Press, Los Alamitos (1990)
13. Freyd, P.J.: Algebraically complete categories. In: Proceedings 1990 Como Category Theory Conference. Lecture Notes in Mathematics, vol. 1488, pp. 95–104 (1991)
14. Freyd, P.J.: Remarks on algebraically compact categories. In: Fourman, Johnstone, Pitts (eds.) Workshop on Applications of Categories in Computer Science, Proceedings of the London Mathematical Society Symposium. London Mathematical Society Lecture Note Series, vol. 177, pp. 95–106. Cambridge University Press, Cambridge (1992)
15. Gibbons, J., Hutton, G., Altenkirch, T.: When is a function a fold or an unfold? In: Proceedings of the 4th International Workshop on Coalgebraic Methods in Computer Science. Electronic Notes in Theoretical Computer Science, vol. 44 (2001)
16. Glimming, J.: Dialgebraic semantics of typed object calculi. Licentiate Thesis (May 2005) TRITA-NA-0511
17. Glimming, J., Ghani, N.: Difunctorial semantics of object calculus. In: Proceedings of the Second Workshop on Object Oriented Developments (WOOD 2004). Electronic Notes in Theoretical Computer Science, vol. 138, Elsevier, Amsterdam (2004)

18. Kamin, S.N.: Inheritance in Smalltalk-80: a denotational definition. In: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 80–87. ACM Press, New York (1988)
19. Meertens, L.: Paramorphisms. Formal Aspects of Computing 4(5), 413–424 (1992)
20. Meijer, E., Fokkinga, M., Paterson, R.: Functional programming with bananas, lenses, envelopes and barbed wire. In: Hughes, J. (ed.) Functional Programming Languages and Computer Architecture. LNCS, vol. 523, pp. 124–144. Springer, Heidelberg (1991)
21. Meijer, E., Hutton, G.: Bananas in space: extending fold and unfold to exponential types. In: Peyton-Jones, S. (ed.) Functional Programming Languages and Computer Architecture, pp. 324–333. Association for Computing Machinery (1995)
22. Paterson, R.: Operators. In: Lecture Notes for the International Summerschool on Constructive Algorithmics, Ameland, Netherlands. CWI Amsterdam, Utrecht University, University of Nijmegen (September 1990)
23. Pitts, A.M.: A co-induction principle for recursively defined domains. Theoretical Computer Science 124, 195–219 (1994)
24. Pitts, A.M.: Relational properties of domains. Information and Computation 127(2), 66–90 (1996)
25. Plotkin, G.D.: Post-graduate lecture notes in advanced domain theory (incorporating the “Pisa Notes”). Dept. of Computer Science, Univ. of Edinburgh (1981)
26. Reus, B., Streicher, T.: Semantics and logic of object calculi. Theorertical Computer Science 316(1), 191–213 (2004)
27. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249(1), 3–80 (2000)
28. Schwinghammer, J.: Reasoning about Denotations of Recursive Objects. Ph.D. thesis, University of Sussex (2005)
29. Scott, D.: Continuous lattices. In: Lawvere, F.W. (ed.) Toposes, Algebraic Geometry, and Logic. Lecture Notes in Mathematics, vol. 274, pp. 97–136. Springer, Heidelberg (1972)
30. Smyth, M., Plotkin, G.: The category-theoretic solution of recursive domain equations. SIAM Journal of Computing 11(4), 761–783 (1982)
31. Stoltenberg-Hansen, V., Lindström, I., Griffor, E.R.: Mathematical Theory of Domains. Cambridge University Press, Cambridge (1994)
32. Tews, H.: Coalgebraic Methods for Object-Oriented Specification. PhD thesis, Technical University of Dresden (2002)
33. Uustalu, T., Vene, V.: Primitive (co)recursion and course-of-value (co)iteration, categorically. Informatica 10(1), 5–26 (1999)
34. Vene, V.: Categorical programming with inductive and coinductive types. PhD thesis, University of Tartu (2000)
35. Viswanathan, R.: Full abstraction for first-order objects with recursive types and subtyping. In: Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS 1998), IEEE Computer Society Press, Los Alamitos (1998)
36. Washburn, G., Weirich, S.: Boxes go bananas: Encoding higher-order abstract syntax with parametric polymorphism. In: Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming, pp. 249–262. ACM Press, New York (2003)

Bisimulation for Neighbourhood Structures

Helle Hvid Hansen[1,2,⋆], Clemens Kupke[1,⋆⋆], and Eric Pacuit[3,⋆⋆⋆]

[1] Centrum voor Wiskunde en Informatica (CWI)
[2] Vrije Universiteit Amsterdam (VUA)
[3] Universiteit van Amsterdam (UvA)

Abstract. Neighbourhood structures are the standard semantic tool used to reason about non-normal modal logics. In coalgebraic terms, a neighbourhood frame is a coalgebra for the contravariant powerset functor composed with itself, denoted by $\mathsf{2}^{\mathsf{2}}$. In our paper, we investigate the coalgebraic equivalence notions of $\mathsf{2}^{\mathsf{2}}$-bisimulation, behavioural equivalence and neighbourhood bisimulation (a notion based on pushouts), with the aim of finding the logically correct notion of equivalence on neighbourhood structures. Our results include relational characterisations for $\mathsf{2}^{\mathsf{2}}$-bisimulation and neighbourhood bisimulation, and an analogue of Van Benthem's characterisation theorem for all three equivalence notions. We also show that behavioural equivalence gives rise to a Hennessy-Milner theorem, and that this is not the case for the other two equivalence notions.

Keywords: Neighbourhood semantics, non-normal modal logic, bisimulation, behavioural equivalence, invariance.

1 Introduction

Neighbourhood semantics (cf. [7]) forms a generalisation of Kripke semantics, and it has become the standard tool for reasoning about non-normal modal logics in which (Kripke valid) principles such as $\Box p \wedge \Box q \;\rightarrow\; \Box(p \wedge q)$ and $\Box p \;\rightarrow\; \Box(p \vee q)$ (mon) are considered not to hold. In a neighbourhood model, each state has associated with it a collection of subsets of the universe (called neighbourhoods), and a modal formula $\Box\phi$ is true at states which have the truth set of ϕ as a neighbourhood. The modal logic of all neighbourhood models is called classical modal logic.

During the past 15-20 years, non-normal modal logics have emerged in the areas of computer science and social choice theory, where system (or agent) properties are formalised in terms of various notions of ability in strategic games (e.g. [2, 19]). These logics have in common that they are monotonic, meaning they contain the above formula (mon). The corresponding property of neighbourhood models is that neighbourhood collections are closed under supersets. Non-monotonic modal logics occur in deontic logic (see e.g. [8]) where monotonicity

⋆ Supported by NWO grant 612.000.316.
⋆⋆ Supported by NWO under FOCUS/BRICKS grant 642.000.502.
⋆⋆⋆ Supported by NSF grant OISE 0502312.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 279–293, 2007.
© Springer-Verlag Berlin Heidelberg 2007

can lead to paradoxical obligations, and in the modelling of knowledge and related epistemic notions (cf. [23, 16]). Furthermore, the topological semantics of modal logic can be seen as neighbourhood semantics (see [22] and references).

In the present paper we try to find the "logically correct" notion of semantic equivalence in neighbourhood structures. For monotonic neighbourhood structures, this question has already been addressed (cf. [18, 11]), but as mentioned in [18], it is not immediate how to generalise monotonic bisimulation to arbitrary neighbourhood structures. This is where coalgebra comes in. Neighbourhood frames are easily seen to be coalgebras for the contravariant powerset functor composed with itself, denoted 2^2. Based on this observation the general theory of coalgebra (cf. [21, 14]) provides us with a number of candidates: behavioural equivalence, 2^2-bisimulation and a third notion (based on pushouts), which we refer to as neighbourhood bisimulation. From the logic point of view, a good equivalence notion E should have the following properties: (rel) E is characterised by relational (back-and-forth) conditions which can be effectively checked for finite models; (hm) the class of finite neighbourhood models is a Hennessy-Milner class with respect to E; and (chr) classical modal logic is the E-invariant fragment of first-order logic interpreted over neighbourhood models, i.e., we would like an analogue of Van Benthem's characterisation theorem ([3]) to hold. These logic-driven criteria form the main points of our investigation.

In section 2 we define basic notions and notation. In section 3 we investigate the three equivalence notions, first for arbitrary set functors, and then for 2^2-coalgebras. We provide relational characterisations for 2^2-bisimulation and neighbourhood bisimulation, and we show, by means of examples, that in general neighbourhood bisimilarity is stronger than behavioural equivalence, and weaker than 2^2-bisimilarity. However, when considered on a single model, the three notions coincide. The above-mentioned examples also demonstrate that 2^2-bisimilarity and neighbourhood bisimilarity fail to satisfy (hm). Furthermore, in much work on coalgebra (cf. [21]) it is often assumed that the functor preserves weak pullbacks, however, it is not always clear whether this requirement is really needed. In [9], weaker functor requirements for congruences are studied, and 2^2 provides an example of a functor which does not preserve weak pullbacks in general, but only the ones consisting of kernel pairs. Finally, in section 4 we prove the analogue of the Van Benthem characterisation theorem, for all three equivalences. To this end, we introduce a notion of modal saturation for neighbourhood models, and since we can show that in a class of modally saturated models, modal equivalence implies behavioural equivalence, it follows that behavioural equivalence has the property (hm). *All omitted proof details can be found in a forthcoming technical report ([12]).* So although behavioural equivalence fails at the property (rel), we still consider it the mathematically optimal equivalence notion. Taking computational aspects into consideration, we find that neighbourhood bisimulations provide a good approximation of behavioural equivalence, while still allowing a fairly simple relational characterisation. 2^2-bisimulations, however, must be considered too strict a notion.

2 Preliminaries and Notation

In this section, we settle on notation, define the necessary coalgebraic notions, and introduce neighbourhood semantics for modal logic. For further reading on coalgebra we refer to [21, 24]. Extended discussions on neighbourhood semantics can be found in [7, 10].

Let X and Y be sets, and $R \subseteq X \times Y$ a relation. For $U \subseteq X$ and $V \subseteq Y$, we denote the R-image of U by $R[U] = \{y \in Y \mid \exists x \in U : xRy\}$, and the R-preimage of V by $R^{-1}[V] = \{x \in X \mid \exists y \in V : xRy\}$. The domain of R is $\mathsf{dom}(R) = R^{-1}[Y]$, and the range of R is $\mathsf{rng}(R) = R[X]$. Note that in the special case that R is (the graph of) a function, then image, preimage, domain and range amount to the usual definitions. Given a set X, we denote by $\mathcal{P}(X)$ the powerset of X, and for a subset $Y \subseteq X$, we write Y^c for the complement $X \setminus Y$ of Y in X.

Let $\mathsf{At} = \{p_j \mid j \in \omega\}$ be a fixed, countable set of atomic sentences. The basic modal language $\mathcal{L}(\mathsf{At})$ is defined by the grammar: $\phi ::= p_j \mid \neg\phi \mid \phi \wedge \phi \mid \Box\phi$, where $j \in \omega$. To ease notation, we write $\mathcal{L}$ instead of $\mathcal{L}(\mathsf{At})$. Formulas of $\mathcal{L}$ are interpreted in neighbourhood models.

Definition 1. *A* neighbourhood frame *is a tuple $\langle S, \nu \rangle$ where S is a nonempty set and $\nu : S \to \mathcal{P}(\mathcal{P}(S))$ is a neighbourhood function which assigns to each state $s \in S$ a collection of neighbourhoods. A* neighbourhood model *based on a neighbourhood frame $\langle S, \nu \rangle$ is a tuple $\langle S, \nu, V \rangle$ where $V : \mathsf{At} \to \mathcal{P}(S)$ is a valuation function.*

Let $\mathcal{M} = \langle S, \nu, V \rangle$ be a neighbourhood model and $s \in S$. Truth of the atomic propositions is defined via the valuation: $\mathcal{M}, s \models p_i$ iff $s \in V(p_i)$, and inductively over the boolean connectives as usual. For the modal operator, we write $\mathcal{M}, s \models \Box\phi$ iff $(\phi)^{\mathcal{M}} \in \nu(s)$, where $(\phi)^{\mathcal{M}} = \{t \in S \mid \mathcal{M}, t \models \phi\}$ denotes the *truth set of ϕ in $\mathcal{M}$*. Let also $\mathcal{N}$ be a neighbourhood model. Two states, s in $\mathcal{M}$ and t in $\mathcal{N}$, are *modally equivalent (notation: $s \equiv t$)* if they satisfy the same modal formulas, i.e., $s \equiv t$ if and only if for all $\phi \in \mathcal{L}$: $\mathcal{M}, s \models \phi$ iff $\mathcal{N}, t \models \phi$. A subset $X \subseteq S$ is *modally coherent* if for all $s, t \in S$: $s \equiv t$ implies $s \in X$ iff $t \in X$.

The maps between neighbourhood models which preserve the modal structure are referred to as bounded morphisms.

Definition 2. *If $\mathcal{M}_1 = \langle S_1, \nu_1, V_1 \rangle$ and $\mathcal{M}_2 = \langle S_2, \nu_2, V_2 \rangle$ are neighbourhood models, and $f : S_1 \to S_2$ is a function, then f is a* (frame) bounded morphism from $\langle S_1, \nu_1 \rangle$ to $\langle S_2, \nu_2 \rangle$ *(notation: $f : \langle S_1, \nu_1 \rangle \to \langle S_2, \nu_2 \rangle$), if for all $X \subseteq S_2$, we have $f^{-1}[X] \in \nu_1(s)$ iff $X \in \nu_2(f(s))$; and f is a* bounded morphism from $\mathcal{M}_1$ to $\mathcal{M}_2$ *(notation: $f : \mathcal{M}_1 \to \mathcal{M}_2$) if $f : \langle S_1, \nu_1 \rangle \to \langle S_2, \nu_2 \rangle$ and for all $p_j \in \mathsf{At}$, and all $s \in S_1$: $s \in V_1(p_j)$ iff $f(s) \in V_2(p_j)$.*

As usual, bounded morphisms preserve truth of modal formulas. That is, if $f : \mathcal{M}_1 \to \mathcal{M}_2$, then for all $\mathcal{L}$-formulas ϕ, and all states s in $\mathcal{M}_1$: $\mathcal{M}_1, s \models \phi$ iff $\mathcal{M}_2, f(s) \models \phi$. This can be proved by a straightforward induction on the formula structure (left to the reader).

We will work in the category Set of sets and functions. Let $\mathrm{F} : \mathsf{Set} \to \mathsf{Set}$ be a functor. Recall that an F-*coalgebra* is a pair $\langle S, \sigma \rangle$ where S is a set, and $\sigma : S \to \mathrm{F}(S)$ is a function, sometimes called the *coalgebra map*. Given two F-coalgebras, $\langle S_1, \sigma_1 \rangle$ and $\langle S_2, \sigma_2 \rangle$, a function $f : S_1 \to S_2$ is a *coalgebra morphism* if $\mathrm{F}(f) \circ \sigma_1 = \sigma_2 \circ f$.

The contravariant powerset functor $\mathsf{2} : \mathsf{Set} \to \mathsf{Set}$ maps a set X to $\mathcal{P}(X)$, and a function $f : X \to Y$ to the inverse image function $f^{-1} : \mathcal{P}(Y) \to \mathcal{P}(X)$. The functor $\mathsf{2}^\mathsf{2}$ is defined as the composition of $\mathsf{2}$ with itself. It should be clear that neighbourhood frames are $\mathsf{2}^\mathsf{2}$-coalgebras and vice versa, although we follow standard logic practice and exclude the empty coalgebra from being a neighbourhood structure. Similarly, a neighbourhood model $\langle S, \nu, V \rangle$ corresponds with a coalgebra map $\langle \nu, V' \rangle : S \to \mathsf{2}^\mathsf{2}(S) \times \mathcal{P}(\mathsf{At})$ for the functor $\mathsf{2}^\mathsf{2}(-) \times \mathcal{P}(\mathsf{At})$ by viewing the valuation $V : \mathsf{At} \to \mathcal{P}(S)$ as a map $V' : S \to \mathcal{P}(\mathsf{At})$ where $p_i \in V'(s)$ iff $s \in V(p_i)$. Moreover, it is straightforward to show a function $f : S_1 \to S_2$ is a bounded morphism between the neighbourhood frames $\mathcal{S}_1 = \langle S_1, \nu_1 \rangle$ and $\mathcal{S}_2 = \langle S_2, \nu_2 \rangle$ iff f is a coalgebra morphism from $\mathcal{S}_1$ to $\mathcal{S}_2$. Similarly, $\mathsf{2}^\mathsf{2}(-) \times \mathcal{P}(\mathsf{At})$-coalgebra morphisms are simply the same as bounded morphisms between neighbourhood models. In what follows we will switch freely between the coalgebraic setting and the neighbourhood setting.

Finally, we will need a number of technical constructions. The disjoint union of two sets S_1 and S_2 is denoted by $S_1 + S_2$. Disjoint unions of neighbourhood frame/models are instances of the category theoretical notion of *coproducts*, and they lift disjoint unions of sets to neighbourhood frames/models such that the inclusion maps are bounded morphisms. This amounts to the following definition for neighbourhood models; the definition for neighbourhood frames is obtained by leaving out the part about the valuations.

Definition 3. *Let $\mathcal{M}_1 = \langle S_1, \nu_1, V_1 \rangle$ and $\mathcal{M}_2 = \langle S_2, \nu_2, V_2 \rangle$ be two neighbourhood models. The* disjoint union of $\mathcal{M}_1$ and $\mathcal{M}_2$ *is the neighbourhood model $\mathcal{M}_1 + \mathcal{M}_2 = \langle S_1 + S_2, \nu, V \rangle$ where for all $p_j \in \mathsf{At}$, $V(p_j) = V_1(p_j) \cup V_2(p_j)$; and for $i = 1, 2$, for all $X \subseteq S_1 + S_2$, and $s \in S_i$, $X \in \nu(s)$ iff $X \cap S_i \in \nu_i(s)$.*

In the sequel we will also use pullbacks and pushouts. We now remind the reader of how these can be constructed in Set. For the general definition we refer to any standard textbook on category theory (e.g. [1]).

First, given a relation $R \subseteq S_1 \times S_2$, we can view R as a relation on $S_1 + S_2$. We denote by $\hat{R}$ the smallest equivalence relation on $S_1 + S_2$ that contains R, and $(S_1 + S_2)/\hat{R}$ is the set of $\hat{R}$-equivalence classes.

Definition 4. *Let $f_1 : S_1 \to Z$ and $f_2 : S_2 \to Z$ be functions. The* canonical pullback of f_1 and f_2 *(in Set) is the triple $(\mathrm{pb}(f_1, f_2), \pi_1, \pi_2)$, where $\mathrm{pb}(f_1, f_2) := \{(s_1, s_2) \in S_1 \times S_2 \mid f_1(s_1) = f_2(s_2)\}$; and $\pi_1 : \mathrm{pb}(f_1, f_2) \to S_1$ and $\pi_2 : \mathrm{pb}(f_1, f_2) \to S_2$ are the projections.*

Let $R \subseteq S_1 \times S_2$ be a relation with projections $\pi_1 : R \to S_1$ and $\pi_2 : R \to S_2$. The canonical pushout of R *(in Set) is the triple $(\mathrm{po}(\pi_1, \pi_2), p_1, p_2)$, where $\mathrm{po}(\pi_1, \pi_2) := \big(S_1 + S_2\big)/\hat{R}$, and $p_1 : S_1 \to \mathrm{po}(\pi_1, \pi_2)$ and $p_2 : S_2 \to \mathrm{po}(\pi_1, \pi_2)$ are the obvious quotient maps.*

3 Equivalence Notions

In this section we will study various notions of "observational equivalence" for neighbourhood frames in detail. In the first part we list the three coalgebraic equivalence notions that we are going to consider. In the second part we spell out in detail what these three equivalence notions mean on neighbourhood frames.

3.1 Three Coalgebraic Notions of Equivalence

Remark 1. In this subsection we introduce behavioural and relational equivalences. We want to stress that we use the word "equivalence" to indicate that a relation relates only equivalent points. We do not require these equivalences to be equivalence relations.

The main observation for defining equivalences between coalgebras is that coalgebra morphisms preserve the behaviour of coalgebra states. This basic idea motivates the well-known coalgebraic definitions of bisimilarity and behavioural equivalence. In the following F denotes an arbitrary Set functor.

Definition 5. *Let $\mathcal{S}_1 = \langle S_1, \nu_1 \rangle$, $\mathcal{S}_2 = \langle S_2, \nu_2 \rangle$ be* F*-coalgebras. A relation $R \subseteq S_1 \times S_2$ is an (*F*-)bisimulation between $\mathcal{S}_1$ and $\mathcal{S}_2$ if there exists a function $\mu : R \to \mathrm{F}(R)$ such that for both $i = 1, 2$ the projection map $\pi_i : R \to S_i$ is a coalgebra morphism from $\langle R, \mu \rangle$ to $\mathcal{S}_i$. Two states s_1 and s_2 are* (F-)bisimilar *if they are linked by some bisimulation (notation: $s_1 \leftrightarroweq s_2$). We call $R \subseteq S_1 \times S_2$ a* behavioural equivalence *between $\mathcal{S}_1$ and $\mathcal{S}_2$ if there are a* F*-coalgebra $\langle Z, \lambda \rangle$ and* F*-coalgebra morphisms $f_i : \langle S_i, \nu_i \rangle \to \langle Z, \lambda \rangle$ for $i = 1, 2$ such that $R = \mathrm{pb}(f_1, f_2)$. Two states s_1 and s_2 that are related by some behavioural equivalence are called* behaviourally equivalent *(notation: $s_1 \leftrightarroweq^b s_2$).*

It has been proven in [21] that two states are F-bisimilar iff they are behaviourally equivalent under the assumption that the functor F is weak pullback preserving. The same article, however, tells us that the functor 2^2 that we want to study lacks this property. Therefore it makes sense to look at both 2^2-bisimulations and behavioural equivalences on our quest for the right notion of equivalence. In fact, we will also look at a third notion that, to the best of our knowledge, has not been considered before, namely the notion of a *relational equivalence*. The motivation for introducing relational equivalences is to remedy one obvious shortcoming of behavioural equivalences: in general it is difficult to provide some criterion for a relation R to be a behavioural equivalence. Bisimulations, in contrast, can be nicely characterized using relation lifting (cf. e.g. [20]). For example when considering Kripke frames ($\mathcal{P}$-coalgebras) this characterization yields the well-known forth and back conditions for Kripke bisimulations. We want to have a similar characterization of behavioural equivalence - even if the functor does not preserve weak pullbacks.

Definition 6. *Let $\mathcal{S}_1 = \langle S_1, \nu_1 \rangle$ and $\mathcal{S}_2 = \langle S_2, \nu_2 \rangle$ be* F*-coalgebras. Furthermore let $R \subseteq S_1 \times S_2$ be a relation and let $\langle Z, p_1, p_2 \rangle$ be the canonical pushout of R (cf. Def. 4). Then R is called a* relational equivalence *between $\mathcal{S}_1$ and $\mathcal{S}_2$ if there*

exists a coalgebra map $\lambda : Z \to \mathrm{F}(Z)$ *such that the functions* p_1 *and* p_2 *become coalgebra morphisms from* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *to* $\langle Z, \lambda \rangle$ *(see the diagram below). If two states* s_1 *and* s_2 *are related by some relational equivalence we write* $s_1 \leftrightarroweq^r s_2$*.*

$$\begin{array}{ccccc} & & R & & \\ & \swarrow^{\pi_1} & & \searrow^{\pi_2} & \\ S_1 & \xrightarrow{p_1} & Z & \xleftarrow{p_2} & S_2 \\ \downarrow \nu_1 & & \downarrow \exists \lambda & & \downarrow \nu_2 \\ \mathrm{F}(S_1) & \xrightarrow{\mathrm{F}(p_1)} & \mathrm{F}(Z) & \xleftarrow{\mathrm{F}(p_2)} & \mathrm{F}(S_2) \end{array}$$

We note that the definition of a relational equivalence is independent of the concrete representation of the pushout. This follows easily from the fact that pushouts are unique up-to isomorphism.

Remark 2. The main advantage of relational equivalences is that they can be characterized by some form of relation lifting [1]: Let $\langle S_1, \nu_1 \rangle$ and $\langle S_2, \nu_2 \rangle$ be F-coalgebras, let $R \subseteq S_1 \times S_2$ with projections π_1, π_2 and let $(\mathrm{po}(\pi_1, \pi_2), p_1, p_2)$ the canonical pushout of R. We define the F-lifting $\hat{\mathrm{F}}$ of R, by $\hat{\mathrm{F}}(R) := \mathrm{pb}(\mathrm{F}p_1, \mathrm{F}p_2) \subseteq \mathrm{F}(S_1) \times \mathrm{F}(S_2)$. It is not difficult to see that R is a relational equivalence iff for all $(s_1, s_2) \in R$ we have $(\nu_1(s_1), \nu_2(s_2)) \in \hat{\mathrm{F}}(R)$. If F preserves weak pullbacks one can show that $\hat{F}(R) = \overline{F}(R)$ where $\overline{F}$ denotes the well-known lifting of F to the category Rel of sets and relations (for the Definition of $\overline{\mathrm{F}}$ consult e.g. [20]).

Definition 6 ensures that relational equivalences only relate behavioural equivalent points. The following proposition provides a first comparison between the three equivalence notions. We leave the easy, but instructive proof to the reader.

Proposition 1. *Let* $\mathcal{S}_1 = \langle S_1, \nu_1 \rangle$ *and* $\mathcal{S}_2 = \langle S_2, \nu_2 \rangle$ *be* F*-coalgebras. We have for all* $s_1 \in S_1$ *and* $s_2 \in S_2$*:* $s_1 \leftrightarroweq s_2$ *implies* $s_1 \leftrightarroweq^r s_2$ *implies* $s_1 \leftrightarroweq^b s_2$*.*

This proposition is clearly not enough to justify the introduction of relational equivalences: our motivation was to give a characterization of behavioural equivalence using a relation lifting. We will demonstrate, however, that behavioural equivalences give us in general a strictly weaker notion of equivalence between coalgebras than relational equivalences. Luckily both notions coincide if we restrict our attention to "full" relations. In particular, we obtain the result that behavioural equivalence and relational equivalence amount to the same thing when studied on a single coalgebra.

Lemma 1. *If* $\mathcal{S}_1 = \langle S_1, \nu_1 \rangle$ *and* $\mathcal{S}_2 = \langle S_2, \nu_2 \rangle$ *are* F*-coalgebras and* $R \subseteq S_1 \times S_2$ *is a behavioural equivalence between* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *that is* full*, i.e.* $\mathsf{dom}(R) = S_1$ *and* $\mathsf{rng}(R) = S_2$*, then* R *is a relational equivalence.*

Proof. Let R be a behavioural equivalence with projection maps $\pi_1 : R \to S_1$ and $\pi_2 : R \to S_2$. Then there are some F-coalgebra $\langle Z, \lambda \rangle$ and coalgebra morphisms

[1] The definition of $\hat{F}$ goes back to an idea by Kurz ([13]) for defining a relation lifting of non weak pullback preserving functors.

$f_i : S_i \to Z$ for $i = 1, 2$ such that $R = \mathrm{pb}(f_1, f_2)$. Let $\langle Z', p_1, p_2\rangle$ be the canonical pushout of R. We are going to define a function $\lambda' : Z' \to \mathrm{F}(Z')$ such that p_i is a coalgebra morphism from $\mathcal{S}_i$ to $\langle Z', \lambda'\rangle$ for $i = 1, 2$.

By the universal property of the pushout there has to be a function $j : Z' \to Z$ such that $j \circ p_i = f_i$ for $i = 1, 2$. We claim that this function is injective. First it follows from the definition of the canonical pushout that both p_1 and p_2 are surjective, because R is a full relation. Let now $z_1, z_2 \in Z'$ and suppose that $j(z_1) = j(z_2)$. The surjectivity of the p_i's implies that there are $s_1 \in S_1$ and $s_2 \in S_2$ such that $p_1(s_1) = z_1$ and $p_2(s_2) = z_2$. Hence $j(p_1(s_1)) = j(p_2(s_2))$ which in turn yields $f_1(s_1) = f_2(s_2)$. This implies $(s_1, s_2) \in R$ and as a consequence we get $p_1(s_1) = p_2(s_2)$, i.e., $z_1 = z_2$. This demonstrates that j is injective and thus there is some surjective map $e : Z \to Z'$ with $e \circ j = \mathrm{id}_{Z'}$. Now put $\lambda' := \mathrm{F}(e) \circ \lambda \circ j$. It is straightforward to check that for $i = 1, 2$ the function p_i is a coalgebra morphism from $\mathcal{S}_i$ to $\langle Z', \lambda'\rangle$ as required.

Theorem 1. *Let $\mathcal{S} = \langle S, \nu\rangle$ be an* F*-coalgebra. Every behavioural equivalence $R \subseteq S \times S$ on $\mathcal{S}$ is contained in a relational equivalence. Hence $s \underline{\leftrightarrow}^b s'$ iff $s \underline{\leftrightarrow}^r s'$ for all $s, s' \in S$.*

Proof. The theorem is a consequence of Lemma 1 and the fact that every behavioural equivalence R on a coalgebra $\langle S, \nu\rangle$ is contained in a full one: If $R = \mathrm{pb}(f_1, f_2)$ for two coalgebra morphisms f_1 and f_2 we construct the coequalizer h of f_1 and f_2 in the category of F-coalgebras (cf. e.g. [21, Sec. 4.2]). If we put $f := h \circ f_1$ we obtain $R \subseteq R' := \mathrm{pb}(f, f)$, and R' is obviously full.

3.2 Equivalences Between Neighbourhood Frames

In this subsection we instantiate the three coalgebraic equivalence notions for 2^2-coalgebras, i.e., for neighbourhood frames.

We first consider 2^2-bisimulations. Recall from Def. 5 that a relation $R \subseteq S_1 \times S_2$ is a 2^2-bisimulation between two 2^2-coalgebras $\mathcal{S}_1 = \langle S_1, \nu_1\rangle$ and $\mathcal{S}_2 = \langle S_2, \nu_2\rangle$ if the projection maps π_1 and π_2 are bounded morphisms (2^2-coalgebra morphisms) from some 2^2-coalgebra (R, μ) to $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively. By Definition 2 of a bounded morphism this means that for $(s_1, s_2) \in R$ and $i = 1, 2$:

$$U \in \nu_i(s_i) \quad \text{iff} \quad \pi_i^{-1}[U] \in \mu(s_1, s_2) \qquad \text{for } U \subseteq S_i.$$

This leads to two "minimal requirements" on the neighbourhood functions ν_1 and ν_2 for pairs (s_1, s_2) related by a 2^2-bisimulation. For all $U_i, U_i' \subseteq S_i$, $i = 1, 2$:

1. $\pi_i^{-1}[U_i] = \pi_i^{-1}[U_i']$ implies $U_i \in \nu_i(s_i)$ iff $U_i' \in \nu_i(s_i)$,
2. $\pi_1^{-1}[U_1] = \pi_2^{-1}[U_2]$ implies $U_1 \in \nu_1(s_1)$ iff $U_1' \in \nu_2(s_2)$.

The following definition will help us to state these requirements in a concise way.

Definition 7. *Let $R \subseteq S_1 \times S_2$ be a relation with projection maps $\pi_i : R \to S_i$ for $i = 1, 2$. A set $U \subseteq S_1$ is called R-*unrelated *if $U \cap \mathsf{dom}(R) = \emptyset$. Similarly we call $V \subseteq S_2$ R-*unrelated *if $V \cap \mathsf{rng}(R) = \emptyset$. Furthermore we say two sets $U \subseteq S_1$ and $V \subseteq S_2$ are R-*coherent *if $\pi_1^{-1}[U] = \pi_2^{-1}[V]$.*

It is easy to check that for sets $U, U' \subseteq S_i$ we have $\pi_i^{-1}[U] = \pi_i^{-1}[U']$ iff the symmetric difference $U \Delta U'$ of U and U' is R-unrelated, i.e., iff U and U' only differ in points that do not occur in the relation R. The notion of R-coherency can also be formulated in terms of the relation R: Let $R \subseteq S_1 \times S_2$ be a relation and let $U \subseteq S_1$, $V \subseteq S_2$. Then U and V are R-coherent iff $R[U] \subseteq V$ and $R^{-1}[V] \subseteq U$.

Using the notions of R-coherency and R-unrelatedness we can reformulate the previous requirements and prove that they in fact characterize 2^2-bisimulations.

Proposition 2. *Let $\mathcal{S}_1 = \langle S_1, \nu_1 \rangle$ and $\mathcal{S}_2 = \langle S_2, \nu_2 \rangle$ be neighbourhood frames. A relation $R \subseteq S_1 \times S_2$ is a* 2^2-bisimulation *between $\mathcal{S}_1$ and $\mathcal{S}_2$ iff for all $(s_1, s_2) \in R$, for all $U_1, U_1' \subseteq S_1$ and for all $U_2, U_2' \subseteq S_2$ the following two conditions are satisfied:*

1. *$U_i \Delta U_i'$ is R-unrelated implies $U_i \in \nu_i(s_i)$ iff $U_i' \in \nu_i(s_i)$, for $i = 1, 2$.*
2. *U_1 and U_2 are R-coherent implies $U_1 \in \nu_1(s_1)$ iff $U_2 \in \nu_2(s_2)$.*

We will now demonstrate with an example that 2^2-bisimulations are too restrictive, i.e., we give an example of two states that *should be* regarded as equivalent but which are not 2^2-bisimilar.

Example 1. Let $T := \{t_1, t_2, t_3\}$ and $S := \{s\}$. Furthermore put $\nu_1(t_1) = \nu_1(t_2) := \{\{t_2\}\}$, $\nu_1(t_3) := \{\emptyset\}$ and $\nu_2(s) := \emptyset$ (cf. Fig. 1). We claim that there is no 2^2-bisimulation between $\langle T, \nu_1 \rangle$ and $\langle S, \nu_2 \rangle$ which relates t_1 and s.

We first note that t_3 and s cannot be related by a 2^2-bisimulation. This follows easily from the fact that $\emptyset \subseteq T$ and $\emptyset \subseteq S$ are R-coherent, and $\emptyset \in \nu_1(t_3)$ and $\emptyset \notin \nu_2(s)$. Suppose now R is a 2^2-bisimulation such that $(t_1, s) \in R$. It must then be the case that $\{t_3\} = \{t_3, t_2\} \Delta \{t_2\}$ is R-unrelated as we saw above. Therefore it follows by condition 1 of Proposition 2 that $\{t_3, t_2\} \in \nu_1(t_1)$ - a contradiction.

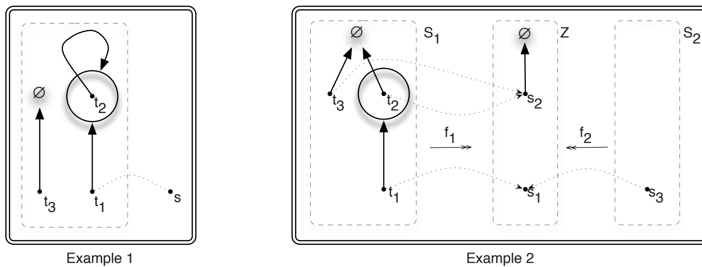


Fig. 1. Examples

But what justifies our claim that t_1 and s *should be* bisimilar? The reason is that t_1 and s are modally equivalent: in order to see this one has first to observe that the states t_1 and t_2 are obviously modally equivalent since they have the same neighbourhoods. Therefore $\{t_2\}$, the only neighbourhood set of t_1, is *undefinable*, i.e. every formula that is true at t_2 will be also true at t_1.

The semantics of the $\Box$-operator, however, only takes *definable* neighbourhoods into account, i.e. those neighbourhoods which consist exactly of those states that make a certain modal formula true. Hence it is possible to prove the modal equivalence of t_1 and s using an easy induction on the structure of a formula.

So let us have a look at our second candidate for an equivalence notion between 2^2-coalgebras, namely what we called *relational equivalence.* In the sequel the relational equivalences between neighbourhood frames will be referred to as *neighbourhood bisimulations.* The following proposition gives a characterization of neighbourhood bisimulations in set-theoretic terms.

Proposition 3. *Let $\mathcal{S}_i = \langle S_i, \nu_i \rangle$, $i = 1, 2$, be neighbourhood frames. A relation $R \subseteq S_1 \times S_2$ is a* neighbourhood bisimulation *iff for all $(s_1, s_2) \in R$ and for all R-coherent sets $U_1 \subseteq S_1$ and $U_2 \subseteq S_2$:*

The good news about neighbourhood bisimuations is that they capture the fact that the states t_1 and s in Example 1 are equivalent: The reader is invited to check that in this case $R := \{(t_1, s), (t_2, s)\}$ is a neighbourhood bisimulation. The next question is: how are neighbourhood bisimulations and behavioural equivalences related? Unfortunately the following example shows that neighbourhood bisimilarity is strictly stronger than behavioural equivalence.

Example 2. We are going to describe the situation that is depicted on the right in Figure 1. Let $S_1 := \{t_1, t_2, t_3\}$, $S_2 := \{s_3\}$ and define the neighbourhood functions ν_1 and ν_2 as follows: $\nu_1(t_1) := \{\{t_2\}\}$, $\nu_1(t_2) = \nu_1(t_3) := \{\emptyset\}$ and $\nu_2(s_3) := \emptyset$. We claim that the relation $R := \{(t_1, s_3)\}$ is a behavioural equivalence. Let $Z := \{s_1, s_2\}$, $\lambda(s_1) := \emptyset$ and $\lambda(s_2) := \{\emptyset\}$. Furthermore for $i \in \{1, 2\}$ we define functions $f_i : S_i \to Z$ by putting $f_1(t_1) := s_1$, $f_1(t_2) = f_1(t_3) := s_2$ and $f_2(s_3) := s_1$. Then it is straightforward to check that f_1 and f_2 are in fact bounded morphisms and that $R = \mathrm{pb}(f_1, f_2)$ as required.

At first this might look a bit surprising, because the neighbourhood frames (S_1, ν_1) and (S_2, ν_2) look rather different. But again it is not difficult to see that the states t_1 and s_3 should be considered equivalent because they are modally equivalent. Like in Example 1 the modal equivalence of t_1 and s_3 follows from the fact that the set $\{t_2\}$, the only neighbourhood of t_1, is not definable: all formulas that are true at t_2 are also true at t_3.

However t_1 and s_3 are not neighbourhood bisimilar: suppose for a contradiction that $(t_1, s_3) \in R'$ for some relational equivalence R'. Then it is easy to see that also $(t_2, s_3) \in R'$ (otherwise we obtain a contradicition from the fact that $\{t_2\}$ and $\emptyset$ are R-coherent). But $\emptyset \in \nu_1(t_2)$ now would imply $\emptyset \in \nu_2(s_3)$ because $\emptyset$ and $\emptyset$ are R'-coherent - a contradiction.

To sum it up: Example 1 showed that neighbourhood bisimulations are a clear improvement when compared to 2^2-bisimulations. Example 2, however, demonstrates that neighbourhood bisimulations are in general not able to capture behavioural equivalence of neighbourhood frames. If we consider behavioural equivalences on one neighbourhood frame all equivalence notions coincide.

Proposition 4. *Let $\mathcal{S} = (S, \nu)$ be a neighbourhood frame and $s_1, s_2 \in S$. Then $s_1 \leftrightarrows s_2$ iff $s_1 \leftrightarrows^r s_2$ iff $s_1 \leftrightarrows^b s_2$.*

Proof. The second equivalence is an instance of the more general result in Theorem 1. The first equivalence is a consequence of Prop. 2 and Prop. 3. Alternatively, the proposition can be proven using the result in [9] that congruence relations are bisimulations in case the functor weakly preserves kernel pairs - a property that the functor 2^2 has.

Remark 3. The results of this section can be easily extended from neighbourhood frames to neighbourhood *models*: a relation R is a (neighbourhood) bisimulation/behavioural equivalence between neighbourhood models, if R is a (neighbourhood) bisimulation/behavioural equivalence between the underlying neighbourhood frames which relates only points that satisfy the same propositions.

4 The Classical Modal Fragment of First-Order Logic

We will now prove that the three equivalence notions described in section 3 all characterise the modal fragment of first-order logic over the class of neighbourhood models (Theorem 2). This result is an analogue of Van Benthem's characterisation theorem for normal modal logic (cf. [3]): *On the class of Kripke models, modal logic is the (Kripke) bisimulation-invariant fragment of first-order logic*. The content of Van Benthem's theorem is that the basic modal language (with $\Box$) can be seen as a fragment of a first-order language which has a binary predicate $\mathsf{R}_\Box$, and a unary predicate P for each atomic proposition p in the modal language. Formulas of this first-order language can be interpreted in Kripke models in the obvious way. Van Benthem's theorem tells us that a first-order formula $\alpha(x)$ is invariant under Kripke bisimulation if and only if $\alpha(x)$ is equivalent to a modal formula.

4.1 Translation into First-Order Logic

The first step towards a Van Benthem-style characterisation theorem for classical modal logic is to show that $\mathcal{L}$ can be viewed as a fragment of first-order logic. It will be convenient to work with a *two-sorted* first-order language. Formally, there are two sorts $\{\mathsf{s}, \mathsf{n}\}$. Terms of the first sort (s) are intended to represent states, whereas terms of the second sort (n) are intended to represent neighbourhoods. We assume there are countable sets of variables of each sort. To simplify notation we use the following conventions: $x, y, x', y', x_1, y_2, \ldots$ denote variables of sort s (*state variables*) and $u, v, u', v', u_1, v_1, \ldots$ denote variables of sort n (*neighbourhood variables*). The language is built from a signature containing a unary predicate P_i (of sort s) for each $i \in \omega$, a binary relation symbol N relating elements of sort s to elements of sort n, and a binary relation symbol E relating elements of sort n to elements of sort s. The intended interpretation of $x\mathsf{N}u$ is "u is a neighbourhood of x", and the intended interpretation of $u\mathsf{E}x$ is "x is an element of u". The language $\mathcal{L}_1$ is built from the following grammar:

$$\phi \quad ::= \quad x = y \mid u = v \mid \mathsf{P}_i x \mid x\mathsf{N}u \mid u\mathsf{E}x \mid \neg\phi \mid \phi \wedge \psi \mid \exists x \phi \mid \exists u \phi$$

where $i \in \omega$; x and y are state variables; and u and v are neighbourhood variables. The usual abbreviations (eg. $\forall$ for $\neg\exists\neg$) apply.

Formulas of $\mathcal{L}_1$ are interpreted in two-sorted first-order structures $\mathfrak{M} = \langle D, \{P_i \mid i \in \omega\}, N, E\rangle$ where $D = D^{\mathsf{s}} \cup D^{\mathsf{n}}$ (and $D^{\mathsf{s}} \cap D^{\mathsf{n}} = \emptyset$), each $P_i \subseteq D^{\mathsf{s}}$, $N \subseteq D^{\mathsf{s}} \times D^{\mathsf{n}}$ and $E \subseteq D^{\mathsf{n}} \times D^{\mathsf{s}}$. The usual definitions of free and bound variables apply. Truth of sentences (formulas with no free variables) $\phi \in \mathcal{L}_1$ in a structure $\mathfrak{M}$ (denoted $\mathfrak{M} \models \phi$) is defined as expected. If x is a free state variable in ϕ (denoted $\phi(x)$), then we write $\mathfrak{M} \models \phi[s]$ to mean that ϕ is true in $\mathfrak{M}$ when $s \in D^{\mathsf{s}}$ is assigned to x. Note that $\mathfrak{M} \models \exists x\phi$ iff there is an element $s \in D^{\mathsf{s}}$ such that $\mathfrak{M} \models \phi[s]$. If Ψ is a set of $\mathcal{L}_1$-formulas, and $\mathfrak{M}$ is an $\mathcal{L}_1$-model, then $\mathfrak{M} \models \Psi$ means that for all $\psi \in \Psi$, $\mathfrak{M} \models \psi$. Given a class $\mathbf{K}$ of $\mathcal{L}_1$-models, we denote the *semantic consequence relation over* $\mathbf{K}$ by $\models_{\mathbf{K}}$. That is, for a set of $\mathcal{L}_1$-formulas $\Psi \cup \{\phi\}$, we have $\Psi \models_{\mathbf{K}} \phi$, if for all $\mathfrak{M} \in \mathbf{K}$, $\mathfrak{M} \models \Psi$ implies $\mathfrak{M} \models \phi$.

We can translate modal formulas of $\mathcal{L}$ and neighbourhood models to the first-order setting in a natural way:

Definition 8. *Let $\mathcal{M} = \langle S, \nu, V\rangle$ be a neighbourhood model. The* first-order translation *of $\mathcal{M}$ is the structure $\mathcal{M}^\circ = \langle D, \{P_i \mid i \in \omega\}, R_\nu, R_\ni\rangle$ where*

- $D^{\mathsf{s}} = S$, $D^{\mathsf{n}} = \nu[S] = \bigcup_{s\in S} \nu(s)$
- $P_i = V(p_i)$ *for each* $i \in \omega$,
- $R_\nu = \{(s, U) \mid s \in D^{\mathsf{s}}, U \in \nu(s)\}$,
- $R_\ni = \{(U, s) \mid s \in D^{\mathsf{s}}, s \in U\}$.

Definition 9. *The* standard translation *of the basic modal language is a family of functions $st_x : \mathcal{L} \to \mathcal{L}_1$ defined as follows: $st_x(p_i) = \mathsf{P}_i x$, $st_x(\neg\phi) = \neg st_x(\phi)$, $st_x(\phi \wedge \psi) = st_x(\phi) \wedge st_x(\psi)$, and*

$$st_x(\Box\phi) = \exists u(x\mathsf{N}u \wedge (\forall y(u\mathsf{E}y \leftrightarrow st_y(\phi))).$$

Standard translations preserve truth; the easy proof is left to the reader.

Lemma 2. *Let $\mathcal{M}$ be a neighbourhood model and $\phi \in \mathcal{L}$. For each $s \in S$, $\mathcal{M}, s \models \phi$ iff $\mathcal{M}^\circ \models st_x(\phi)[s]$.*

In the Kripke case, every first-order model for the language with $\mathsf{R}_\Box$ can be seen as Kripke model. However, it is not the case that every $\mathcal{L}_1$-structure is the translation of a neighbourhood model. Luckily, we can axiomatize the subclass of neighbourhood models up to isomorphism. Let $\mathbf{N} = \{\mathfrak{M} \mid \mathfrak{M} \cong \mathcal{M}^\circ$ for some neighbourhood model $\mathcal{M}\}$, and let NAX be the following axioms

(A1) $\exists x(x = x)$
(A2) $\forall u\exists x(x\mathsf{N}u)$
(A3) $\forall u, v(\neg(u = v) \to \exists x((u\mathsf{E}x \wedge \neg v\mathsf{E}x) \vee (\neg u\mathsf{E}x \wedge v\mathsf{E}x)))$

It is not hard to see that if $\mathcal{M}$ is a neighbourhood model, then $\mathcal{M}^\circ \models$ NAX. The next result states that, in fact, NAX completely characterizes the class $\mathbf{N}$.

Proposition 5. *Suppose $\mathfrak{M}$ is an $\mathcal{L}_1$-model and $\mathfrak{M} \models$ NAX. Then there is a neighbourhood model $\mathfrak{M}_\circ$ such that $\mathfrak{M} \cong (\mathfrak{M}_\circ)^\circ$.*

Thus, in a precise way, we can think of models in $\mathbf{N}$ as neighbourhood models. In particular, if $\mathfrak{M}$ and $\mathfrak{N}$ are in $\mathbf{N}$ we will write $\mathfrak{M} + \mathfrak{N}$ by which we (strictly speaking) mean the $\mathcal{L}_1$-model $(\mathfrak{M}_\circ + \mathfrak{N}_\circ)^\circ$ (which is also in $\mathbf{N}$).

Furthermore, Proposition 5 implies that we can work relative to $\mathbf{N}$ while still preserving nice first-order properties such as compactness and the existence of countably saturated models. These properties are essential in the proof of Theorem 2.

4.2 Characterisation Theorem

We are now able to formulate our characterisation theorem. Let $\sim$ be a relation on model-state pairs. Over the class $\mathbf{N}$, an $\mathcal{L}_1$-formula $\alpha(x)$ is *invariant under* $\sim$, if for all models $\mathfrak{M}_1$ and $\mathfrak{M}_2$ in $\mathbf{N}$ and all sort s-domain elements s_1 and s_2 of $\mathfrak{M}_1$ and $\mathfrak{M}_2$, respectively, we have $\mathfrak{M}_1, s_1 \sim \mathfrak{M}_2, s_2$ implies $\mathfrak{M}_1 \models \alpha[s_1]$ iff $\mathfrak{M}_2 \models \alpha[s_2]$. Over the class $\mathbf{N}$, an $\mathcal{L}_1$-formula $\alpha(x)$ *is equivalent to the translation of a modal formula* if there is a modal formula $\phi \in \mathcal{L}$ such that for all models $\mathfrak{M}$ in $\mathbf{N}$, and all s-domain elements s in $\mathfrak{M}$, $\mathfrak{M} \models \alpha[s]$ iff $\mathfrak{M} \models st_x(\phi)[s]$.

Theorem 2. *Let $\alpha(x)$ be an $\mathcal{L}_1$-formula. Over the class $\mathbf{N}$ (of neighbourhood models) the following are equivalent:*

1. *$\alpha(x)$ is equivalent to the translation of a modal formula,*
2. *$\alpha(x)$ is invariant under behavioural equivalence,*
3. *$\alpha(x)$ is invariant under neighbourhood bisimilarity,*
4. *$\alpha(x)$ is invariant under $\mathsf{2}^{\mathsf{2}}$-bisimilarity.*

Our proof of Theorem 2 uses essentially the same ingredients as the proof of Van Benthem's theorem (see e.g. [5]). In particular, we define a notion of modal saturation which ensures that modal equivalence implies behavioural equivalence. To this end, we need the following notion of satisfiability. Let Ψ be a set of $\mathcal{L}$-formulas, and let $\mathcal{M} = \langle S, \nu, V \rangle$ be a neighbourhood model. We say that Ψ *is satisfiable in a subset* $X \subseteq S$ of $\mathcal{M}$, if there is an $s \in X$ such that for all $\psi \in \Psi$, $\mathcal{M}, s \models \psi$. The set Ψ *is finitely satisfiable in* $X \subseteq S$, if any finite subset $\Psi_0 \subseteq \Psi$ is satisfiable in X. Recall (from pg. 281) that $X \subseteq S$ is modally coherent if for all $s, t \in S$: $s \equiv t$ implies $s \in X$ iff $t \in X$.

Definition 10 (Modal saturation). *A neighbourhood model $\mathcal{M} = \langle S, \nu, V \rangle$ is* modally saturated*, if for all modally coherent neighbourhoods $X \in \nu[S]$, and all sets Ψ of modal $\mathcal{L}$-formulas the following holds:*
(i) If Ψ is finitely satisfiable in X, then Ψ is satisfiable in X, and
(ii) If Ψ is finitely satisfiable in X^c, then Ψ is satisfiable in X^c.

The reason we need modally saturated models is that they allow quotienting with the modal equivalence relation. The property which ensures this *modal quotient* is well-defined is that in a modally saturated model, a modally coherent neighbourhood is definable by a modal formula. The consequence is that modally equivalent states are identified in the modal quotient, and hence behaviourally equivalent via the quotient map, and we have the following proposition.

Proposition 6. *Let $\mathcal{M} = \langle S, \nu, V\rangle$ be a modally saturated neighbourhood model. We have for all $s, t \in S$: $s \equiv t$ iff $s \leftrightarroweq^b t$.*

Since all three equivalence notions coincide on a single model (Proposition 4), we obtain the following corollary.

Corollary 1. *Let $\mathcal{M} = \langle S, \nu, V\rangle$ be a modally saturated neighbourhood model. We have for all $s, t \in S$: $s \equiv t$ iff $s \leftrightarroweq^b t$ iff $s \leftrightarroweq^r t$ iff $s \leftrightarroweq t$.*

Furthermore, it can easily be shown that finite neighbourhood models are modally saturated, hence the modal quotient of the disjoint union of two finite neighbourhood models is well-defined. This means that over the class of finite neighbourhood models, we can always construct a behavioural equivalence containing any given pair of modally equivalent states. In other words, finite neighbourhood models form a Hennessy-Milner class with respect to behavioural equivalence. This, however, is not the case with respect to 2^2-bisimulation or neighbourhood bisimulation, as Examples 1 and 2 in section 3 show. We sum up in the next proposition.

Proposition 7. *Over the class of finite neighbourhood models, modal equivalence implies behavioural equivalence, but not 2^2-bisimilarity nor neighbourhood bisimilarity.*

In the proof of the characterisation theorem, we will need to construct modally saturated models from arbitrary neighbourhood models. The first step towards this is to obtain countably saturated $\mathcal{L}_1$-models. This can be done in the form of ultrapowers using standard first-order logic techniques: Every $\mathcal{L}_1$-model has a countably saturated, elementary extension (see e.g. [6]). The second step is to show that any countably saturated neighbourhood model (viewed as a $\mathcal{L}_1$-model) is modally saturated. This can be proved with a standard argument. We are now ready to prove Theorem 2.

Proof of Theorem 2. It is clear that *2* $\Rightarrow$ *3* $\Rightarrow$ *4* (cf. Proposition 1). To see that *4* $\Rightarrow$ *2*, we only need to recall (cf. [21]) that graphs of bounded morphisms are 2^2-bisimulations. Furthermore, as truth of modal formulas is preserved by behavioural equivalence, *1* $\Rightarrow$ *2* is clear. We complete the proof by showing that *2* $\Rightarrow$ *1*.

Let $\text{MOC}_{\mathbf{N}}(\alpha) = \{st_x(\phi) \mid \phi \in \mathcal{L}, \alpha(x) \models_{\mathbf{N}} st_x(\phi)\}$ be the set of modal consequences of $\alpha(x)$ over the class $\mathbf{N}$. It suffices to show that $\text{MOC}_{\mathbf{N}}(\alpha) \models_{\mathbf{N}} \alpha(x)$, since then by compactness there is a finite subset $\Gamma(x) \subseteq \text{MOC}_{\mathbf{N}}(\alpha)$ such that $\Gamma(x) \models_{\mathbf{N}} \alpha(x)$ and $\alpha(x) \models_{\mathbf{N}} \bigwedge \Gamma(x)$. It follows that $\alpha(x)$ is $\mathbf{N}$-equivalent to $\bigwedge \Gamma(x)$, which is the translation of a modal formula. So suppose $\mathfrak{M}$ is a model in $\mathbf{N}$ and $\text{MOC}_{\mathbf{N}}(\alpha)$ is satisfied at some element s in $\mathfrak{M}$. We must show that $\mathfrak{M} \models \alpha[s]$. Consider the set $T(x) = \{st_x(\phi) \mid \mathfrak{M}_\circ, s \models \phi\} \cup \{\alpha(x)\}$. Using a standard compactness argument, we can show that $T(x)$ is $\mathbf{N}$-consistent, hence $T(x)$ is satisfied at an element t in some $\mathfrak{N} \in \mathbf{N}$. By construction, s and t are modally equivalent. Take now a countably saturated, elementary extension $\mathfrak{U}$ of $\mathfrak{M}+\mathfrak{N}$. Note that $\mathfrak{U} \in \mathbf{N}$, since satisfiablity of NAX is preserved under elementary extensions. Moreover, the images s_U and t_U in $\mathfrak{U}$ of s and t, respectively, are

also modally equivalent, since modal truth is transferred by elementary maps. Now since $\mathfrak{U}$ is modally saturated, it follows from Proposition 6 that s_U and t_U are behaviourally equivalent. The construction is illustrated in the following diagram; $\preceq$ indicates that the map is elementary.

$$\begin{array}{ccccc} \mathrm{MOC}_{\mathbf{N}}(\alpha)[s] \dashv \mathfrak{M} & \xrightarrow{i} & \mathfrak{M} + \mathfrak{N} & \xleftarrow{j} & \mathfrak{N} \models \alpha[t] \\ & & \downarrow \preceq & & \\ & & \mathfrak{U} & & \end{array}$$

Finally, we can transfer the truth of $\alpha(x)$ from $\mathfrak{N}, t$ to $\mathfrak{M}, s$ by using the invariance of $\alpha(x)$ under behavioural equivalences, elementary maps and bounded morphisms (which are functional 2^2-bisimulations and hence also behavioural equivalences). □

5 Discussion and Related Work

The main result in our paper is the characterisation theorem (Theorem 2). Our proof builds on ideas from the original proof of the Van Benthem characterisation theorem ([3]). Using similar techniques we can easily prove Craig interpolation for classical modal logic. In the future, we plan to investigate whether other model-theoretic results from normal modal logic carry over to the neighbourhood setting.

Closely related to our work are also the invariance results by Pauly ([18]) on monotonic modal logic and by Ten Cate et al. ([22]) on topological modal logic. Furthermore there seems to be a connection between our work and the results on Chu spaces in [4] where Van Benthem characterises the Chu transform invariant fragment of a two-sorted first-order logic. We also would want to explore the possibility of proving our result using game-theoretic techniques similar to the ones exploited by Otto ([15]).

We want to stress that the paper also contains observations that might be useful in universal coalgebra. We saw that relational equivalences capture behavioural equivalence on F-coalgebras for an *arbitrary* Set functor F. (see Theorem 1). One advantage of these relational equivalences lies in the fact that they can be characterised by a kind of relation lifting (see Remark 2). Therefore we believe the notion of a relational equivalence might be interesting in situations where the functor under consideration does not preserve weak pullbacks. In particular, we want to explore the exact relationship of our results on relational equivalences and the work by Gumm & Schröder ([9]).

Finally our work might be relevant for coalgebraic modal logic (see e.g. [17]). Our idea can be sketched as follows: Given a collection of *predicate liftings* for a functor F we can turn any F-coalgebra into some kind of neighbourhood frame. We would like to combine this well-known connection with Theorem 2, in order to prove that, under certain assumptions, coalgebraic modal logic can be viewed as the bisimulation invariant fragment of some many-sorted first-order logic.

References

[1] Adámek, J.: Theory of Mathematical Structures. Reidel Publications (1983)
[2] Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49(5), 672–713 (2002)
[3] van Benthem, J.: Correspondence theory. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. II, pp. 167–247. Reidel, Doordrecht (1984)
[4] van Benthem, J.: Information transfer across Chu spaces. Logic Journal of the IGPL 8(6), 719–731 (2000)
[5] Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press, Cambridge (2001)
[6] Chang, C., Keisler, H.: Model Theory. North-Holland, Amsterdam (1973)
[7] Chellas, B.F.: Modal Logic - An Introduction. Cambridge University Press, Cambridge (1980)
[8] Goble, L.: Murder most gentle: The paradox deepens. Philosophical Studies 64(2), 217–227 (1991)
[9] Gumm, H.P., Schröder, T.: Types and coalgebraic structure. Algebra universalis 53, 229–252 (2005)
[10] Hansen, H.H.: Monotonic modal logic (Master's thesis). Research Report PP-2003-24, ILLC, University of Amsterdam (2003)
[11] Hansen, H.H., Kupke, C.: A coalgebraic perspective on monotone modal logic. In: Proceedings CMCS'04. ENTCS, vol. 106, pp. 121–143. Elsevier, Amsterdam (2004)
[12] Hansen, H.H., Kupke, C., Pacuit, E.: Bisimulation for neighbourhood structures. CWI technical report (to appear)
[13] Kurz, A.: Personal communication
[14] Kurz, A.: Logics for Coalgebras and Applications to Computer Science. PhD thesis, Ludwig-Maximilians-Universität (2000)
[15] Otto, M.: Bisimulation invariance and finite models (Association for Symbolic Logic). In: Logic Colloquium '02. Lecture Notes in Logic, Association for Symbolic Logic vol. 27 (2006)
[16] Padmanabhan, V., Governatori, G., Su, K.: Knowledge assesment: A modal logic approach. In: Proceedings of the 3rd Int. Workshop on Knowledge and Reasoning for Answering Questions (KRAQ) (2007)
[17] Pattinson, D.: Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. Theoretical Computer Science 309, 177–193 (2003)
[18] Pauly, M.: Bisimulation for general non-normal modal logic. Manuscript (1999)
[19] Pauly, M.: A modal logic for coalitional power in games. Journal of Logic and Computation 12(1), 149–166 (2002)
[20] Rutten, J.J.M.M.: Relators and metric bisimulations. In: Proccedings of CMCS'98. ENTCS, vol. 11 (1998)
[21] Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249, 3–80 (2000)
[22] ten Cate, B., Gabelaia, D., Sustretov, D.: Modal languages for topology: Expressivity and definability (Under submission)
[23] Vardi, M.: On epistemic logic and logical omniscience. In: Halpern, J. (ed.) Proceedings TARK'86, pp. 293–305. Morgan Kaufmann, San Francisco (1986)
[24] Venema, Y.: Algebras and coalgebras. In: Handbook of Modal Logic, vol. 3, pp. 331–426. Elsevier, Amsterdam (2006)

Algebraic Models of Simultaneous Multithreaded and Multi-core Processors

N.A. Harman

Department of Computer Science, University of Wales Swansea,
Swansea SA2 8PP, UK
n.a.harman@swan.ac.uk

Abstract. Much current work on modelling and verifying microprocessors can accommodate pipelined and superscalar processors. However, superscalar and pipelined processors are no longer state-of-the-art: Simultaneous Multithreaded (SMT) and Multi-core, or Chip-Level Multithreaded (CMT) microprocessors enable a single microprocessor implementation to present itself to the programmer as multiple (virtual in the case of SMT) processors with shared state. This paper builds on a series which has developed a hierarchy of many-sorted algebraic models, able to model a variety of processor types, at different levels of temporal and data abstraction. These models address both the behavioural definition of microprocessors, and also the question: what does it mean for a microprocessor implemenation to be correct? They also consider how the process of formal verification can be simplified by indentifying some easily-checked preconditions (the *one-step theorems*). We extend the existing algebraic tools for modeling microprocessors and their correctness to SMT and CMT. We outline how the one-step theorems for simplifying verification are modified for SMT and CMT processors. The particular problems that are addressed are: how to map multiple (possibly interacting) user-level SMT/CMT models to a single implementation? And how to accommodate the (unavoidable) presence of implementation level components in the user-level model?

Keywords: many-sorted algebra, microprocessors, correctness, verification, threaded.

Formal approaches to the correctness of computer hardware are now well established and enjoy a degree of industrial adoption, with a range of impressive examples [1,7,17]. However, despite this success, there is little attention paid to (a) *how* hardware systems should be modelled, and (b) what it *formally* means for an implementation to be correct with respect to a specification. Modelling techniques are wide-ranging and correctness concepts are - sometimes subtly and sometimes wildly - different; often without a clear statement of what 'correctness' means in a particular instance. For example, some correctness models consider only the passage of a single, isolated, instruction through an execution pipeline, and do not consider interaction (or *dependencies*)between instructions. This paper forms part of a series that attempts to address these issues: focussing specifically on algebraic models of the behaviour and correctness of different classes

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 294–311, 2007.
© Springer-Verlag Berlin Heidelberg 2007

of microprocessor implementation. Past work has addressed, within an algebraic framework, microprogrammed [15]; pipelined [10]; and superscalar [9] processors. The pipelined model has been used to verify the ARM6 microprocessor: the first 'commercial' processor to be verified not explicitly developed with verification in mind [7,8]. Here, the model is extended to *Simultaneous Multithreaded* (SMT)[1] and *Chip-Level Multithreaded* (CMT) (commonly called *Multi-Core*) processors. Currently, the model described in this paper is the only developed approach to modelling such processors, which include later Pentium IVs (SMT) and Core Duo (CMT).

Microprocessor implementations are growing in complexity. The simplest and most obvious case is an implementation G of a processor specification F in which the execution of each machine instruction is completed before the next one starts: now restricted to simple embedded processors. The timing relationship between specification F and G in such a case is trivial: each clock cycle in F corresponds uniquely with one or more cycles in G. More complex timing relationships increase the complexity of the timing relationship between F and G.

– *Pipelined* implementations overlap the execution of machine instructions. Consequently the relationship between clock cycles in F and G is no longer unique. However, it is still always possible to identify a unique time in G corresponding with the start/end of instructions in F.
– *Superscalar* implementations attempt to execute multiple instructions simultaneously, sometimes out of program order. In the (common) event of multiple instructions ending simultaneously (or out of order) there may be no identifiable cycle of G's clock S corresponding with a cycle of F's clock T.
– SMT and CMT implementations behave like multiple specification level processors F. In the case of SMT these processors are virtual, and in the case of CMT they are real. In both cases, some of the implementation processor state is *shared* between specification level processors - more in the case of SMT than CMT (where it is typically limited to cache). Furthermore, SMT processors are by definition also superscalar; in principle, CMT processors need not be, but currently-available examples are.

In this paper we extend a suite of existing algebraic models and correctness definitions [6,9,10,15] to accommodate SMT and CMT processors. The structure of the paper is as follows: first we briefly review the field; then we introduce the basic concepts of the model: a structured hierarchy of algebras representing processors; clocks dividing up time; retimings relating clocks at different levels of abstraction and their extension to superscalar timing models; iterated map models of microprocessors; correctness models and the one-step theorems that simplify the verification process. Then we extend the existing iterated map model and correctness model to accommodate SMT and CMT processors. Finally, we consider a simple abstract example.

[1] Called *Hyperthreading* by Intel.

1 Related Work

There is a substantial body of work devoted to microprocessor verification which we briefly summarize (neglecting work on verifying processor *fragments*.). A common characteristic of much of the work is the need to address a specific, usually complex[2] example.

Early work includes [11], which models and verifies a simple processor with eight instructions. This particular example, and variations, was subsequently a common case study [16,12]. A more substantial example was *Viper* which was intended for commercial use [4] though note the rigour of its 'verified' status was controversial (leading to unresolved legal action). This leads to the 'obvious' observation that even with rigorous verification is is difficult to ensure the a verified design is actually *manufactured* correctly.

A landmark leading to much subsequent work is [2] which develops the concept of *flushing* and the notion of verification based on comparisons of fragments of execution traces with appropriate timing (and data) abstraction - though some of these concepts appeared earlier in [22] and the pre-publication versions of [15,16]. Such techniques can be classified (in the useful terminology of [20] which describes a recent evolution of the technique) as *simulation based correspondence*. Techniques derived from [2] have been successfully applied to a wide range of examples (as have the techniques described in [7,8]). Key concepts like the relationship between time at different levels of abstraction, and how it can be addressed appear in [19]. Work on pipelined and superscalar models has parallels with our own model: substantial differences are that although the concept of *timing abstraction* is present, formally the notion of *time itself* is often not. The presence of explicit time in our model does entail proofs to establish that proofs based on finite state traces do lead to valid correctness proofs. However, these are only required once for each variation of the model: see Section 4.

Another interesting classifying question for pipelined and superscalar processors is: are specification state components distributed in time in an implementation [23]; or (our position) are they *functions* of implementation state components from the *same* time? Consider a three-stage pipeline in which instructions are fetched three cycles ahead of execution[3]. Is the the program counter *pc* in the specification the value from the implementation three [specification] cycles earlier, or the current value less three[4]? The latter view always enables us to separate timing and data abstraction maps whereas the former tends to result in a combined data and timing abstraction map.

A significant alternative to techniques derived from [2] (though still owing much to it) are the *completion functions* of [17]. In this, the effect of each stage of an execution pipeline on the programmer-visible state of a processor is considered separately (modeled by individual maps termed completion functions). This provides an obvious and useful partition of the complete verification obligation,

[2] At least with reference to the state-of-the-art in processor verification at the time.

[3] Neglecting complications to do with branching and so on.

[4] More likely, some multiple of three.

since each of these can be considered separately. Completion functions have also been successfully applied in practice.

Recent work addresses progressively larger and more complex examples: the VAMP project, [1], the ARM6 verification project at Cambridge[7,8], and Hunt's Group Austin,Texas[20].

2 Clocks and Basic Models of Computers

A clock T is a single-sorted algebra $T = (\mathbf{N} \mid 0, t+1)$ denoting intervals of time called *clock cycles*. Time is defined in terms of events and not vice versa: that is, rather than events being distributed in some (probably inflexible) sequence of time, typically clock cycles mark the beginning/end of 'interesting' events, and need not be equal in length as measured in 'real' time.

Systems are modelled by a hierarchy of algebraic models.

- A single-sorted *state algebra* $St = (A, T \mid F)$ where A is the state set and $F : T \times A \to A$ is an *iterated map*.
- State algebra St is implemented in terms of a many-sorted *next-state algebra* $Nst = (A, T \mid init, next)$ with F defined as follows

$$F(0, a) = init(a),$$
$$F(t+1, a) = next(F(t, a))$$

  where $init : A \to A$ and $next : A \to A$ are the *initialization* and *next-state* functions.
- In turn, Nst is implemented in terms of a (most probably) many-sorted *machine algebra*: finite bit sequences and operations on bit sequences (of, generally, various different lengths) typical of those found in low-level hardware. Here state set A of St is a Cartesian product constructed from simpler state components.

In general, we expect machine algebra operations to be at most [simultaneous] primitive recursive functions. Hence F is usually a simultaneous primitive recursive function. In practice, the majority of the effort in constructing a microprocessor representation is concentrated on the definition of *next* and *init*.

2.1 Algebraic Basis

In practice, simple many-sorted algebra is sufficient to define microprocessor models within our algebraic hierarchy: particularly abstract descriptions of the models and simple examples (as in this paper). Furthermore, we can in such cases employ conventional mathematical notation, which is compact and readable. We also generally adhere to the initial model. However, although we have no particular desire to complicate the theoretical foundations of our model, the practicalities of dealing with large examples demand some machine-readable form (to date we have usually chosen *Maude* [3] though other options are obviously possible). In such cases we find order sorting and membership algebra [18] to be extremely convenient tools even though they are not *formally* necessary for our purposes.

2.2 Initialization Functions in Iterated Map Models

The rôle of initialization functions is *not* to describe the initial behaviour of a system. Rather it is to eliminate unwanted *starting states in traces*. For example, consider an implementation with memory m, program counter pc and instruction register ir: we may initially require $ir = m(pc)$, and hence not wish to consider starting states that do not have this property. In practice, initialization functions are commonly not required for Programmer's Model descriptions of processors as often (though not always) all programmer-visible states are legal. In our simple example, this is the case: also, and unusually, the implementation of the example does not need an initialization function (Section 6).

The choice of initialization function will vary according to circumstances. However, our usually-preferred initialization function $init : A \to A$ leaves initial state $a \in A$ unchanged *provided* a is already consistent with correct future state traces of F. Typically, there will be a (possibly large) conjunction of (possibly complex) relations (for example, $ir = m(pc)$) κ between the components of A that must be true for correct future traces. We can regard κ as a consistency-checking *invariant* that must hold, at certain times, for the correct state evolution of F: in the case where F represents the implementation of a microprocessor, those times will correspond to the start/end of machine instructions[5]. Invariant κ may be checked by an initialization function h, on initial state $a \in A$: if κ holds, then $h(a) = a$. Such initialization functions are an important part of the verification process (Sections 3 and 4), and, together with duration functions, are analogous to the *pipeline invariants* of [5] and others.

3 Correctness Models for Non-pipelined, Pipelined and Superscalar Processors

Correctness models relate iterated maps $F : T \times A \to A$ and $G : S \times B \to B$ by mapping some $s \in S$ to some $t \in T$; and some $b \in B$ to some $a \in A$. If G is a *correct implementation* of F, then we can construct a commutative diagram[6]:

$$\begin{array}{ccc} T \times A & \xrightarrow{F} & A \\ \uparrow \alpha & & \uparrow \beta \\ S \times B & \xrightarrow{G} & B. \end{array} \tag{1}$$

In order to define correctness in a particular instance, we must construct maps α and β. The majority of our effort is devoted to time: the relationship between state sets A and B is a typically a simple projection $\psi : B \to A$. However, timing abstraction is complex.

[5] Identifying start/end times of instructions may be problematic in superscalar examples: [6,9].

[6] Note that generally the diagram will *not* commute for all $s \in S$, and usually not for all $b \in B$.

We define a *retiming* $\lambda : S \to T$ to be a surjective (all specification times occur in the implementation) and monotonic (time does not go backwards) map. The timing relationship between microprocessor models at different levels of abstraction depends on the state of the processor. Hence, retimings are typically parameterized by the implementation state, and since microprocessors are deterministic, λ is uniquely determined by the initial implementation state.

$$\lambda : B \to [S \to T]$$

In addition to retimings, we also need *immersions* $\overline{\lambda} : T \to S$:

$$\overline{\lambda}(t) = \text{least } s \mid \lambda(s) \geq t$$

and the *start* operator $start : [S \to T] \to [S \to S]$

$$start(\lambda)(s) = \overline{\lambda}\lambda(s).$$

Figure 1 shows a retiming λ between two clocks, S and T, together with the corresponding immersion and start function.

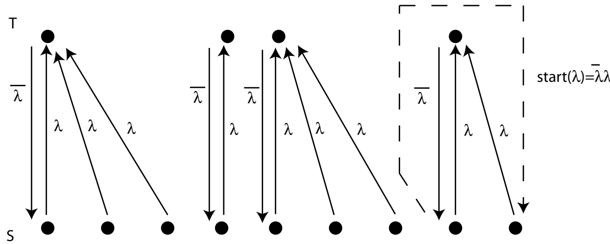


Fig. 1. A retiming from T to S with associated immersion and tools

Microprocessors can be modelled at different levels of abstraction. We are concerned with the lowest level accessible by a programmer: the *programmer's model PM* (commonly called the *architecture* and the most abstract implementation level: the *abstract circuit model AC*.(variously called the *organization, microarchitecture* or *register transfer level.*) Clock cycles in PM models are most conveniently chosen to correspond with machine instructions: in AC models they are [some multiple of] system clock cycles.

Definition 1. *A non-pipelined microprocessor implementation G of AC is correct with respect to a specification F of PM if and only if the state of G under data abstraction map ψ is identical to the state of F for all times $s \in S$ corresponding with the start/end of cycles of T. That is, for all $s = start(\lambda)(s)$:*

$$F(\overline{\lambda}(b)(s), \psi(b)) = \psi(G(s, b)).$$

Alternatively, if the following diagram commutes:

$$\begin{array}{ccc} T \times A & \xrightarrow{F} & A \\ \Big\uparrow (\lambda, \psi) & & \Big\uparrow \psi \\ S \times B & \xrightarrow{G} & B. \end{array}$$

In a pipelined processor, instruction execution overlaps, and *during instruction execution* we cannot uniquely relate cycles of S with cycles of T. However, instructions *terminate* at unique times: If instruction i terminates at time s_i, then no other instruction will terminate at time s_i. Provided retiming λ relates s to the time $t_i \in T$ corresponding with the end of instruction i and the start of $i+1$, our correctness model above still applies [10].

Superscalar processors attempt to execute multiple [pipelined] instructions in parallel, and instructions are allowed to terminate simultaneously, or out of program order. We cannot uniquely associate cycles of S corresponding with the start/end of instructions with cycles of T. Our approach is based on the following: in the event that instructions i and $i+1$ terminate simultaneously or out of order, it is not meaningful to ask 'is G correct with respect to F after i has terminated but before $i+1$' because there is no such time.

We introduce a new *retirement clock* R marking the completion of one or more instructions, with retimings $\lambda_1 : T \to R$ and $\lambda_2 : S \to R$. Retiming λ_1 captures the relationship between the sequential 'one-at-a-time' execution model of the programmer and the actual order of instruction completion; λ_2 marks instruction completion times with respect to the system clock. The non-surjective *adjunct retiming* $\rho : S \to T$ constructed by

$$\rho(s) = \overline{\lambda}_1 \lambda_2(s)$$

relates system clock times and the completion of machine instructions.

Our existing correctness model still applies if we replace retiming λ with adjunct retiming ρ [6].

We can relate two hierarchies of state, next-state and machine algebras for each of F and G with a *correctness algebra Corr*:

$$Corr = (A, B, T, S \mid F, G, \lambda, \psi).$$

In the case of a superscalar processor:

$$Corr = (A, B, T, S, R \mid F, G, \lambda_1, \lambda_2, \psi),$$

where adjunct retiming ρ is constructed λ_1 and λ_2.

4 One-Step Theorems

The obvious correctness proof for the models in section 3 is by induction over clock S. However, we can eliminate induction by using the *one-step theorems*. Given iterated map $G : S \times B \to B$ and state-dependent retiming $\lambda : B \to [S \to T]$, we require two conditions.

– Iterated map G is *time-consistent*. That is:

$$G(\overline{\lambda}(b)(t) + \overline{\lambda}(b)(t'), b) = G(\overline{\lambda}(b)(t), G(\overline{\lambda}(b)(t'), b))$$

for all $s = start(\lambda)(s)$.

– Retiming λ is *uniform*. That is for all $t \in T$

$$\overline{\lambda}(b)(t+1) - \overline{\lambda}(b)(t) = dur(G(\overline{\lambda}(b)(t), b)),$$

where $dur : B \to \mathbf{N}^+$ is a *duration function*.

Informally, microprocessors are functions only of their state (and possibly inputs) at any given time $s \in S$ of some clock: not of the numerical value of s. The conditions above establish the independence of AC model G from the *numerical value* of s. As well as being necessary conditions for the application of the one-step theorems, time-consistency and uniformity are characteristics of real hardware, so models that did not possess them would be flawed.

Recall (Section 2.2) that not all elements of an implementation state set B will be consistent with correct execution of G. Time-consistency requires a carefully constructed initialization function $init$ which leaves all 'legal' states unchanged: otherwise, for some state $b \in B$, $init(b) \neq b$ and hence time-consistency would not hold. In practice, implementations are so complex that identifying legal states can be difficult, and consequently defining $init$ is often difficult. A systematic mechanism is introduced in [10] that can be used whenever the contents of a pipeline are uniquely determined at time s by its contents at time s. This is not always the case - consider an example with two integer execution units, in which the unit chosen for a particular instructions is determined by which has the shortest queue: the contents of the queues may be a function of instructions that have already left the pipeline.

To establish that retiming λ is uniform, it is sufficient to define its immersion in terms of a duration function:

$$\begin{aligned}\overline{\lambda}(b)(0) &= 0,\\ \overline{\lambda}(b)(t+1) &= \overline{\lambda}(b)(t) + dur(G(\overline{\lambda}(b)(t), b)).\end{aligned}$$

Because a typical implementation G is extremely complex, defining dur independently of G is usually prohibitively difficult. The usual definition is *non-constructive* of the form:

$$dur(b) = \text{least } s \mid end(G(s, b)),$$

where $end : B \to \mathbf{B}$ is some function that identifies when one (or more in superscalar processors) instructions have completed. In the case that G forms part of the definition of dur and λ, our correctness model makes no statement about how long each instruction will take to execute.

Initially, it seems that establishing time-consistency requires induction. However, the first *one-step theorem* addresses this.

Theorem 1. *Given iterated map $G : S \times B \to B$, and retiming $\lambda : S \to [S \to T]$ then to establish*

$$G(\overline{\lambda}(b)(t) + \overline{\lambda}(b)(t'), b) = G(\overline{\lambda}(b)(t), G(\overline{\lambda}(b)(t'), b))$$

for all $s = start(\lambda)(s)$, it is sufficient to show that:

$$G(0, b) = init_G(G(0, b)), \text{ and}$$
$$G(\overline{\lambda}(b)(1), b) = init(G(\overline{\lambda}(b)(1), b)).$$

The proof [6] is omitted.

The second one-step theorem can be used to establish correctness.

Theorem 2. *Given time-consistent iterated maps $F : S \times A \to A$ and $G : S \times B \to B$, and uniform retiming $\lambda : S \to [S \to T]$ then to establish*

$$F(\overline{\lambda}(b)(s), \psi(b)) = \psi(G(s, b))$$

for all $s = start(\lambda)(s)$, it is sufficient to show that:

$$F(0, \psi(b)) = \psi(G(0, b)), \text{ and}$$
$$F(1, \psi(b)) = \psi(G(\overline{\lambda}(b)(1), b)).$$

The proof [6] is omitted.

We omit discussion of the cases when F and G are related by an adjunct retiming ρ, and when F and G also depend on *input-output streams*, other than to say that the one step theorems still hold [6,10] .

5 VTM Model Definition

In this section, we consider how we can extend our existing microprocessor model to accommodate SMT and CMT. From the perspective of an operating system kernel programmer, an SMT/CMTprocessor appears as multiple PM-level processors in which some state is *shared*. These processors will be implemented, collectively, by a single AC-level model We will use the term *Virtual Thread Model* (VTM) to distinguish these specification-level processors from the conventional PM level. In practice, we typically first define a (simple) PM-level model and then extend it to become a VTM model in a relatively mechanical way.

The temporal relationship with *other* VTM processors is exposed via the shared state. This relationship is defined by state information that is *not* present in the VTM state, but is in the *implementation* (ITM) state. For example, one implementation may choose to prioritize one thread at the expense of others as a function of state elements not visible at the VTM level, while another implementation *of the same VTM level model* may not. Consequently, VTM models must be defined over not only a PM state set, but also by at least part of the corresponding ITM state. In this paper we choose to use the complete ITM state. However, there is a case for introducing a new, intermediate, level of

abstraction [13]. Consequently, we will subsequently use the term *Intermediate Thread Processor* (ITM) for the AC-level implementation of a group of VTM-level processors, to preserve this possible distinction.

Consider a SMT/CMT processor that is able to execute n threads - that is, it appears to be n (virtual) processors. Each processor F^i_{VTM}, $i \in \{1,\ldots, n\}$, will operate over its own clock T_i; the state of F^i_{VTM} will be composed of some parts that are *local* to F^i_{VTM} and some parts that are *shared* with $F^1_{\mathrm{VTM}},\ldots,$ $F^{i-1}_{\mathrm{VTM}}, F^{i+1}_{\mathrm{VTM}},\ldots, F^n_{\mathrm{VTM}}$. We assume, without loss of generality, that the private state elements $priv \in \Sigma^{\mathrm{priv}}_{\mathrm{VTM}}$ precede the shared state elements $share \in \Sigma^{\mathrm{share}}_{\mathrm{VTM}}$ in the state vector:

$$\Sigma_{\mathrm{VTM}} = \Sigma^{\mathrm{priv}}_{\mathrm{VTM}} \times \Sigma^{\mathrm{share}}_{\mathrm{VTM}}$$

The state trace of each VTM processor will be a function of its own local and shared state, and the shared state of all other VTM processors. There is only one shared state in the ITM level implementation. However each individual VTM model has its own copy of the shared state, *from its own perspective*: a conceit we wish to maintain. Consequently we need to *merge* the shared states of each VTM level processor.

Each VTM processor operates with its own clock: to correctly merge shared states from different VTM processors, we must match states at the appropriate times. We can relate times on different VTM processors using the [adjunct] retimings and corresponding immersions between VTM and ITM level clocks: $\rho_i(b)\overline{\rho}_j(b)(t_j)$ is the time on clock T_i corresponding to time $t_j \in T_j$ given starting (implementation) state $b \in B$.

Definition 2. *Given clocks T_i, $i \in \{1,\ldots, n\}$; ITM state set Σ_{ITM}; VTM state set Σ_{VTM}; adjunct retimings $\rho_i : \Sigma_{\mathrm{ITM}} \to [S \to T_i]$; private state projection functions $\pi^i_{\mathrm{priv}} : \Sigma_{\mathrm{VTM}} \to \Sigma^{\mathrm{priv}}_{\mathrm{VTM}}$;* merge operators *$\tau_i : (\Sigma_{\mathrm{VTM}})^n \to \Sigma^{\mathrm{share}}_{\mathrm{VTM}}$, for $i \in \{1,\ldots,n\}$; initialization function $init : \Sigma_{\mathrm{VTM}} \to \Sigma_{\mathrm{VTM}}$; data abstraction maps $\psi_i : \Sigma_{\mathrm{ITM}} \to \Sigma_{\mathrm{VTM}}$ and next-state function $next : \Sigma_{\mathrm{VTM}} \to \Sigma_{\mathrm{VTM}}$, we model an individual VTM level processor F^i_{VTM} as follows.*

$$\begin{aligned}
&F^i_{\mathrm{VTM}} : \Sigma_{\mathrm{ITM}} \to [T_i \times \Sigma_{\mathrm{VTM}} \to \Sigma_{\mathrm{VTM}}] \\
&F^i_{\mathrm{VTM}}(\sigma_{\mathrm{AC}})(0, \sigma_{\mathrm{VTM}}) = init(\sigma_{\mathrm{VTM}}) \\
&F^i_{\mathrm{VTM}}(\sigma_{\mathrm{AC}})(t+1, \sigma_{\mathrm{VTM}}) = next[(\pi^i_{\mathrm{priv}}(F^i_{\mathrm{VTM}}(\sigma_{\mathrm{AC}})(t, \sigma_{\mathrm{VTM}}))), \\
&\qquad \tau_i(F^i_{\mathrm{VTM}}(\sigma_{\mathrm{AC}})(t, \sigma_{\mathrm{VTM}}), \\
&\qquad\quad F^1_{\mathrm{VTM}}(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_1(\sigma_{\mathrm{AC}})(t), \psi_1(\sigma_{\mathrm{AC}})), \\
&\qquad\qquad \vdots \\
&\qquad\quad F^{i-1}_{\mathrm{VTM}}(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_{i-1}(\sigma_{\mathrm{AC}})(t), \psi_{i-1}(\sigma_{\mathrm{AC}})), \\
&\qquad\quad F^{i+1}_{\mathrm{VTM}}(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_{i+1}(\sigma_{\mathrm{AC}})(t), \psi_{i+1}(\sigma_{\mathrm{AC}})), \\
&\qquad\qquad \vdots \\
&\qquad\quad F^{n}_{\mathrm{VTM}}(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_{n}(\sigma_{\mathrm{AC}})(t), \psi_{n}(\sigma_{\mathrm{AC}})))].
\end{aligned}$$

Note that as well as being parameterized by the ITM-level state $\Sigma_{\rm ITM}$, our VTM-level definition contains the state dependent adjunct retimings ρ_i. Recall that it is usual to define ρ_i in terms of some ITM-level implementation (Section 4). Consequently, the ITM implementation is deeply embedded in the definition of a VTM level model. We return to this issue in Section 7.

A significant issue in our VTM level model is that next-state and initialization functions are from a [probably existing] PM level model. Since the definitions of *next* and *init* are by far the most complex part of model definition, it is important to be able to reuse them which is the case here.

Observe that $\rho_i(\sigma_{\rm AC})\overline{\rho}_i(\sigma_{\rm AC})(t) = t$ and $\sigma_{\rm VTM} = \psi_i(\sigma_{\rm AC})$ (for appropriately chosen $\sigma_{\rm VTM}$ and $\sigma_{\rm AC}$). Hence we can slightly simplify definition 2:

Definition 3. *Given* T_i, $i \in \{1,\ldots, n\}$*;* $\Sigma_{\rm ITM}$*;* $\Sigma_{\rm VTM}$*;* $\rho_i : \Sigma_{\rm ITM} \to [S \to T_i]$*;* $\pi^i_{\rm priv} : \Sigma_{\rm VTM} \to \Sigma^{\rm priv}_{\rm VTM}$*;*$\tau_i : (\Sigma_{\rm VTM})^n \to \Sigma^{\rm share}_{\rm VTM}$*, for* $i \in \{1,\ldots,n\}$*; init :* $\Sigma_{\rm VTM} \to \Sigma_{\rm VTM}$*;* $\psi_i : \Sigma_{\rm ITM} \to \Sigma_{\rm VTM}$ *and next :* $\Sigma_{\rm VTM} \to \Sigma_{\rm VTM}$ *as in definition 2, we model an individual* VTM *level processor* $F^i_{\rm VTM}$ *as follows.*

$$
\begin{aligned}
&F^i_{\rm VTM} : \Sigma_{\rm ITM} \to [T_i \times \Sigma_{\rm VTM} \to \Sigma_{\rm VTM}]\\
&F^i_{\rm VTM}(\sigma_{\rm AC})(0, \sigma_{\rm VTM}) = init(\sigma_{\rm VTM})\\
&F^i_{\rm VTM}(\sigma_{\rm AC})(t+1, \sigma_{\rm VTM}) = next[(\pi^i_{\rm priv}(F^i_{\rm VTM}(\sigma_{\rm AC})(t, \sigma_{\rm VTM}))),\\
&\qquad \tau_i(F^1_{\rm VTM}(\sigma_{\rm AC})(\rho_i(\sigma_{\rm AC})\overline{\rho}_1(\sigma_{\rm AC})(t), \psi_1(\sigma_{\rm AC})),\\
&\qquad\qquad \vdots\\
&\qquad\quad F^n_{\rm VTM}(\sigma_{\rm AC})(\rho_i(\sigma_{\rm AC})\overline{\rho}_n(\sigma_{\rm AC})(t), \psi_n(\sigma_{\rm AC})))].
\end{aligned}
$$

The definitions above represent the most general case: in some circumstances they can be simplified, depending on the requirements of the merge operators τ_i that unify the various shared state components of the n VTM processors. The definitions of τ_i will depend on the precise nature of the shared state $\Sigma^{\rm share}_{\rm VTM}$; and the behaviour of the processor implementation - for example, if two VTM processors attempt to update the same state unit simultaneously. Commonly, the shared state will consist of the processor's main memory. In some circumstances, the definitions of τ_i will not be functions of the private state; and in others all the merge operators τ_i are identical: see [13].

5.1 Correctness of the VTM Model

We now consider what it means for a VTM level model to be correctly implemented by an ITM level model. Note that because there are n VTM level processors corresponding to each ITM level processor, there are n separate correctness statements.

Definition 4. *ITM model* $G : S \times \Sigma_{\rm ITM} \to \Sigma_{\rm ITM}$ *is said to be a* correct implementation *of VTM model* $F^i_{\rm VTM} : \Sigma_{\rm ITM} \to [T_i \times \Sigma_{\rm VTM} \to \Sigma_{\rm VTM}]$, $i \in \{1,\ldots,n\}$ *if, given adjunct retimings* $\rho_i \in Ret(\Sigma_{\rm ITM}, S, T_i)$ *and surjective data abstraction map* $\psi : \Sigma_{\rm ITM} \to \Sigma_{\rm VTM}$*, then for each clock* T_i*, for all* $s = start(\rho_i(\sigma_{\rm AC}))(s)$

and $\sigma_{\mathrm{AC}} \in \Sigma_{\mathrm{ITM}}$*, the following diagrams commute for* $i \in \{1, \ldots, n\}$

$$\begin{array}{ccc} T_i \times \Sigma_{\mathrm{VTM}} & \stackrel{F^i_{\mathrm{VTM}}(\Sigma_{\mathrm{ITM}})}{\longrightarrow} & \Sigma_{\mathrm{VTM}} \\ \uparrow (\rho_i, \psi_i) & & \uparrow \psi_i \\ S \times \Sigma_{\mathrm{ITM}} & \stackrel{F_{\mathrm{ITM}}}{\longrightarrow} & \Sigma_{\mathrm{ITM}}. \end{array}$$

Alternatively:

$$F_{\mathrm{ITM}}(\overline{\rho}_i(\sigma_{\mathrm{AC}})(s), \psi(\sigma_{\mathrm{AC}})) = \psi(F^i_{\mathrm{VTM}}(\sigma_{\mathrm{AC}})(s, \sigma_{\mathrm{AC}})).$$

5.2 Extending the One-Step Theorems to the VTM Model

In considering how we can apply the one-step theorems to our VTM model, recall that the ITM-level processor $G : S \times B \to B$ is essentially identical to any conventional AC processor model: that is, it represents the implementation of a more abstract specification. We are interested in it correctness with respect to some uniform adjunct retiming ρ at times s such that $s = start(\rho)(s)$. Hence the existing one-step theorems show how to establish that G is time-consistent. We can establish the uniformity of retiming ρ_i by construction in terms of a duration function (section 4). However, we must establish that the time-consistency of the collection of VTM level processors F_i, $i \in \{1 \ldots n\}$ can be determined by a variation of the first one-step theorem (theorem 1).

Theorem 3. *Let* $F_i : \Sigma_{\mathrm{ITM}} \to [T_i \times \Sigma_{\mathrm{VTM}} \to \Sigma_{\mathrm{VTM}}]$ *be the* i^{th} *component of a* VTM *level processor. Let* $G : S \times \Sigma_{\mathrm{ITM}} \to \Sigma_{\mathrm{ITM}}$ *be a time-consistent (candidate)* ITM*-level implementation of* F_i, $i \in \{1,\ldots,n\}$*, and let* $\lambda \in \Sigma_{\mathrm{ITM}} \to [S \to T_i]$ *be a uniform retiming. For all* $t \in T_i$ *and* $i \in \{1,\ldots,n\}$*:*

$$F_i(G(0, \sigma_{\mathrm{AC}})(t + t', \sigma_{\mathrm{VTM}}) = F_i(G(\overline{\rho}_i(\sigma_{\mathrm{AC}})(t'))(t, F_i(G(0, \sigma_{\mathrm{AC}}))(t', \sigma_{\mathrm{AC}}))$$

if and only if

$$\begin{aligned} F_i(\sigma_{\mathrm{AC}})(0, \sigma_{\mathrm{VTM}}) &= h_i(F_i(\sigma_{\mathrm{AC}})(0, \sigma_{\mathrm{VTM}})), \text{ and} \\ F_i(\sigma_{\mathrm{AC}})(1, \sigma_{\mathrm{VTM}}) &= h_i(F_i(\sigma_{\mathrm{AC}})(1, \sigma_{\mathrm{VTM}})) \end{aligned}$$

Note that we do not require that G is a *correct* implementation of F (which would result in a circular argument); only that it is time-consistent.

Proof. Given that λ is a uniform retiming and G is a time-consistent iterated map, the state trace of $F_i(G(0, \sigma_{\mathrm{AC}}))(t + t', \sigma_{\mathrm{VTM}})$ is as follows:

$$\begin{aligned} & F_i(G(0, \sigma_{\mathrm{AC}})(0, \sigma_{\mathrm{VTM}}), \\ F_i(G(0, \sigma_{\mathrm{AC}})(1, \sigma_{\mathrm{VTM}}) =& F_i(G(\overline{\lambda}_i(\sigma_{\mathrm{AC}})(1), \sigma_{\mathrm{VTM}}))(0, \sigma_{\mathrm{VTM}}), \\ & \ldots\ldots \\ F_i(G(0, \sigma_{\mathrm{AC}})(t', \sigma_{\mathrm{VTM}}) =& F_i(G(\overline{\lambda}_i(\sigma_{\mathrm{AC}})(t'), \sigma_{\mathrm{VTM}}))(0, \sigma_{\mathrm{VTM}}), \\ F_i(G(0, \sigma_{\mathrm{AC}})(t' + 1, \sigma_{\mathrm{VTM}}) =& F_i(G(\overline{\lambda}_i(\sigma_{\mathrm{AC}})(t'), \sigma_{\mathrm{VTM}}))(1, \sigma_{\mathrm{VTM}}), \\ & \ldots\ldots \\ F_i(G(0, \sigma_{\mathrm{AC}})(t + t', \sigma_{\mathrm{VTM}}) =& F_i(G(\overline{\lambda}_i(\sigma_{\mathrm{AC}})(t'), \sigma_{\mathrm{VTM}}))(t, \sigma_{\mathrm{VTM}}). \end{aligned}$$

(Note that at this point the *correctness* of the state trace is not important: only its *time consistency*.) Observe that the state trace passes through $F_i(G(\overline{\lambda}_i(\sigma_{\mathrm{AC}})(t'), \sigma_{\mathrm{VTM}}))(0, \sigma_{\mathrm{VTM}})$, and hence we can stop execution after t' cycles, reset the clock to zero and and restart execution for a further t cycles, ending in the same final state provided:

$$F_i(\sigma_{\mathrm{AC}})(t, \sigma_{\mathrm{VTM}}) = init(F_i(\sigma_{\mathrm{AC}})(t, \sigma_{\mathrm{VTM}})).$$

By induction over t:

$$\begin{aligned}
F_i(\sigma_{\mathrm{AC}})(t+1, \sigma_{\mathrm{VTM}}) =& next[(\pi^i_{\mathrm{priv}}(F_i(\sigma_{\mathrm{AC}})(t, \sigma_{\mathrm{VTM}}))),\\
& \tau_i(F_1(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_1(\sigma_{\mathrm{AC}})(t), \psi_1(\sigma_{\mathrm{AC}})),\\
& \qquad \vdots\\
& \quad F_n(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_n(\sigma_{\mathrm{AC}})(t), \psi_n(\sigma_{\mathrm{AC}})))],\\
=& F_i(\sigma_{\mathrm{AC}})(1, init((\pi^i_{\mathrm{priv}}(F_i(\sigma_{\mathrm{AC}})(t, \sigma_{\mathrm{VTM}}))),\\
& \tau_i(F_1(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_1(\sigma_{\mathrm{AC}})(t), \psi_1(\sigma_{\mathrm{AC}})),\\
& \qquad \vdots\\
& \quad F_n(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_n(\sigma_{\mathrm{AC}})(t), \psi_n(\sigma_{\mathrm{AC}}))))),\\
=& F_i(\sigma_{\mathrm{AC}})(1, (\pi^i_{\mathrm{priv}}(F_i(\sigma_{\mathrm{AC}})(t, \sigma_{\mathrm{VTM}}))),\\
& \tau_i(F_1(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_1(\sigma_{\mathrm{AC}})(t), \psi_1(\sigma_{\mathrm{AC}})),\\
& \qquad \vdots\\
& \quad F_n(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_n(\sigma_{\mathrm{AC}})(t), \psi_n(\sigma_{\mathrm{AC}})))),\\
=& init(F_i(\sigma_{\mathrm{AC}})(1, (\pi^i_{\mathrm{priv}}(F_i(\sigma_{\mathrm{AC}})(t, \sigma_{\mathrm{VTM}}))),\\
& \tau_i(F_1(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_1(\sigma_{\mathrm{AC}})(t), \psi_1(\sigma_{\mathrm{AC}})),\\
& \qquad \vdots\\
& \quad F_n(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_n(\sigma_{\mathrm{AC}})(t), \psi_n(\sigma_{\mathrm{AC}}))))),\\
=& init(next[(\pi^i_{\mathrm{priv}}(F_i(\sigma_{\mathrm{AC}})(t, \sigma_{\mathrm{VTM}}))),\\
& \tau_i(F_1(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_1(\sigma_{\mathrm{AC}})(t), \psi_1(\sigma_{\mathrm{AC}})),\\
& \qquad \vdots\\
& \quad F_n(\sigma_{\mathrm{AC}})(\rho_i(\sigma_{\mathrm{AC}})\overline{\rho}_n(\sigma_{\mathrm{AC}})(t), \psi_n(\sigma_{\mathrm{AC}})))]),\\
=& init(F_i(\sigma_{\mathrm{AC}})(t+1, \sigma_{\mathrm{VTM}})).
\end{aligned}$$

6 A Simple Example

We now consider a simple example of SMT/CMT. In practice, SMT/CMT processors are extremely complex: they are usually by their nature superscalar, and even simple superscalar examples are complex [6]. A CMT microprocessor

example appears in [14], and an SMT microprocessor example is in development, both based on extensions of the pipelined and superscalar examples in [6,9]. However, in this paper, we restrict ourselves to a simple example, that captures the key features of SMT/CMT: an implementation that can implement multiple (virtual) specifications, that are in turn able to interact via shared memory. We first define the conventional PM level model TR; then we define an ITM level implementation TRH which implements two virtual TR processors, with a shared state element; and finally we use the next-state and initialization functions from TR to define a VTM level model TRV.

$TR : T \times \Sigma_{\mathrm{TR}} \to \Sigma_{\mathrm{TR}}$ consists of a memory $m \in [\mathbf{N} \to \mathbf{N}]$ and a memory address register $mr \in \mathbf{N}$ computing over a clock T.

$$\Sigma_{\mathrm{TR}} = [\mathbf{N} \to \mathbf{N}] \times \mathbf{N}.$$

On each cycle of clock T, TR will compute the function $next_{\mathrm{TR}} : \Sigma_{\mathrm{TR}} \to \Sigma_{\mathrm{TR}}$

$$next_{\mathrm{TR}}(m, mr) = m[mr/m[mr] + m[mr-1]], mr + 1.$$

(We use the notation $m[j]$ to represent the jth element of m, and $m[i/j]$ to represent the replacement of the ith element of m by j.) Because all states of TR are legal, an initialization function is not required. We define TR as follows

$$\begin{aligned} TR(0, m, mr) &= m, mr, \\ TR(t+1, m, mr) &= next_{\mathrm{TR}}(TR(t, m, mr)). \end{aligned}$$

The implementation $TRH : S \times \Sigma_{\mathrm{TRH}} \to \Sigma_{\mathrm{TRH}}$ implements two virtual TR processors as before, with memory m shared and mr private. Hence the state of TRH is:

$$\Sigma_{\mathrm{TRH}} = [\mathbf{N} \to \mathbf{N}] \times \mathbf{N}^2$$

We define TRH as follows

$$\begin{aligned} TRH(0, m, mr_1, mr_2) &= (m, mr_1, mr_2), \\ TRH(s+1, m, mr_1, mr_2) &= next_{\mathrm{TRH}}(TRH(s, m, mr_1, mr_2)), \end{aligned}$$

where $next_{\mathrm{TRH}} : \Sigma_{\mathrm{TRH}} \to \Sigma_{\mathrm{TRH}}$ is defined as

$$\begin{aligned} &next_{\mathrm{TRH}}(m, mr_1, mr_2) = \\ &\qquad m[mr_2/m[mr_2] + m[mr_2 - 1]][mr_1/m[mr_1] + m[mr_1 - 1]]. \end{aligned}$$

Again, an initialization function is not required. Observe that we perform the mr_2 write first, so in the event that both writes are to the same memory word, the mr_1 result will replace the mr_2 result.

The VTM level model TRV consists of TRV_1 and TRV_2:

$$\begin{aligned} TRV_1 &: \Sigma_{\mathrm{TRH}} \to [T_1 \times \Sigma_{\mathrm{TR}} \to \Sigma_{\mathrm{TR}}] \\ TRV_2 &: \Sigma_{\mathrm{TRH}} \to [T_2 \times \Sigma_{\mathrm{TR}} \to \Sigma_{\mathrm{TR}}] \end{aligned}$$

$TRV_i,\ i \in \{1,2\}$ will be defined in terms of the PM level next-state function $next_{\mathrm{TR}}$, projection functions $\pi^{\mathrm{TRV,i}}_{\mathrm{priv}} : \Sigma_{\mathrm{TRH}} \to \mathbf{N},\ i \in \{1,2\}$, data abstraction maps $\psi_i : \Sigma_{\mathrm{TRH}} \to \Sigma_{\mathrm{TR}}$, merge operators $\tau^{\mathrm{TRV}}_i : ([\mathbf{N} \to \mathbf{N}] \times \mathbf{N})^2 \to [\mathbf{N} \to \mathbf{N}] \times \mathbf{N},\ i \in \{1,2\}$; and retimings $\lambda_i : \Sigma_{\mathrm{TRH}} \to [S \to T_i],\ i \in \{1,2\}$ defined as follows.

$$\begin{aligned}
\pi^{\mathrm{TRV,1}}_{\mathrm{priv}}(m, mr_1, mr_2) &= mr_1,\\
\pi^{\mathrm{TRV,2}}_{\mathrm{priv}}(m, mr_1, mr_2) &= mr_2,\\
\psi_1(m, mr_1, mr_2) &= (m, mr_1),\\
\psi_2(m, mr_1, mr_2) &= (m, mr_2),
\end{aligned}$$

with merge operators

$$\begin{aligned}
&\tau^{\mathrm{TRV}}_1(m1, mr1, m2, mr2) =\\
&\qquad m2[mr1/m1[mr1]
\end{aligned}$$

and

$$\begin{aligned}
&\tau^{\mathrm{TRV}}_2(m2, mr2, m1, mr1) =\\
&\qquad \begin{cases} m1, & \text{if } mr1 = mr2\\ m1[mr2/m2[mr2]], & \text{otherwise,}\end{cases}
\end{aligned}$$

The first merge operator τ^{TRV}_1 copies the changes made by $TRV1$ regardless of the value of mr_1; however τ^{TRV}_2 only copies changes made by TRV if they will not overwrite the immediately proceeding change made by $TRV1$.

The retimings λ_i and corresponding immersions are simply:

$$\lambda_i(m, b, mr1, mr2)(s) = s,\ i = \{1,2\},$$

and

$$\overline{\lambda}(m, b, mr1, mr2)(t) = t.$$

We can define TRV_1 and TRV_2 (expanding the trivial definitions of the projection operators and retimings) as follows:

$$\begin{aligned}
TRV_1(m, mr_1, mr_2)(0, m, mr_1) &= (m, mr_1),\\
TRV_1(m, mr_1, mr_2)(t+1, m, mr_1) &= next_{\mathrm{TR}}(\tau^{\mathrm{TRV}}_1(TRV_1(m, mr_1, mr_2)(t, m, mr_1),\\
&\qquad TRV_2(m, mr_1, mr_2)(t, (m, mr_2)), mr_1),\\
TRV_2(m, mr_1, mr_2)(0, m, mr_1) &= (m, mr_2),\\
TRV_1(m, mr_1, mr_2)(t+1, m, mr_2) &= next_{\mathrm{TR}}(\tau^{\mathrm{TRV}}_2(TRV_2(m, mr_1, mr_2)(t, m, mr_2),\\
&\qquad TRV_1(m, mr_1, mr_2)(t, (m, mr_1)), mr_2).
\end{aligned}$$

6.1 Correctness of the Example

To establish the correctness of the example above we must:

- establish the time consistency of the VTM and ITM models; and
- establish the correctness of the ITM model with respect to the VTM models using theorem 2.

Time Consistency of the VTM and ITM Models

While we can apply theorem 3 to establish the time consistency of the VTM models TRV_1 and TRV_2 it is simpler in this case to observe that all iterated maps without initialization functions are time-consistent [6].

Correctness of the ITM Model

To establish the correctness of the ITM model TRH, we must show that

$$\begin{aligned} TRV_1(0, m, mr_1, mr_2) &= \psi_1(TRH(0, m, mr_1, mr_2); \\ TRV_1(1, m, mr_1, mr_2) &= \psi_1(TRH(1, m, mr_1, mr_2); \\ TRV_1(0, m, mr_1, mr_2) &= \psi_2(TRH(0, m, mr_1, mr_2); \text{ and} \\ TRV_1(1, m, mr_1, mr_2) &= \psi_2(TRH(1, m, mr_1, mr_2). \end{aligned} \tag{2}$$

We omit the (identity) retiming immersions. The correctness of the equations in 2 follows trivially from the definitions.

7 Concluding Remarks

We have extended our existing algebraic models of microprocessors and their correctness to superscalar SMT and CMT processor implementations, which represent the state-of-the-art in current commercial implementation. The algebraic model developed here is the first to address such processors. However, although we can successfully model such processors, and define what it means for them to be correct, practical verification of realistic examples would be a formidable undertaking. (This is generally the case: practical verifications of complete non-trivial processors are currently limited to non-superscalar pipelined processors.) There are some practical steps that can be taken to reduce complexity: we omit discussion here, but see [13]. Although helpful, these simplifications are unlikely to make realistic examples practical at the current time. Nonetheless, we feel that a considered approach to modeling processors and their correctness that runs ahead of actual application is useful: the modeling approaches to pipelined processors that were ultimately used to verify ARM6 [7,8] were developed some years in advance of their practical use.

A point worthy of comment is the presence of the definition of the ITM level implementation in the definition of the VTM level model. This should not be surprising: in practice, the implementation of an SMT or CMT processor does impact the behaviour seen by programmers; and the timing behaviour of all

processors is a function of their implementation. This last fact is generally acknowledged in our model by the definition of retimings in terms of the ITM-level model. There has been a general weakening of the long-established separation of processor architecture and implementation: good compilers for modern processors need to be aware of implementation details (e.g. how many functional units are there, and of what type) in order to generate high-quality code, particularly in the case of modern superscalar processors.

Finally, observe that a situation similar to SMT occurs with operating system kernels: a single physical processor presents as multiple virtual processors. The situation is somewhat different (in a kernel a privileged virtual processor at the higher level of abstraction, rather than the lower) but we believe the work here can be adapted to accommodate operating system kernels. Together with [21] on modelling high- and low-level languages and their relationships, this would produce a chain of fundamentally identical algebraic models from high-level languages to abstract hardware.

References

1. Beyer, S., Jacobi, C., Kröning, D., Leinenbach, D., Paul, W.: Instantiating uninterpreted functional unit and memory system: Functional verification of VAMP. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 51–65. Springer, Heidelberg (2003)
2. Burch, J., Dill, D.: Automatic verification of pipelined microprocessor control. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 68–80. Springer, Heidelberg (1994)
3. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: The Maude 2.0 system. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 76–87. Springer, Heidelberg (2003)
4. Cohn, A.: A proof of correctness of the VIPER microprocessor: the first levels. In: Birtwistle, G., Subrahmanyam, P.A. (eds.) VLSI Specification, Verification and Synthesis, pp. 27–72. Kluwer Academic Publishers, Dordrecht (1987)
5. Cyrluk, D., Rushby, J., Srivas, M.: Systematic formal verification of interpreters. In: IEE international conference on formal engineering methods ICFEM'97, pp. 140–149 (1997)
6. Fox, A.C.J.: Algebraic Representation of Advanced Microprocessors. PhD thesis, Department of Computer Science, University of Wales Swansea (1998)
7. Fox, A.C.J: Formal specification and verification of ARM6. In: Basin, D., Wolff, B. (eds.) TPHOLs 2003. LNCS, vol. 2758, pp. 25–40. Springer, Heidelberg (2003)
8. Fox, A.C.J.: An algebraic framework for verifying the correctness of hardware with input and output: a formalization in HOL. In: Fiadeiro, J.L., Harman, N., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 157–174. Springer, Heidelberg (2005)
9. Fox, A.C.J., Harman, N.A.: Algebraic models of superscalar microprocessor implementations: A case study. In: Möller, B., Tucker, J.V. (eds.) Prospects for Hardware Foundations. LNCS, vol. 1546, pp. 138–183. Springer, Heidelberg (1998)
10. Fox, A.C.J., Harman, N.A.: Algebraic models of correctness for abstract pipelines. The Journal of Algebraic and Logic Programming 57, 71–107 (2003)
11. Gordon, M.: Proving a computer correct with the LCF-LSM hardware verification system. Technical report, Technical Report No. 42, Computer Laboratory, University of Cambridge (1983)

12. Graham, B., Birtwistle, G.: Formalising the design of an SECD chip. In: Leeser, M., Brown, G. (eds.) Hardware Specification, Verification and Synthesis: Mathematical Aspects. LNCS, vol. 408, pp. 40–66. Springer, Heidelberg (1990)
13. Harman, N.A: Algebraic models of simultaneous multi-threaded and chip-level multi-threaded microprocessors. Submitted to the Journal of Algebraic and Logic Programming (2007)
14. Harman, N.A: Modelling SMT and CMT processors: A simple case study. Technical Report CSR7-2007, University of Wales Swansea, Computer Science Department (2007), http://cs.swan.ac.uk/reports/yr2007/CSR7-2007.pdf
15. Harman, N.A., Tucker, J.V.: Algebraic models of microprocessors: Architecture and organisation. Acta Informatica 33, 421–456 (1996)
16. Harman, N.A., Tucker, J.V.: Algebraic models of microprocessors: the verification of a simple computer. In: Stavridou, V. (ed.) Proceedings of the 2nd IMA Conference on Mathematics for Dependable Systems, pp. 135–170 (1997)
17. Hosabettu, R., Gopalakrishnan, G., Srivas, M.: Formal verification of a complex pipelined processor. Formal Methods in System Design 23(2), 171–213 (2003)
18. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (ed.) WADT 1997. LNCS, vol. 1376, pp. 18–61. Springer, Heidelberg (1997)
19. Miller, S., Srivas, M.: Formal verification of the AAMP5 microprocessor: a case study in the industrial use of formal methods. In: Proceedings of WIFT 95, Boca Raton (1995)
20. Ray, S., Hunt, W.A.: Deductive verification of pipelined machines using first-order quantification. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 31–43. Springer, Heidelberg (2004)
21. Stephenson, K.: Algebraic specification of the Java virtual machine. In: Möller, B., Tucker, J.V. (eds.) Prospects for Hardware Foundations. LNCS, vol. 1546, Springer, Heidelberg (1998)
22. Windley, P.: A theory of generic intepreters. In: Milne, G.J., Pierre, L. (eds.) CHARME 1993. LNCS, vol. 683, pp. 122–134. Springer, Heidelberg (1993)
23. Windley, P., Burch, J.: Mechanically checking a lemma used in an automatic verification tool. In: Srivas, M., Camilleri, A. (eds.) FMCAD 1996. LNCS, vol. 1166, pp. 362–376. Springer, Heidelberg (1996)

Quasitoposes, Quasiadhesive Categories and Artin Glueing

Peter T. Johnstone[1], Stephen Lack[2], and Paweł Sobociński[3,⋆]

[1] DPMMS, University of Cambridge, United Kingdom
[2] School of Computing and Mathematics, University of Western Sydney, Australia
[3] ECS, University of Southampton, United Kingdom

Abstract. Adhesive categories are a class of categories in which pushouts along monos are well-behaved with respect to pullbacks. Recently it has been shown that any topos is adhesive. Many examples of interest to computer scientists are not adhesive, a fact which motivated the introduction of quasiadhesive categories. We show that several of these examples arise via a glueing construction which yields quasitoposes. We show that, surprisingly, not all such quasitoposes are quasiadhesive and characterise precisely those which are by giving a succinct necessary and sufficient condition on the lattice of subobjects.

1 Introduction

Adhesive categories, introduced in [8], are a class of categories where pushouts along monos exist and are well-behaved with respect to pullbacks. They capture several examples of interest to computer scientists, in particular presheaf toposes. Amongst other applications, they have allowed the generalisation of several aspects of the theory of graph transformations. Several results which before were proved concretely at the level of the category of graphs and graph homomorphisms **Graph** have been generalised and shown to hold in any adhesive category. Because adhesive categories also enjoy useful closure properties, such theory is widely applicable.

Recently it has been shown by the second and the third authors that toposes are adhesive categories [10]. This result, while perhaps not surprising, is useful because topos theory is a well-established branch of mathematics with wide relevance to diverse fields such as logic, geometry and topology. Adhesive categories have less structure than toposes, meaning for example that they enjoy more closure properties (for instance, adhesive categories are closed under coslice). In particular, there are adhesive categories which are not toposes.

Early in the development of the theory of adhesive categories it became clear that the class of adhesive categories was too restrictive for several important examples, notably many arising from the theory of algebraic specifications. In such categories the class of regular monos (the monos which arise as equalisers) differs from the class of all monos. Since it is easy to show that all monos in an adhesive category are regular, it is immediate that the examples are not instances of

⋆ Research partially supported by EPSRC grant EP/D066565/1. The second author gratefully acknowledges the support of the Australian Research Council.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 312–326, 2007.
© Springer-Verlag Berlin Heidelberg 2007

adhesive categories. However, it was also clear that pushouts along *regular* monos enjoyed many of the properties which pushouts along monos enjoy in adhesive categories – for instance such pushouts are also pullbacks and regular monos are stable under pushout. The examples motivated the theory of quasiadhesive categories in which pushouts along regular monos are well-behaved with respect to pullbacks. An example of interest to computer scientists is the category of termgraphs [3]. There have been several other attempts at generalising the original definition of adhesivity, notably adhesive HLR categories [5] and weak adhesive HLR categories [6].

Here we return to some of the examples which first motivated the introduction of quasiadhesive categories and show that they all arise as instances of a glueing construction. As a consequence of this, we show that the categories are quasitoposes. Roughly, quasitoposes are to toposes as quasiadhesive categories are to adhesive categories, in the sense that much of the structure is assumed only of regular monos (in a topos, as in an adhesive category, all monos are regular). In fact, the nomenclature of topos theory motivated the name 'quasiadhesive'.

The central result of the paper can be considered surprising: quasitoposes are a generalisation of toposes roughly as quasiadhesive categories are a generalisation of adhesive categories and since as mentioned previously, toposes are adhesive, one could expect also that quasitoposes are quasiadhesive. As we shall show, this is not the case. In fact, we shall characterise which quasitoposes are quasiadhesive: precisely those where unions of regular subobjects are regular in the subobject lattice. The proof of the main result is interesting in part because it constructs a direct counterexample in any quasitopos in which the condition fails. An example of a quasitopos which is not quasiadhesive is the category of binary relations **BRel**, also known as the category of simple graphs.

Returning to our examples, we take advantage of the fact that they arise uniformly and exhibit a sufficient and necessary condition on the functor along which gluing occurs for the resulting category to be quasiadhesive. This characterisation allows us immediately to derive which of the categories are quasiadhesive. For instance, the category of injective functions, **Inj** is a quasiadhesive quasitopos while the quasitopos **Spec** of algebraic specifications is not quasiadhesive.

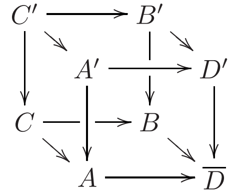
Structure of the paper. In §2 we recall the definitions of adhesive and quasiadhesive categories. In §3 we introduce our main motivating examples. We show that these examples arise uniformly as a certain full subcategory of a category obtained by Artin glueing in §4 and prove that such categories are quasitoposes. We prove that a quasitopos is quasiadhesive if and only if unions of regular subobjects are regular in §5. Using this result, we show a necessary and sufficient condition on the glueing functor which allows us to immediately show which of the examples are quasiadhesive. We conclude in §6.

2 Preliminaries

Here we shall briefly recall the notions of adhesive and quasiadhesive categories together with a few of their properties. Adhesive and quasiadhesive categories

rely on the notion of a van Kampen (VK) square. Van Kampen squares are pushouts which satisfy a certain axiomatic condition.

Definition 1. A *van Kampen square* is a pushout which satisfies the following: for any commutative cube in which it is the bottom face and which has the left and rear faces pullbacks, the front and right faces are pullbacks if and only if the top face is a pushout. Another way of stating the "only if" part of the above condition is that such a pushout is required to be stable under pullback.



A category **C** is adhesive when it has pullbacks, pushouts along monos and such pushouts are VK squares. All monos are regular in an adhesive category.

Lemma 2. *Monos and regular monos coincide in any adhesive category.*

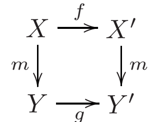
Proof. See [9, Lemma 4.9]. □

A category **C** is quasiadhesive when it has pullbacks, pushouts along regular monos and such pushouts are VK squares. An adhesive category is a quasiadhesive category where all the monos are regular, in this sense adhesive categories can be regarded as "degenerate" quasiadhesive categories.

3 Motivating Examples

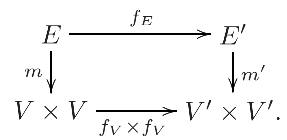
In this section we introduce several examples which fail to be adhesive. As we shall see in Section 4, all of them can be seen as instances of a particular construction and as a consequence all are quasitoposes.

Example 3 (Injective Functions). Let **Inj** be the category with objects the injective functions $m: X \to Y$ in **Set** and arrows commutative diagrams as illustrated to the right.



An object of **Inj** can be thought of as a set together with a chosen subset, equivalently a set equipped with a unary predicate. The morphisms are those functions which preserve the subset/predicate in the obvious way. The monos are precisely those where g is an injective function. The regular monos are those monos which reflect the predicate; this condition is easily seen to be equivalent to requiring that the resulting diagram of monos in **Set** is a pullback.

Example 4 (Binary relations). The category of binary relations, **BRel** has as objects triples $\langle V, E, m\rangle$ with $m: E \to V \times V$ an injective function. Arrows are the obvious commutative diagrams:



It follows easily that **BRel** is equivalent to the category of graphs with at most one edge from one vertex to another (ie $\{ \langle V, E \rangle \mid E \subseteq V \times V \}$), and arrows ordinary graph morphisms. Some authors refer to such objects as simple graphs. As with **Inj**, the monos in **BRel** are easily seen to be the diagrams arising from f_V being injective. Again, the regular monos are those which reflect edges, or equivalently pullback diagrams with all maps being monos.

The following example appeared in [4]. Fix an arbitrary nonempty set $\mathcal{P}$ of *predicates* with an arity function $ar\colon \mathcal{P} \to \mathbb{N}$. Given a set of *atoms* X, a predicate $P \in \mathcal{P}$, and $x_1, \ldots, x_{arP} \in X$, the formal expression $P(x_1, \ldots, x_{arP})$ is called an *atomic formula over* X. Let $AF_{\mathcal{P}}(X)$ denote the set of atomic formulas.

Example 5 (Structures). Let $\mathbf{Str}_{\mathcal{P}}$ be the category where the objects are *structures* – pairs $\langle X, Y \rangle$ where X is a set and $Y \subseteq AF_{\mathcal{P}}(X)$. A structure morphism $\langle f, g \rangle$ where $f\colon X \to X'$ and $g\colon Y \to Y'$ such that $g(P(x_1, \ldots, x_{arP})) \equiv P(fx_1, \ldots, fx_{arP})$ – that is, atomic formulas are preserved.

The monos are clearly those homomorphisms where the map on the underlying sets is injective. The regular monos are those where also predicates are reflected, in the obvious way.

Example 6 (Algebraic specifications). A *signature* is a quadruple $\Sigma = \langle S, P, s, t \rangle$ where S is a set of *sorts*, P is a set of *operators*, $s\colon P \to S^*$ (where S^* denotes the free monoid over S) is a function giving the sorts of the domain of an operator and and $t\colon P \to S$ is the function giving the sort of the codomain of an operator. We write $p\colon s_1 \times \ldots \times s_k \to s$ if $s(p) = s_1 \ldots s_k$ and $t(p) = s$. A *signature morphism* $f\colon \Sigma \to \Sigma'$ is a pair $f = \langle f_S, f_P \rangle$ where $f_S\colon S \to S'$, $f_P\colon P \to P'$ such that if $p\colon s_1 \times \ldots \times s_k \to s$ then $f_P(p)\colon f_S(s_1) \times \ldots \times f_S(s_k) \to f_S(s)$. The category **Sig** of signatures and signature morphisms is a presheaf topos – indeed, it is isomorphic to the category of hypergraphs[1] with edges restricted to having a single target node (but an arbitrary finite number of source nodes).

A *term* $\sigma(\mathtt{x_1} : s_1, \ldots, \mathtt{x_n} : s_n) : s$ over a signature Σ is the obvious formal construction built up from the basic operators of Σ and composition which may contain instances of variables $\mathtt{x_i} : s_i$. To avoid extra complexity, we shall assume that variables appear at most once within a term. Each term has a unique sort s determined by the codomain sort of the root operator in the syntax tree of the term. An *equation* over a signature Σ is a formal expression of the form $\sigma_1(\mathtt{x_1} : s_1, \ldots, \mathtt{x_n} : s_n) : s \equiv \sigma_2(\mathtt{x_1} : s_1, \ldots, \mathtt{x_n} : s_n) : s$.

An *algebraic specification* is a pair $\mathcal{S} = \langle \Sigma, E \rangle$ where Σ is a signature and E is a set of equations over Σ. An algebraic specification morphism $f\colon \mathcal{S} \to \mathcal{S}'$ is a pair $f = \langle \langle f_S, f_P \rangle, f_E \rangle$ where $\langle f_S, f_P \rangle : \Sigma \to \Sigma'$ is a signature morphism and $f_E\colon E \to E'$ is a function satisfying

$$\begin{aligned} f_E(\ \sigma_1(\mathtt{x_1} : s_1, \ldots, \mathtt{x_n} : s_n) : s \equiv \sigma_2(\mathtt{x_1} : s_1, \ldots, \mathtt{x_n} : s_n) : s\ ) = \\ f_P(\sigma_1)(\mathtt{x}_1' : f_S(s_1), \ldots, \mathtt{x}_n' : f_S(s_n)) : f_S(s) \equiv \\ f_P(\sigma_2)(\mathtt{x}_1' : f_S(s_1), \ldots, \mathtt{x}_n' : f_S(s_n)) : f_S(s) \end{aligned}$$

[1] Some authors use the term ‘multigraphs’.

for some injective renaming of variables $\mathtt{x_i} \mapsto \mathtt{x'_i}$. The category of algebraic specifications and algebraic specification morphisms is denoted **Spec**.

A mono is an algebraic specification morphism with an underlying signature morphism that is injective (on both sorts and operators). A regular mono also reflects equations, in the obvious way.

Example 7 (Safely marked Petri nets). A Petri net is a tuple $N = \langle P, T, s, t \rangle$ where P is a set of places, T is a set of transitions and $s, t \colon T \to P^*$ are, respectively, the sources and targets of a transition. In other words, we think of a net as a multi-graph. A net morphism $f \colon N \to N'$ is a pair $f = \langle f_P, f_T \rangle$ where $f_P \colon P \to P'$, $f_T \colon T \to T'$ such that $s' f_T = f_P^* s$ and $t' f_T = f_P^* t$ (sources and targets of transitions are preserved). Such a choice of morphism can be useful when deriving compositional labelled equivalences for nets, see for instance [13]. The category of Petri nets and morphisms is denoted **PNet**. It is easily seen to be a presheaf topos. A *marked place-transition net* $\mathcal{N}$ is a pair $\langle N, K, k \rangle$ where $N \in$ **PNet**, K is a set of *tokens*, and $k \colon K \to P$ is a mapping of tokens to places. A place-transition net morphism $f \colon \mathcal{N} \to \mathcal{N}'$ is a pair $f = \langle f_N, f_K \rangle$ where $f_N : N \to N'$ is a map of Petri nets and $f_K : K \to K'$ is a function between the sets of tokens satisfying $k' f_K = f_P k$ (places of tokens are preserved). Let **PTNet** be the category of marked place-transition nets and morphisms. A *safely marked place-transition net* is a place-transition net where there is at most one token on each place, that is, the function $k \colon K \to P$ is injective. Let **SPTNet** be the full subcategory of **PTNet** consisting of safely marked nets.

A mono in **SPTNet** is a morphism of marked-nets which is injective on the underlying net. A regular mono has the additional property that the marking is reflected.

Lemma 8. *The categories* **Inj** *(Example 3),* **BRel** *(Example 4),* **Str** *(Example 5),* **Spec** *(Example 6), and* **SPTNet** *(Example 7) are not adhesive.*

Proof. Immediate since the classes of monos and regular monos do not coincide in any of these categories (cf Lemma 2). □

4 Glueing

In this section we shall demonstrate that the examples discussed in §3 are formed using a particular variant of a general construction known as Artin glueing [2].

More explicitly, we shall see that the examples given in §3 are actually certain full subcategories of categories obtained by glueing. Using a well-known result, categories obtained by Artin glueing are quasitoposes. Using the fact that also the aforementioned full subcategories of quasitoposes are themselves quasitoposes (Theorem 16) we know that the examples are quasitoposes.

We begin by recalling the definition of a quasitopos.

Definition 9. A category **C** is said to be a quasitopos when it satisfies all of the following conditions:

(i) it has finite limits and colimits;
(ii) it is locally cartesian closed;
(iii) it has a regular-subobject-classifier.

Quasitoposes and quasiadhesive categories share several basic properties, as we outline below.

Proposition 10. *The following hold in any quasitopos $\mathcal{E}$ and in any quasiadhesive category* $\mathbf{C}$*:*

(i) pushouts along regular monos are also pullbacks;
(ii) regular monos are stable under pushout;
(iii) unions of regular subobjects are effective[2]*;*

Proof. Quasitoposes: (i) and (ii) see [7, A2.6.2] and (iii) see [7, A1.4.3]. Quasiadhesive categories: (i) and (ii) see [9, Lemma 6.1]. Part (iii) is a straightforward generalisation of [9, Theorem 5.1]. □

In addition, quasitoposes admit a number of factorisation systems.

Proposition 11. *In any quasitopos, every arrow can be factorised into an epi followed by a regular mono or by a regular epi followed by a mono.*

Proof. For the regular epi - mono factorisation see [7, Scholium 1.3.5]. For the mono - regular epi factorisation one can use the dual of [7, Scholium 1.3.5] since regular monos are stable under pushout. □

Given categories $\mathbf{C}$, $\mathbf{D}$ and a functor $T\colon \mathbf{D} \to \mathbf{C}$, we write $\mathbf{C}/T$ for the category with objects arrows $f\colon C \to TD$ for $C \in \mathbf{C}$, $D \in \mathbf{D}$. An arrow in $\mathbf{C}/T$ is a pair $\langle g, h\rangle : f \to f'$ consisting of an arrow $g\colon C \to C'$ in $\mathbf{C}$ and $h\colon D \to D'$ in $\mathbf{D}$ such that $(Th)f = f'g$.

$$\begin{array}{ccc} C & \xrightarrow{g} & C' \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f'} \\ TD & \xrightarrow[Th]{} & TD' \end{array}$$

Definition 12 (Artin glueing). A category $\mathbf{Z}$ is said to be obtained by Artin glueing if it is the slice category $\mathbf{C}/T$ for some functor $T\colon \mathbf{D} \to \mathbf{C}$ where $\mathbf{C}$ and $\mathbf{D}$ are quasitoposes.

We shall usually perform Artin glueing along a pullback-preserving functor $T\colon \mathbf{C} \to \mathbf{D}$. It is a well-known "folk" theorem[3] that when $\mathbf{C}$ has finite limits (and this will always be the case for us, since $\mathbf{C}$ will be a quasitopos) then T also preserves all finite limits. The category obtained by glueing along such a functor will in fact be a quasitopos; the following theorem is actually a special case of a more general result of Carboni and Johnstone [2, Theorem 3.3]. We will rely on the 'if' direction which was first shown by Rosebrugh and Wood [11].

[2] Unions of subobjects are said to be effective when the union of two subobjects is obtained by pushing out along their intersection.
[3] See [2, Lemma 1.1] for a proof.

Theorem 13. *If* $\mathbf{C}$, $\mathbf{D}$ *are quasitoposes then* $\mathbf{C}/T$ *is a quasitopos iff* T *preserves pullbacks.* □

We shall denote by $\mathbf{C}/\!\!/T$ the full subcategory of $\mathbf{C}/T$ with objects the monos $m\colon C \to TD$ in $\mathbf{C}$.[4] The following lemma lists some properties of regular monos in $\mathbf{C}/T$ and $\mathbf{C}/\!\!/T$.

Lemma 14. *Suppose that* $\mathbf{C}$, $\mathbf{D}$ *are quasitoposes and* $T\colon \mathbf{D} \to \mathbf{C}$ *preserves pullbacks. Then:*

(i) A map $\langle g, h\rangle$ *in* $\mathbf{C}/T$ *is a regular mono iff* g *is a regular mono in* $\mathbf{C}$ *and* h *is a regular mono in* $\mathbf{D}$.
(ii) A map $\langle g, h\rangle$ *in* $\mathbf{C}/\!\!/T$ *is a regular mono iff* g *is a regular mono in* $\mathbf{C}$, h *is a regular mono in* $\mathbf{D}$ *and the resulting square in* $\mathbf{C}$ *is a pullback diagram.*

Proof. ($\Rightarrow$) Since T preserves pullbacks and $\mathbf{C}$ is finitely complete, T preserves equalisers. In particular, this means that equalisers are constructed componentwise in $\mathbf{C}/T$. It is easy to check that the full subcategory $\mathbf{C}/\!\!/T$ is closed with respect to equalisers and the resulting square is a pullback.
($\Leftarrow$) Suppose that g is a regular mono in $\mathbf{C}$ and that h is a regular mono in $\mathbf{D}$. Let α, β be the cokernel pair of g (obtained by pushing out g along itself in $\mathbf{C}$) and φ, ψ be a pair in $\mathbf{D}$ of which h is the equaliser. Since pushouts along regular monos are also pullbacks in $\mathbf{C}$ (cf Proposition 10, (i)), it follows that g is the equaliser of α and β. Using the fact that α and β are the cokernel pair, let f_2 be the unique map such that $f_2\alpha = T\varphi.f_1$ and $f_2\beta = T\psi.f_1$ (see the first diagram below). It follows that $\langle g, h\rangle$ is the equaliser of $\langle \alpha, \varphi\rangle$ and $\langle \beta, \psi\rangle$ in $\mathbf{C}/T$.

$$\begin{array}{ccccc} C & \xrightarrow{g} & C_1 & \overset{\alpha}{\underset{\beta}{\rightrightarrows}} & C_2 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f_1} & & \vdots\,{\scriptstyle f_2} \\ TD & \xrightarrow[Th]{} & TD_1 & \overset{T\varphi}{\underset{T\psi}{\rightrightarrows}} & TD_2 \end{array} \qquad \begin{array}{ccccc} C & \xrightarrow{g} & C_1 & \overset{T\varphi.f_1}{\underset{T\psi.f_1}{\rightrightarrows}} & TD_2 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f_1} & & \downarrow \\ TD & \xrightarrow[Th]{} & TD_1 & \overset{T\varphi}{\underset{T\psi}{\rightrightarrows}} & TD_2 \end{array}$$

For $\mathbf{C}/\!\!/T$, let φ and ψ be a pair in $\mathbf{D}$ of which h is the equaliser. We shall show that g is the equaliser of $T\varphi.f_1$ and $T\psi.f_1$. Indeed, suppose there is a map $x\colon X \to C_1$ such that $T\varphi.f_1 x = T\psi.f_1 x$. Since T preserves equalisers, there is a unique map $y\colon X \to TD$ such that $Th.y = f_1 x$. Using the fact that the square is a pullback, there is a unique map $z\colon X \to C$ such that $fz = y$ and $gz = x$. It follows that $\langle g, h\rangle$ is the equaliser of $\langle T\varphi.f_1, \varphi\rangle$ and $\langle T\psi.f_1, \psi\rangle$ in $\mathbf{C}/\!\!/T$, as illustrated in the second diagram above. □

In Theorem 16 we shall show that if T is a pullback-preserving functor between quasitoposes then also $\mathbf{C}/\!\!/T$ is a quasitopos. Note that when $\mathbf{C}$ and $\mathbf{D}$ are toposes

[4] One could also define $\mathbf{C}/\!\!/T$ to be the full subcategory of $\mathbf{C}/T$ with objects the *regular* monos. In that case, the conclusion of Theorem 16 would still hold and can be proved using a modified version of Lemma 15. For the purposes of this paper the precise definition used is a moot point since in all of our examples both $\mathbf{C}$ and $\mathbf{D}$ are actually toposes.

then $\mathbf{C}/T$ is also a topos; on the other hand, as will be demonstrated by the examples, $\mathbf{C}/\!\!/T$ will usually be only a quasitopos. We first prove a technical lemma.

Lemma 15. *In the following we assume that* $T\colon \mathbf{D} \to \mathbf{C}$ *preserves finite limits.*

(i) If $\mathbf{C}$ *and* $\mathbf{D}$ *are cartesian closed then also* $\mathbf{C}/\!\!/T$ *is cartesian closed;*
(ii) If $\mathbf{C}$ *and* $\mathbf{D}$ *are locally cartesian closed then also* $\mathbf{C}/\!\!/T$ *is locally cartesian closed;*
(iii) If $\mathbf{C}$ *and* $\mathbf{D}$ *have finite colimits and* $\mathbf{C}$ *has regular epi - mono factorisations then* $\mathbf{C}/\!\!/T$ *has finite colimits;*
(iv) If $\mathbf{D}$ *is a quasitopos then* $\mathbf{C}/\!\!/T$ *has a regular-subobject-classifier.*

Proof. (i) It is well-known (see [2]) that with these assumptions $\mathbf{C}/T$ is cartesian closed. The internal hom of $C_1 \to TD_1$ and $C_2 \to TD_2$ is the pullback $P \to T[D_1, D_2]$ of $[C_1, C_2] \to [C_1, TD_2]$ along $T[D_1, D_2] \to [TD_1, TD_2] \to [C_1, TD_2]$, as illustrated below.

$$\begin{array}{ccccc} P & & \longrightarrow & & [C_1, C_2] \\ \downarrow & & & & \downarrow \\ T[D_1, D_2] & \longrightarrow & [TD_1, TD_2] & \longrightarrow & [C_1, TD_2] \end{array}$$

Clearly if $C_2 \to TD_2$ is mono then so is $[C_1, C_2] \to [C_1, TD_2]$ and therefore also $P \to T[D_1, D_2]$. Since $\mathbf{C}/\!\!/T$ is a full subcategory of the cartesian-closed category $\mathbf{C}/T$ and is closed under exponentiation, it itself is cartesian-closed.
(ii) Suppose that $C \to TD$ is an object of $\mathbf{C}/\!\!/T$. Then $(\mathbf{C}/\!\!/T)/(C \to TD) = (\mathbf{C}/C)/\!\!/T'$, where $T'\colon \mathbf{D}/D \to \mathbf{C}/C$ is given by first applying T to get $\mathbf{D}/D \to \mathbf{C}/TD$, and then pulling back along $C \to TD$ as in $\mathbf{C}/TD \to \mathbf{C}/C$. Now $\mathbf{C}/C$ and $\mathbf{D}/D$ are cartesian closed by assumption, and T' clearly preserves finite limits, so $(\mathbf{C}/C)/\!\!/T'$ is cartesian closed by *(i)*, and so $\mathbf{C}/\!\!/T$ is locally cartesian closed.
(iii) It is well-known that in this case also $\mathbf{C}/T$ has finite colimits. The presence of the regular epi - mono factorisation system in $\mathbf{C}$ ensures that $\mathbf{C}/\!\!/T$ is a reflective subcategory of $\mathbf{C}/T$ and so it too has finite colimits.
(iv) A regular subobject in $\mathbf{C}/\!\!/T$ is a pullback square, as illustrated, in which the horizontal maps are regular mono. If $\mathbf{D}$ has a regular-subobject-classifier W, then it is straightforward to verify that $TW \to TW$ is a regular-subobject-classifier in $\mathbf{C}/\!\!/T$.

$$\begin{array}{ccc} C' & \longrightarrow & C \\ \downarrow & & \downarrow \\ TD' & \longrightarrow & TD \end{array}$$

□

Theorem 16. *If* $\mathbf{C}$*,* $\mathbf{D}$ *are quasitoposes and* $T\colon \mathbf{D} \to \mathbf{C}$ *preserves pullbacks then* $\mathbf{C}/\!\!/T$ *is a quasitopos.*

Proof. The proof follows using the results of Lemma 15 and applying the usual reduction in the style of [2]. Indeed, if $\mathbf{C}$, $\mathbf{D}$ are quasitoposes and $T\colon \mathbf{D} \to \mathbf{C}$ preserves pullbacks then $T_1\colon \mathbf{D} \to \mathbf{C}/T1$ preserves finite limits (since it preserves pullbacks and the terminal object) and so $(\mathbf{C}/T1)/\!\!/T_1$ is a quasitopos. But this is just $\mathbf{C}/\!\!/T$. □

We have given a direct proof of Theorem 16. A shorter proof would use the fact that the objects of $\mathbf{C}/\!\!/T$ are the separated objects for the closure operator corresponding to the fact that $\mathbf{D}$ is an open sub-quasitopos of $\mathbf{C}/T$.

We shall now demonstrate that the examples of §3 are of the form $\mathbf{D}/\!\!/T$ for specific choices of $\mathbf{C}, \mathbf{D}$ and pullback-preserving $T\colon \mathbf{D} \to \mathbf{C}$. Theorem 16 ensures that such categories are quasitoposes, thus eliminating the need for tedious direct proofs.

Proposition 17. *The categories* **Inj**, **BRel**, **Str**, **Spec** *and* **SPTNet** *are of the form* $\mathbf{C}/\!\!/T$ *for* $T\colon \mathbf{D} \to \mathbf{C}$ *a pullback-preserving functor between toposes.*

Proof. **Inj** (cf Example 3): Let $\mathbf{C}, \mathbf{D} = \mathbf{Set}$ and let T be the identity functor. It is immediate that $\mathbf{Inj} \cong \mathbf{Set}/\!\!/T$.

BRel (cf Example 4): Let $\mathbf{C}, \mathbf{D} = \mathbf{Set}$ and let $TX = X \times X$. It is immediate that $\mathbf{BRel} \cong \mathbf{Set}/\!\!/T$; also, T preserves limits since it is representable – indeed, $TX \cong \mathbf{Set}(2, X)$.

Str (cf Example 5): Let $T\colon \mathbf{Set} \to \mathbf{Set}$ be defined $TX = \sum_{P \in \mathcal{P}} X^{ar(P)} \cong \sum_{P \in \mathcal{P}} \mathbf{Set}(ar(P), X)$. Pullback preservation is immediate. Notice that there is a bijection $TX \cong AF_{\mathcal{P}}(X)$, so to give a subset of atomic formulas is essentially to give a mono with codomain TX; hence it is easy to show that $\mathbf{Str}_{\mathcal{P}} \simeq \mathbf{Set}/\!\!/T$.

Spec (cf Example 6): Let $T\colon \mathbf{Sig} \to \mathbf{Set}$ be the free term functor: a signature $\Sigma = \langle S, P, s, t\rangle$ is taken to the set $T\Sigma$ of all terms. The action on signature morphisms is canonical.

$$\begin{array}{ccc} U\Sigma & \xrightarrow{q_2} & T\Sigma \\ {\scriptstyle q_1}\downarrow & & \downarrow{\scriptstyle p} \\ T\Sigma & \xrightarrow[p]{} & S^* \times S \end{array}$$

Let $S^* \times S$ denote the set of nonempty words over S and $p\colon T\Sigma \to S^* \times S$ be the evident map which takes a term to its "type". Define $U\Sigma$ to be the illustrated pullback diagram in $\mathbf{Set}$. It follows that $U\colon \mathbf{Sig} \to \mathbf{Set}$ is a functor. Intuitively, $U\Sigma$ is the set of "well-typed" equations (consisting of two terms of the same result sort and taking an equal number of variables, with each corresponding pair agreeing on the sorts) between terms over Σ. It follows that $\mathbf{Spec} \simeq \mathbf{Set}/\!\!/U$, we omit the proof of the fact that U preserves pullbacks;

SPTNet (cf Example 7): Let $U\colon \mathbf{PNet} \to \mathbf{Set}$ be the forgetful functor which takes a Petri net to its set of places. It follows that $\mathbf{SPTNet} \simeq \mathbf{Set}/\!\!/U$. Using the fact that limits are computed pointwise in $\mathbf{PNet}$, U preserves them. □

Corollary 18. *The categories* **Inj**, **BRel**, **Str**, **Spec** *and* **SPTNet** *are quasitoposes.*

Proof. Immediate by Proposition 17 and Theorem 16. □

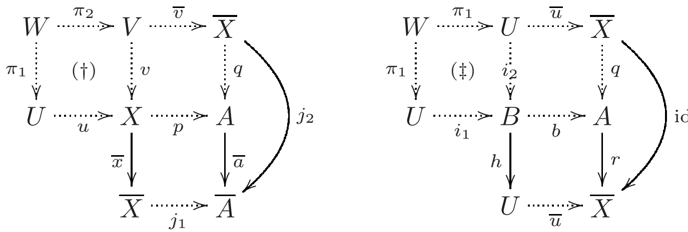
5 Quasitoposes and Quasiadhesive Categories

In this section we shall characterise precisely which quasitoposes are quasiadhesive. We begin by proving an important property of quasiadhesive categories – regular monos are closed under union. This result forms one direction of a characterisation of quasiadhesive quasitoposes, which appears as Theorem 21. The proof itself is a step-by-step construction of a counterexample at an abstract level and is followed by a concrete example in Corollary 20.

Theorem 19. *In quasiadhesive categories, binary unions of regular subobjects are regular*

Proof. Suppose that $Z \in \mathbf{C}$ and that U and V are two regular subobjects of Z such that $U \cup V$ is not a regular subobject. We shall show that $\mathbf{C}$ cannot be quasiadhesive by constructing an explicit counterexample cube. Let W and X denote respectively $U \cap V$ and $U \cup V$. Let $\overline{X}$ denote the smallest regular subobject of Z which contains X, ie the join of U and V in the lattice of regular subobjects of Z. This object can be obtained by factorising the map $X \to Z$ into an epi followed by a regular mono. Some work is required to show that this factorisation can be obtained in any quasiadhesive category, we omit the details and only give a sketch: one first shows that $X \to Z$ admits a cokernel pair; secondly, one can construct the equaliser $\overline{X} \to Z$ of the cokernel pair by pulling back, giving the regular mono part of the factorisation. Finally one uses a standard argument to show that the map $\overline{x}\colon X \to \overline{X}$ given by the universal property of equalisers is epi.

We obtain objects A and $\overline{A}$ by constructing the pushouts in the left diagram below, where $\overline{v} = \overline{x}v$. Clearly $\overline{v}$ is regular mono by the usual cancellation properties.

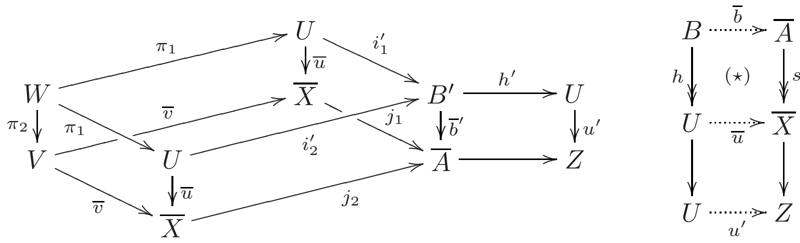


Note that (†) is a pushout by Proposition 10, (iii). All three pushouts are also pullbacks by part (i) of the proposition, since all the horizontal morphisms in the diagram are regular mono. The fact that the lower square is a pullback together with the fact that $\overline{x}$ is not an isomorphism implies that $\overline{a}$ is not an isomorphism.

Now consider the second diagram above in which (‡) is a pushout and $\overline{u} = \overline{x}u$. Using the fact that $\overline{u}\pi_1 = \overline{v}\pi_2$ the pushout of $\overline{X}$ and U along W is A and we obtain a map $b\colon B \to A$ such that the upper right square commutes and $bi_1 = pu$. By the pasting properties of pushouts, this square is also a pushout. Let $h\colon B \to U$ be the codiagonal (the unique map such that $hi_1 = hi_2 = \mathrm{id}_U$) and let r be the unique map which satisfies $rq = \mathrm{id}_{\overline{X}}$ and $rb = \overline{u}h$. The outer rectangle is clearly a pushout and thus the lower square is also a pushout, by cancellation.

Consider the first diagram below, it shows that the pullback of $\overline{A} \to Z$ and $U \to Z$ is the pushout B of U together with itself along W. To see this, let B' denote this pullback and note that the pullback of $U \to Z$ along $\overline{X} \to Z$ is just U and similarly the pullback of U and V is W. Hence all the vertical faces of the diagram are pullbacks and since pushouts along regular monos are stable under pullback, the upper face of the cube must be a pushout diagram; hence $B \cong B'$.

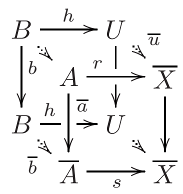
In particular, we can erase all the primes from the diagram.



In the second diagram above, let s be the codiagonal. As we have established, the outer region is a pullback and it is immediate that the lower square is a pullback also – thus the upper square $(\star)$ must be a pullback by cancellation. We claim that also that $(\star)$ is a pushout.

To see this, assume that there are maps $f\colon \overline{A} \to C$ and $g\colon U \to C$ such that $f\overline{b} = gh$. Since h and s are epi (being codiagonals), it is enough to show that there exists a map $\alpha\colon \overline{X} \to C$ such that $\alpha s = f$. Note that by the construction of $\overline{A}$ a morphism $f\colon \overline{A} \to C$ corresponds to a pair of morphisms $f_1 = fj_1 = faq\colon \overline{X} \to C$ and $f_2 = fj_2\colon \overline{X} \to C$. We shall show that we can take α to be f_1; to show $f_1 s = f$, it is enough to show that the two maps agree when precomposed with j_1 and j_2; thus we need to show that $f_1 = f_2$. Clearly f_1 and f_2 agree when restricted to $V \rightarrowtail \overline{X}$. Also $f_1\overline{u} = fj_1\overline{u} = f\overline{b}i_2 = f\overline{a}bi_2 = f\overline{a}q\overline{u} = fj_2\overline{u} = f_2\overline{u}$. Thus they must agree on the union $\overline{x}\colon X \rightarrowtail \overline{X}$ of these subobjects. And x is epi by construction, so $f_1 = f_2$.

Now consider the illustrated cube; the top and bottom faces are both pushouts, the diagonal edges are regular mono and three of the four vertical faces are pullbacks. But the front face is not a pullback, because as we have observed previously, $\overline{a}\colon A \to \overline{A}$ is not an isomorphism.



□

Note that, although we presented the proof of Theorem 19 above as a proof by contradiction, the argument is in fact constructive: it shows that if $\mathbf{C}$ is quasiadhesive then the mono $\overline{a}\colon A \to \overline{A}$ must be an isomorphism, and hence so is $\overline{x}\colon X \to \overline{X}$.

Corollary 20. The construction in the proof of Theorem 19 allows us to conclude that the category of binary relations **BRel** is *not* quasiadhesive. Indeed, considering the objects of **BRel** as simple graphs, it is immediate that the two vertices of $a \to b$ are regular subobjects but their union is not regular. The counterexample constructed starting from these two subobjects is shown in Fig 1.

The following theorem is the main result of this section. It gives a sufficient and necessary condition for a quasitopos to be quasiadhesive. The condition is easy to state and usually straightforward to check – a quasitopos is quasiadhesive if and only if regular subobjects are closed under binary unions.

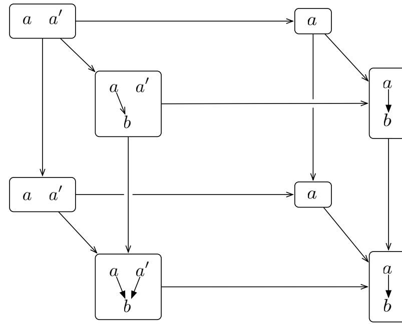


Fig. 1. A counterexample demonstrating that **BRel** is not quasiadhesive

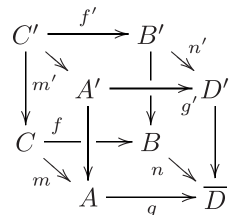
Theorem 21. *Let* $\mathbf{C}$ *be a quasitopos. Then the following are equivalent:*

(i) $\mathbf{C}$ *is quasiadhesive.*
(ii) the class of regular subobjects of any object is closed under binary union.

Proof. (i) $\Rightarrow$ (ii): Is immediate by the conclusion of Theorem 19 and Proposition 11.

(ii) $\Rightarrow$ (i): We give a brief sketch of a "direct" argument. A shorter but slightly more technical proof relies on connections between quasiadhesivity and Artin glueing in quasitoposes.

Consider the illustrated cube: since we can factorise maps into a regular epi followed by a mono and van Kampen squares are closed under pasting, it suffices to consider the situation where f is a mono or f is a regular epi. The case where f is a mono is straightforward; the reasoning for the case where f is a regular epi is similar to the proof of [10, Theorem 25].



□

When are the hypotheses of Theorem 21 satisfied? Clearly, they hold if $\mathbf{C}$ is a topos; in particular, the fact that toposes are adhesive is a consequence, since all monos are regular and by the conclusion of Theorem 21 they are quasiadhesive. In this sense, the above result is a generalisation of the main result of [10]. They also hold if $\mathbf{C}$ is a Heyting algebra, where the only regular monos are isomorphisms.

Proposition 22. *Heyting algebras are quasiadhesive.* □

Clearly, in other quasitoposes, it suffices to check the condition that unions of regular subobjects are regular. We have seen that **BRel** does not satisfy the condition and thus is not quasiadhesive. As our examples are of the form $\mathbf{C}/\!\!/T$, it is useful to understand how unions are computed in such categories.

Lemma 23. *Unions of subobjects in in* $\mathbf{C}/T$ *and* $\mathbf{C}/\!\!/T$ *are computed pointwise. That is, the union of subobjects* $\langle C_1 \rightarrowtail C, D_1 \rightarrowtail D\rangle$ *and* $\langle C_2 \rightarrowtail C, D_2 \rightarrowtail D\rangle$ *of*

$f\colon C \rightarrowtail TD$ is $\langle C_1 \cup C_2 \rightarrowtail C, D_1 \cup D_2 \rightarrowtail D\rangle$, where the unions are formed in $\mathbf{C}$ *and* $\mathbf{D}$*, respectively.*

Proof. $\mathbf{C}/T$ is a quasitopos by Theorem 13; $\mathbf{C}/\!\!/T$ is a quasitopos by Theorem 16 and thus unions are effective in both the categories (Proposition 10, (iii)). But all colimits in $\mathbf{C}/T$ are computed pointwise and pullbacks are computed pointwise because T preserves them. Because $\mathbf{C}/\!\!/T$ is reflective in $\mathbf{C}/T$, it suffices to check that the map $C_1 \cup C_2 \to T(D_1 \cup D_2)$ is a mono, but this follows easily since in the commutative diagram below, $C_1 \cup C_2 \to C$ and $C \to TD$ are mono.

$$\begin{array}{ccc} C_1 \cup C_2 & \longrightarrow & C \\ \downarrow & & \downarrow \\ T(D_1 \cup D_2) & \longrightarrow & TD \end{array}$$

□

Recall that in Theorem 16 we showed that for a pullback-preserving functors $T\colon \mathbf{D} \to \mathbf{C}$ between quasitoposes, $\mathbf{C}/\!\!/T$ is a quasitopos. The following result characterises precisely those T for which $\mathbf{C}/\!\!/T$ is also quasiadhesive.

Theorem 24. *Suppose* $\mathbf{C}$, $\mathbf{D}$ *are quasiadhesive quasitoposes and* $T\colon \mathbf{D} \to \mathbf{C}$ *is pullback-preserving. Then* $\mathbf{C}/\!\!/T$ *is quasiadhesive iff* T *preserves unions of regular subobjects*[5].

Proof. ($\Leftarrow$). By Theorem 16, $\mathbf{C}/\!\!/T$ is a quasitopos. Suppose that T preserves unions of regular subobjects. For $i \in \{1, 2\}$ suppose that we have regular subobjects of $C \to TD$ in $\mathbf{C}/\!\!/T$, as illustrated in the first diagram below. Their union is calculated pointwise (Lemma 23), is illustrated in the second diagram. Because $\mathbf{C}$ and $\mathbf{D}$ are quasiadhesive, the horizontal maps in the second diagram are regular. Using part (ii) of Lemma 14, it suffices to show that the square is a pullback. We show this directly, suppose there are $\alpha\colon X \to C$ and $\beta\colon X \to T(D_1 \cup D_2)$ such that $m\alpha = Td.\beta$. Using the assumption we can take $T(D_1 \cup D_2) = TD_1 \cup TD_2$ and pull back β to obtain a pushout diagram which decomposes X, as illustrated.

$$\begin{array}{ccc} C_i & \xrightarrow{c_i} & C \\ {\scriptstyle m_i}\downarrow & & \downarrow{\scriptstyle m} \\ TD_i & \xrightarrow[Td_i]{} & TD \end{array} \qquad \begin{array}{ccc} C_1 \cup C_2 & \xrightarrow{c} & C \\ {\scriptstyle m_3}\downarrow & & \downarrow{\scriptstyle m} \\ T(D_1 \cup D_2) & \xrightarrow[Td]{} & TD \end{array}$$

$$\begin{array}{l} X_3 \to X_2,\ X_3 \to X_1,\ X_2 \xrightarrow{x_2} X,\ X_1 \xrightarrow{x_1} X,\ X_3 \to TD_1 \cup TD_2,\ X_2 \xrightarrow{\beta_2} TD_2,\ X_1 \xrightarrow{\beta_1} TD_1,\ X \xrightarrow{\beta} TD_1 \cup TD_2 \\ TD_1 \cup TD_2 \to TD_2,\ TD_1 \cup TD_2 \to TD_1,\ TD_2 \to TD_1 \cup TD_2,\ TD_1 \to TD_1 \cup TD_2 \end{array}$$

Using the fact that the subobject diagrams are pullbacks, we obtain $h_i : X_i \to C_i$ such that $m_i h_i = \beta_i$ and $c_i h_i = \alpha x_i$. Using the decomposition of X, we obtain a unique map $h\colon X \to C$ such that $hx_i = j_i h_i$ where $j_i : C_i \to C_1 \cup C_2$. A routine calculation confirms that $ch = \alpha$ and $m_3 h = \beta$.

[5] That is, for $D_1 \to D$, $D_2 \to D$ regular subobjects, $T(D_1 \cup D_1) = TD_1 \cup TD_2$ as subobjects of TD.

($\Rightarrow$) Assume that the quasitopos $\mathbf{C}/\!\!/T$ is quasiadhesive, then unions of regular subobjects are regular by Theorem 21. Let $D_1 \to D$ and $D_2 \to D$ be regular monos. They lead to corresponding regular monos in $\mathbf{C}/\!\!/T$ as illustrated in the diagram for $i = 1, 2$.

$$\begin{array}{ccc} TD_i & \longrightarrow & TD \\ \downarrow & (\dagger) & \downarrow \\ TD_i & \longrightarrow & TD \end{array}$$

The union results in the second diagram above, since unions are computed pointwise (cf Lemma 23). Since the union is a regular subobject, the square is a pullback and thus $T(D_1 \cup D_2) = TD_1 \cup TD_2$.

$$\begin{array}{ccc} TD_1 \cup TD_2 & \longrightarrow & TD \\ \downarrow & & \downarrow \\ T(D_1 \cup D_2) & \longrightarrow & TD \end{array}$$

□

Lemma 25. *The category* **Inj** *of injective functions (cf Example 3) and the category* **SPTNet** *of safely-marked nets(cf Example 7) are quasiadhesive. The categories* $\mathbf{Str}_{\mathcal{P}}$ *where* $\mathcal{P}$ *has predicates of arity* > 1 *(cf Example 5) and* **Spec** *(cf Example 6) are not quasiadhesive.*

Proof. Using Theorem 24 and the fact that all of the examples are of the form $\mathbf{C}/\!\!/T$ (cf Proposition 17), it suffices to check whether T in each case preserves unions of regular subobjects. For **Inj**, $T = \mathrm{id}$ and the condition is obviously satisfied. Similarly, the condition clearly holds for the forgetful functor $U\colon \mathbf{PNet} \to \mathbf{Set}$ since unions in the presheaf topos **PNet** are calculated "pointwise". Any polynomial functor $T\colon \mathbf{Set} \to \mathbf{Set}$ which contains powers ≥ 2 does not preserve the union of the two injections of $1 \to 2$, thus $\mathbf{Str}_{\mathcal{P}}$ is not quasiadhesive when $\mathcal{P}$ has predicates of arity ≥ 2. Similarly, the functor $U : \mathbf{Sig} \to \mathbf{Set}$ does not preserve unions, consider the signature $\mathbf{2}$ with a single sort and two unary predicates and the signature $\mathbf{1}$ with a single sort and a single unary predicate. There are two monos $\mathbf{1} \to \mathbf{2}$ but their union is not preserved by U. □

In Theorem 24 we exhibited a necessary and sufficient condition on T for $\mathbf{C}/\!\!/T$ to be quasiadhesive. The following result shows that the category $\mathbf{C}/T$ obtained by glueing is quasiadhesive if additionally T is cartesian and both $\mathbf{C}$ and $\mathbf{D}$ are quasiadhesive.

Lemma 26. *Let* $\mathbf{C}$ *and* $\mathbf{D}$ *be quasitoposes, and* $T\colon \mathbf{D} \to \mathbf{C}$ *a cartesian functor. Then the quasitopos* $\mathbf{C}/T$ *is quasiadhesive iff both* $\mathbf{C}$ *and* $\mathbf{D}$ *are.*

Proof. As observed in Lemma 14, a mono $\langle m, n\rangle : (A', B', f') \to (A, B, f)$ in $\mathbf{C}/T$ is regular iff both m and n are regular monos. Since unions of subobjects are also constructed 'component-wise', it is easy to see that $\mathbf{C}/T$ inherits the condition on unions of regular subobjects if both $\mathbf{C}$ and $\mathbf{D}$ satisfy it. Conversely, if we have a counterexample to the condition in either $\mathbf{C}$ or $\mathbf{D}$, we can obtain one in $\mathbf{C}/T$ by applying the appropriate direct image functor to it, since both these functors preserve regular monos. □

6 Conclusions and Future Work

We have shown that several examples of interest to computer scientists are quasitoposes obtained by using a variant of the Artin glueing construction. We

characterised the quasitoposes which are quasiadhesive in terms of a condition on the lattice of subobjects. We have refined this condition to the categories which arise from of the aforementioned construction.

There are two clear directions for future work. Firstly, the fact that not all our examples are quasiadhesive raises the question whether one can find a natural class of categories with less structure and more liberal closure conditions than quasitoposes while at the same time covering the basic properties satisfied by adhesive and quasiadhesive categories; useful for applications of rewriting and other related fields, for examples of such properties see for instance [9,1,12]. Secondly, as all of the examples in the present paper are quasitoposes, one could directly evaluate the suitability of rewriting directly on objects in an arbitrary quasitopos and study the resulting theories of parallelism and concurrency.

References

1. Baldan, P., Corradini, A., Heindel, T., König, B., Sobociński, P.: Processes for adhesive rewriting systems. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006 and ETAPS 2006. LNCS, vol. 3921, pp. 202–216. Springer, Heidelberg (2006)
2. Carboni, A., Johnstone, P.T.: Connected limits, familial representability and Artin glueing. Mathematical Structures in Computer Science 5, 441–449 (1995)
3. Corradini, A., Gadducci, F.: On term graphs as an adhesive category. ENCS 127(5), 43–56 (2005)
4. Ehrig, H., Habel, A., Kreowski, H.-J., Parisi-Presicce, F.: Parallelism and concurrency in high-level replacement systems. Mathematical Structures in Computer Science 1 (1991)
5. Ehrig, H., Habel, A., Padberg, J., Prange, U.: Adhesive high-level replacement categories and systems. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, Springer, Heidelberg (2004)
6. Ehrig, H., Prange, U.: Weak adhesive high-level replacement categories and systems: a unifying framework for graph and Petri net transformations. In: Futatsugi, K., Jouannaud, J.-P., Meseguer, J. (eds.) Algebra, Meaning, and Computation. LNCS, vol. 4060, pp. 235–251. Springer, Heidelberg (2006)
7. Johnstone, P.T.: Sketches of an Elephant: A topos theory compendium, vol. 1. Clarendon Press (2002)
8. Lack, S., Sobociński, P.: Adhesive categories. In: Walukiewicz, I. (ed.) FOSSACS 2004. LNCS, vol. 2987, pp. 273–288. Springer, Heidelberg (2004)
9. Lack, S., Sobociński, P.: Adhesive and quasiadhesive categories. Theoretical Informatics and Applications 39(3), 511–546 (2005)
10. Lack, S., Sobociński, P.: Toposes are adhesive. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 184–198. Springer, Heidelberg (2006)
11. Rosebrugh, R.D., Wood, R.J.: Pullback preserving functors. Journal of Pure and Applied Algebra 73, 73–90 (1991)
12. Sassone, V., Sobociński, P.: Reactive systems over cospans. In: LiCS '05, pp. 311–320. IEEE Computer Society Press, Los Alamitos (2005)
13. Sobociński, P., Sassone, V.: A congruence for Petri nets. In: PNGT '06. ENTCS, vol. 127 (2), pp. 107–120. Elsevier, Amsterdam (2004)

Applications of Metric Coinduction

Dexter Kozen[1] and Nicholas Ruozzi[2]

[1] Computer Science, Cornell University, Ithaca, NY 14853-7501, USA
kozen@cs.cornell.edu
[2] Computer Science, Yale University, New Haven, CT 06520-8285, USA
Nicholas.Ruozzi@yale.edu

Abstract. Metric coinduction is a form of coinduction that can be used to establish properties of objects constructed as a limit of finite approximations. One can prove a coinduction step showing that some property is preserved by one step of the approximation process, then automatically infer by the coinduction principle that the property holds of the limit object. This can often be used to avoid complicated analytic arguments involving limits and convergence, replacing them with simpler algebraic arguments. This paper examines the application of this principle in a variety of areas, including infinite streams, Markov chains, Markov decision processes, and non-well-founded sets. These results point to the usefulness of coinduction as a general proof technique.

1 Introduction

Mathematical induction is firmly entrenched as a fundamental and ubiquitous proof principle for proving properties of inductively defined objects. Mathematics and computer science abound with such objects, and mathematical induction is certainly one of the most important tools, if not the most important, at our disposal.

Perhaps less well entrenched is the notion of coinduction. Despite recent interest, coinduction is still not fully established in our collective mathematical consciousness. A contributing factor is that coinduction is often presented in a relatively restricted form. Coinduction is often considered synonymous with bisimulation and is used to establish equality or other relations on infinite data objects such as streams [1] or recursive types [2].

In reality, coinduction is far more general. For example, it has been recently been observed [3] that coinductive reasoning can be used to avoid complicated ε-δ arguments involving the limiting behavior of a stochastic process, replacing them with simpler algebraic arguments that establish a *coinduction hypothesis* as an invariant of the process, then automatically deriving the property in the limit by application of a coinduction principle. The notion of bisimulation is a special case of this: establishing that a certain relation is a bisimulation is tantamount to showing that a certain coinduction hypothesis is an invariant of some process.

Coinduction, as a proof principle, can handle properties other than equality and inequality and extends to other domains. The goal of this paper is to explore

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 327–341, 2007.
© Springer-Verlag Berlin Heidelberg 2007

some of these applications. We focus on four areas: infinite streams, Markov chains, Markov decision processes, and non-well-founded sets. In Section 2, we present the metric coinduction principle. In Section 3, we illustrate the use of the principle in the context of infinite streams as an alternative to traditional methods involving bisimulation. In Sections 4 and 5, we rederive some basic results of the theories of Markov chains and Markov decision processes, showing how metric coinduction can simplify arguments. Finally, in Section 6, we use metric coinduction to derive a new characterization of the hereditarily finite non-well-founded sets.

2 Coinduction in Complete Metric Spaces

2.1 Contractive Maps and Fixpoints

Let (V, d) be a complete metric space. A function $H : V \to V$ is *contractive* if there exists $0 \leq c < 1$ such that for all $u, v \in V$, $d(H(u), H(v)) \leq c \cdot d(u, v)$. The value c is called the *constant of contraction*. A continuous function H is said to be *eventually contractive* if H^n is contractive for some $n \geq 1$. Contractive maps are uniformly continuous, and by the Banach fixpoint theorem, any such map has a unique fixpoint in V.

The fixpoint of a contractive map H can be constructed explicitly as the limit of a Cauchy sequence $u, H(u), H^2(u), \ldots$ starting at any point $u \in V$. The sequence is Cauchy; one can show by elementary arguments that

$$d(H^{n+m}(u), H^n(u)) \leq c^n(1 - c^m)(1 - c)^{-1} \cdot d(H(u), u).$$

Since V is complete, the sequence has a limit u^*, which by continuity must be a fixpoint of H. Moreover, u^* is unique: if $H(u) = u$ and $H(v) = v$, then

$$d(u, v) = d(H(u), H(v)) \leq c \cdot d(u, v) \Rightarrow d(u, v) = 0,$$

therefore $u = v$.

Eventually contractive maps also have unique fixpoints. If H^n is contractive, let u^* be the unique fixpoint of H^n. Then $H(u^*)$ is also a fixpoint of H^n. But then $d(u^*, H(u^*)) = d(H^n(u^*), H^{n+1}(u^*)) \leq c \cdot d(u^*, H(u^*))$, which implies that u^* is also a fixpoint of H.

2.2 The Coinduction Rule

In the applications we will consider, the coinduction rule takes the following simple form: If φ is a closed nonempty subset of a complete metric space V, and if H is an eventually contractive map on V that preserves φ, then the unique fixpoint u^* of H is in φ. Expressed as a proof rule, this says for φ a closed property,

$$\frac{\exists u\ \varphi(u) \qquad \forall u\ \varphi(u) \Rightarrow \varphi(H(u))}{\varphi(u^*)} \tag{1}$$

In [3], the rule was used in the special form in which V was a Banach space (normed linear space) and H was an eventually contractive linear affine map on V.

2.3 Why Is This Coinduction?

We have called (1) a coinduction rule. To justify this terminology, we must exhibit a category of coalgebras and show that the rule (1) is equivalent to the assertion that a certain coalgebra is final in the category. This construction was given in [3], but we repeat it here for completeness.

Say we have a contractive map H on a metric space V and a nonempty closed subset $\varphi \subseteq V$ preserved by H. Define $H(\varphi) = \{H(s) \mid s \in \varphi\}$. Consider the category C whose objects are the nonempty closed subsets of V and whose arrows are the reverse set inclusions; thus there is a unique arrow $\varphi_1 \to \varphi_2$ iff $\varphi_1 \supseteq \varphi_2$. The map $\bar{H}$ defined by $\bar{H}(\varphi) = \mathrm{cl}(H(\varphi))$, where cl denotes closure in the metric topology, is an endofunctor on C, since $\bar{H}(\varphi)$ is a nonempty closed set, and $\varphi_1 \supseteq \varphi_2$ implies $\bar{H}(\varphi_1) \supseteq \bar{H}(\varphi_2)$. An $\bar{H}$-coalgebra is then a nonempty closed set φ such that $\varphi \supseteq \bar{H}(\varphi)$; equivalently, such that $\varphi \supseteq H(\varphi)$. The final coalgebra is $\{u^*\}$, where u^* is the unique fixpoint of H. The coinduction rule (1) says that $\varphi \supseteq H(\varphi) \Rightarrow \varphi \supseteq \{u^*\}$, which is equivalent to the statement that $\{u^*\}$ is final in the category of $\bar{H}$-coalgebras.

3 Streams

Infinite streams have been a very successful source of application of coinductive techniques. The space $\mathcal{S}_\Sigma = (\Sigma^\omega, \mathsf{head}, \mathsf{tail})$ of infinite streams over Σ is the final coalgebra in the category of *simple transition systems* over Σ, whose objects are $(X, \mathsf{obs}, \mathsf{cont})$, where X is a set, $\mathsf{obs} : X \to \Sigma$ gives an *observation* at each state, and $\mathsf{cont} : X \to X$ gives a *continuation* (next state) for each state. The unique morphism $(X, \mathsf{obs}, \mathsf{cont}) \to (\Sigma^\omega, \mathsf{head}, \mathsf{tail})$ maps a state $s \in X$ to the stream $\mathsf{obs}(s), \mathsf{obs}(\mathsf{cont}(s)), \mathsf{obs}(\mathsf{cont}^2(s)), \ldots \in \Sigma^\omega$.

We begin by illustrating the use of the metric coinduction principle in this context as an alternative to traditional methods involving bisimulation. It is well known that $\mathcal{S}_\Sigma$ forms a complete metric space under the distance function $d(\sigma, \tau) \stackrel{\mathrm{def}}{=} 2^{-n}$, where n is the first position at which σ and τ differ. The metric d satisfies the property

$$d(x :: \sigma, y :: \tau) = \begin{cases} \frac{1}{2} d(\sigma, \tau) & x = y \\ 1 & x \neq y. \end{cases}$$

One can also form the product space $\mathcal{S}_\Sigma^2$ with metric

$$d((\sigma_1, \sigma_2), (\tau_1, \tau_2)) \stackrel{\mathrm{def}}{=} \max d(\sigma_1, \tau_1),\ d(\sigma_2, \tau_2).$$

Since distances are bounded, the spaces of continuous operators $\mathcal{S}_\Sigma^2 \to \mathcal{S}_\Sigma$ and $\mathcal{S}_\Sigma \to \mathcal{S}_\Sigma^2$ are also complete metric spaces under the sup metric

$$d(E, F) \stackrel{\mathrm{def}}{=} \sup_x d(E(x), F(x)).$$

Consider the operators $\mathsf{merge} : \mathcal{S}_\Sigma^2 \to \mathcal{S}_\Sigma$ and $\mathsf{split} : \mathcal{S}_\Sigma \to \mathcal{S}_\Sigma^2$ defined informally by

$$\mathsf{merge}\,(a_0a_1a_2\cdots,\ b_0b_1b_2\cdots) = a_0b_0a_1b_1a_2b_2\cdots$$
$$\mathsf{split}\,(a_0a_1a_2\cdots) = (a_0a_2a_4\cdots,\ a_1a_3a_5\cdots).$$

Thus merge forms a single stream from two streams by taking elements alternately, and split separates a single stream into two streams consisting of the even and odd elements, respectively.

Formally, one would define merge and split coinductively as follows:

$$\mathsf{merge}\,(x :: \sigma,\ \tau) \stackrel{\text{def}}{=} x :: \mathsf{merge}\,(\tau,\ \sigma)$$
$$\mathsf{split}\,(x :: y :: \sigma) \stackrel{\text{def}}{=} (x :: \mathsf{split}\,(\sigma)_1,\ y :: \mathsf{split}\,(\sigma)_2)$$

These functions exist and are unique, since they are the unique fixpoints of the eventually contractive maps

$$\alpha : (\mathcal{S}_\Sigma^2 \to \mathcal{S}_\Sigma) \ \to\ (\mathcal{S}_\Sigma^2 \to \mathcal{S}_\Sigma) \qquad \beta : (\mathcal{S}_\Sigma \to \mathcal{S}_\Sigma^2) \ \to\ (\mathcal{S}_\Sigma \to \mathcal{S}_\Sigma^2)$$

defined by

$$\alpha(M)(x :: \sigma,\ \tau) \stackrel{\text{def}}{=} x :: M(\tau,\ \sigma)$$
$$\beta(S)(x :: y :: \sigma) \stackrel{\text{def}}{=} (x :: S(\sigma)_1,\ y :: S(\sigma)_2).$$

We would like to show that merge and split are inverses. Traditionally, one would do this by exhibiting a bisimulation between $\mathsf{merge}\,(\mathsf{split}\,(\sigma))$ and σ, thus concluding that $\mathsf{merge}\,(\mathsf{split}\,(\sigma)) = \sigma$, and another bisimulation between $\mathsf{split}\,(\mathsf{merge}\,(\sigma,\ \tau))$ and $(\sigma,\ \tau)$, thus concluding that $\mathsf{split}\,(\mathsf{merge}\,(\sigma,\ \tau)) = (\sigma,\ \tau)$.

Here is how we would prove this result using the metric coinduction rule (1). Let $M : \mathcal{S}_\Sigma^2 \to \mathcal{S}_\Sigma$ and $S : \mathcal{S}_\Sigma \to \mathcal{S}_\Sigma^2$. If M is a left inverse of S, then $\alpha^2(M)$ is a left inverse of $\beta(S)$:

$$\begin{aligned}
\alpha^2(M)(\beta(S)(x :: y :: \sigma)) &= \alpha(\alpha(M))(x :: S(\sigma)_1,\ y :: S(\sigma)_2) \\
&= x :: \alpha(M)(y :: S(\sigma)_2,\ S(\sigma)_1) \\
&= x :: y :: M(S(\sigma)_1,\ S(\sigma)_2) \\
&= x :: y :: M(S(\sigma)) \\
&= x :: y :: \sigma.
\end{aligned}$$

Similarly, if M is a right inverse of S, then $\alpha^2(M)$ is a right inverse of $\beta(S)$:

$$\begin{aligned}
\beta(S)(\alpha^2(M)(x :: \sigma,\ y :: \tau)) &= \beta(S)(\alpha(\alpha(M))(x :: \sigma,\ y :: \tau)) \\
&= \beta(S)(x :: \alpha(M)(y :: \tau,\ \sigma)) \\
&= \beta(S)(x :: y :: M(\sigma,\ \tau)) \\
&= (x :: S(M(\sigma,\ \tau))_1,\ y :: S(M(\sigma,\ \tau))_2) \\
&= (x :: (\sigma,\ \tau)_1,\ y :: (\sigma,\ \tau)_2) \\
&= (x :: \sigma,\ y :: \tau).
\end{aligned}$$

We conclude that if M and S are inverses, then so are $\alpha^2(M)$ and $\beta(S)$.

The property

$$\varphi(M,S) \stackrel{\text{def}}{\Longleftrightarrow} M \text{ and } S \text{ are inverses} \tag{2}$$

is a nonempty closed property of $(\mathcal{S}_\Sigma^2 \to \mathcal{S}_\Sigma) \times (\mathcal{S}_\Sigma \to \mathcal{S}_\Sigma^2)$ which, as we have just shown, is preserved by the contractive map $(M,S) \mapsto (\alpha^2(M), \beta(S))$. By (1), φ holds of the unique fixpoint (merge, split).

That φ is nonempty and closed requires an argument, but these conditions typically follow from general topological considerations. For example, (2) is nonempty because the spaces $\mathcal{S}_\Sigma$ and $\mathcal{S}_\Sigma^2$ are both homeomorphic to the topological product of countably many copies of the discrete space Σ.

4 Markov Chains

A finite Markov chain is a finite state space, say $\{1,\ldots,n\}$, together with a stochastic matrix $P \in \mathbb{R}^{n\times n}$ of transition probabilities, with P_{st} representing the probability of a transition from state s to state t in one step. The value P_{st}^m is the probability that the system is in state t after m steps, starting in state s.

A fundamental result in the theory of Markov chains is that if P is irreducible and aperiodic (definitions given below), then P_{st}^m tends to $1/\mu_t$ as $m \to \infty$, where μ_t is the *mean first recurrence time* of state t, the expected time of first reentry into state t after leaving state t. Intuitively, if we expect to be in state t about every μ_t steps, then in the long run we expect to be in state t about $1/\mu_t$ of the time.

The proof of this result as given in Feller [4] is rather lengthy, involving a complicated argument to establish the uniform convergence of a certain countable sequence of countable sequences. The complete proof runs to several pages. Introductory texts devote entire chapters to it (e.g. [5]) or omit the proof entirely (e.g. [6]). In this section we show that, assuming some basic spectral properties of stochastic matrices, the coinduction rule can be used to give a simpler alternative proof.

4.1 Spectral Properties

Recall that P is *irreducible* if its underlying support graph is strongly connected. The *support graph* has vertices $\{1,\ldots,n\}$ and directed edges $\{(s,t) \mid P_{st} > 0\}$. A directed graph is *strongly connected* if there is a directed path from any vertex to any other vertex. The matrix P is *aperiodic* if in addition, the gcd of the set $\{m \mid P_{ss}^m > 0\}$ is 1 for all states s. By the Perron–Frobenius theorem (see [7,8]), if P is irreducible and aperiodic, then P has eigenvalue 1 with multiplicity 1 and all other eigenvalues have norm strictly less than 1.

The matrix P is itself not contractive, since 1 is an eigenvalue. However, consider the matrix

$$P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}},$$

where $\mathbf{1}$ is the column vector of all 1's and $^{\mathrm{T}}$ denotes matrix transpose. The matrix $\frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$ is the $n \times n$ matrix all of whose entries are $1/n$.

The spectra of P and $P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$ are closely related, as shown in the following lemma.

Lemma 1. *Let $P \in \mathbb{R}^{n \times n}$ be a stochastic matrix. Any (left) eigenvector x^{T} of $P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$ that lies in the hyperplane $x^{\mathrm{T}}\mathbf{1} = 0$ is also an eigenvector of P with the same eigenvalue, and vice-versa. The only other eigenvalue of P is 1 and the only other eigenvalue of $P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$ is 0.*

Proof. For any eigenvalue λ of P and corresponding eigenvector x^{T},

$$\lambda x^{\mathrm{T}}\mathbf{1} = x^{\mathrm{T}}P\mathbf{1} = x^{\mathrm{T}}\mathbf{1}$$

since $P\mathbf{1} = \mathbf{1}$, so either $\lambda = 1$ or $x^{\mathrm{T}}\mathbf{1} = 0$. Similarly, for any eigenvalue λ of $P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$ and corresponding eigenvector x^{T},

$$\lambda x^{\mathrm{T}}\mathbf{1} = x^{\mathrm{T}}(P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}})\mathbf{1} = x^{\mathrm{T}}\mathbf{1} - x^{\mathrm{T}}\mathbf{1} = 0,$$

so either $\lambda = 0$ or $x^{\mathrm{T}}\mathbf{1} = 0$. But if $x^{\mathrm{T}}\mathbf{1} = 0$, then

$$x^{\mathrm{T}}(P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}) = x^{\mathrm{T}}P - \frac{1}{n}x^{\mathrm{T}}\mathbf{1}\mathbf{1}^{\mathrm{T}} = x^{\mathrm{T}}P,$$

so in this case x^{T} is an eigenvector of P iff it is an eigenvector of $P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$ with the same eigenvalue.

4.2 Coinduction and the Convergence of P^m

If P is irreducible and aperiodic, then $P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$ is eventually contractive, since $\inf_n \sqrt[n]{\|(P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}})^n\|}$ is equal to the *spectral radius* or norm of the largest eigenvalue of $P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$ (see [9]), which by Lemma 1 is less than 1. Thus the map

$$x^{\mathrm{T}} \mapsto x^{\mathrm{T}}(P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}) + \frac{1}{n}\mathbf{1}^{\mathrm{T}} \tag{3}$$

is of the proper form to be used with the metric coinduction rule (1) to establish the convergence of P^m.

Since $P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}$ is eventually contractive, the map (3) has a unique fixpoint u^{T}. The set of stochastic vectors

$$S = \{x^{\mathrm{T}} \mid x^{\mathrm{T}} \geq 0,\ x^{\mathrm{T}}\mathbf{1} = 1\}$$

is closed and preserved by the map (3), since

$$x^{\mathrm{T}}\mathbf{1} = 1 \Rightarrow x^{\mathrm{T}}(P - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathrm{T}}) + \frac{1}{n}\mathbf{1}^{\mathrm{T}} = x^{\mathrm{T}}P,$$

and S is preserved by P. By the metric coinduction rule (1), the unique fixpoint u^{T} is contained in S. By Lemma 1, it is also an eigenvector of 1, and $y^{\mathrm{T}}P^m$ tends to u^{T} for any $y^{\mathrm{T}} \in S$. Applying this to the rows of any stochastic matrix E, we have that EP^m converges to the matrix $\mathbf{1}u^{\mathrm{T}}$.

4.3 Recurrence Statistics

Once we have established the convergence of P^m, we can give a much shorter argument than those of [4,5] that the actual limit of P^m_{st} is $1/\mu_t$. We follow the notation of [4].

Fix a state t, and let $\mu = \mu_t$. Let f_m be the probability that after leaving state t, the system first returns to state t at time m. Let $u_m = P^m_{tt}$ be the probability that the system is in state t at time m after starting in state t. By irreducibility, $\sum_{m=1}^{\infty} f_m = 1$ and $\mu = \sum_{m=1}^{\infty} m f_m < \infty$. Let $\rho_m \stackrel{\text{def}}{=} \sum_{k=m+1}^{\infty} f_k$, and consider the generating functions

$$f(x) \stackrel{\text{def}}{=} \sum_{m=1}^{\infty} f_m x^m \qquad u(x) \stackrel{\text{def}}{=} \sum_{m=0}^{\infty} u_m x^m$$

$$\rho(x) \stackrel{\text{def}}{=} \sum_{m=0}^{\infty} \rho_m x^m \qquad \sigma(x) \stackrel{\text{def}}{=} u_0 + \sum_{m=0}^{\infty} (u_{m+1} - u_m) x^{m+1}.$$

The probabilities u_n obey the recurrence

$$u_0 = 1 \qquad u_n = \sum_{m=0}^{n-1} u_m f_{n-m},$$

which implies that $f(x)u(x) = u(x) - 1$. Elementary algebraic reasoning gives

$$\sigma(x)\rho(x) = 1. \tag{4}$$

Now we claim that both $\sigma(1)$ and $\rho(1)$ converge. The sequence $\rho(1)$ converges to $\mu > 0$, since

$$\rho(1) = \sum_{m=1}^{\infty} \rho_m = \sum_{m=1}^{\infty} m f_m = \mu, \tag{5}$$

and the latter sequence in (5) converges absolutely. For $\sigma(1)$, we have

$$\sigma(1) = u_0 + \sum_{m=0}^{\infty} (u_{m+1} - u_m),$$

which converges by the results of Section 4.2. By (4), $\sigma(1)\rho(1) = 1$, therefore $\sigma(1) = 1/\mu$. But the mth partial sum of $\sigma(1)$ is just $u_0 + \sum_{k=0}^{m-1} (u_{k+1} - u_k) = u_m$, so the sequence u_m converges to $1/\mu$.

5 Markov Decision Processes

In this section, we rederive some fundamental results on Markov decision processes using the metric coinduction principle. A fairly general treatment of this theory is given in [10], and we follow the notation of that paper. However, the strategic use of metric coinduction allows a more streamlined presentation.

5.1 Existence of Optimal Strategies

Let V be the space of bounded real-valued functions on a set of states Ω with the sup norm $\|v\| \stackrel{\text{def}}{=} \sup_{x\in\Omega} |v(x)|$. The space V is complete metric space with metric $\|v-u\|$.

For each state $x \in \Omega$, say we have a set Δ_x of *actions*. A *deterministic strategy* is an element of $\Delta \stackrel{\text{def}}{=} \prod_{x\in\Omega} \Delta_x$, thus a selection of actions, one for each state $x \in \Omega$. More generally, if Δ_x is a measurable space, let $\mathcal{M}(\Delta_x)$ denote the space of probability measures on Δ_x. A *probabilistic strategy* is an element of $\prod_{x\in\Omega} \mathcal{M}(\Delta_x)$, thus a selection of probability measures, one for each $x \in \Omega$. A deterministic strategy can be viewed as a probabilistic strategy in which all the measures are point masses.

Now suppose we have a *utility function* $h : \prod_{x\in\Omega}(\Delta_x \to V \to \mathbb{R})$ with the three properties listed below. The function h induces a function H such that $H_\delta(u)(x) = h(x, \delta_x, u) \in \mathbb{R}$, where $x \in \Omega$, $\delta \in \Delta$, and $u \in V$.

(i) The function H is uniformly bounded as a function of δ and x. That is, $H_\delta : V \to V$, and for any fixed $u \in V$, $\sup_{\delta\in\Delta} \|H_\delta(u)\|$ is finite.
(ii) The functions H_δ are uniformly contractive with constant of contraction $c < 1$. That is, for all $\delta \in \Delta$ and $u, v \in V$, $\|H_\delta(v) - H_\delta(u)\| \le c \cdot \|v-u\|$. Thus H_δ has a unique fixpoint, which we denote by v_δ.
(iii) Every H_δ is *monotone*: if $u \le v$, then $H_\delta(u) \le H_\delta(v)$. The order $\le$ on V is the pointwise order.

Lemma 2. *Define $A : V \to V$ by $A(u)(x) \stackrel{\text{def}}{=} \sup_{d\in\Delta_x} h(x, d, u)$. The supremum exists since the H_δ are uniformly bounded. Then A is contractive with constant of contraction c.*

Proof. Let $\varepsilon > 0$. For $x \in \Omega$, assuming without loss of generality that $A(v)(x) \ge A(u)(x)$,

$$\begin{aligned}
&|A(v)(x) - A(u)(x)| \\
&= \sup_{d\in\Delta_x} h(x,d,v) - \sup_{e\in\Delta_x} h(x,e,u) \\
&\le \varepsilon + h(x,d,v) - \sup_{e\in\Delta_x} h(x,e,u) \qquad \text{for suitably chosen } d \in \Delta_x \\
&\le \varepsilon + h(x,d,v) - h(x,d,u) \\
&\le \varepsilon + c \cdot \|v-u\|.
\end{aligned}$$

Since ε was arbitrary, $|A(v)(x) - A(u)(x)| \le c \cdot \|v-u\|$, thus

$$\|A(v) - A(u)\| \le \sup_x |A(v)(x) - A(u)(x)| \le c \cdot \|v-u\|.$$

Since A is contractive, it has a unique fixpoint v^*.

Lemma 3. *For any δ, $v_\delta \le v^*$.*

Proof. By the coinduction principle, it suffices to show that $u \le v$ implies $H_\delta(u) \le A(v)$. Here the metric space is V^2, the closed property φ is $u \le v$, and the contractive map is (H_δ, A). But if $u \le v$, then by monotonicity,

$$H_\delta(u)(x) \le H_\delta(v)(x) = h(x, \delta_x, v) \le \sup_{d \in \Delta_x} h(x, d, v) = A(v).$$

Lemma 4. *v^* can be approximated arbitrarily closely by v_δ for deterministic strategies δ.*

Proof. Let $\varepsilon > 0$. Let δ be such that for all x,

$$\sup_{d \in \Delta_x} h(x, d, v^*) - h(x, \delta_x, v^*) < (1 - c)\varepsilon.$$

We will show that $\|v^* - v_\delta\| \le \varepsilon$. By the coinduction rule (1), it suffices to show that $\|v^* - u\| \le \varepsilon$ implies $\|v^* - H_\delta(u)\| \le \varepsilon$. Here the metric space is V, the closed property $\varphi(u)$ is $\|v^* - u\| \le \varepsilon$, and the contractive map is H_δ. But if $\|v^* - u\| \le \varepsilon$,

$$\begin{aligned}
\|v^* - H_\delta(u)\| &= \sup_x |v^*(x) - H_\delta(u)(x)| = \sup_x |A(v^*)(x) - H_\delta(u)(x)| \\
&= \sup_x |\sup_{d \in \Delta_x} h(x, d, v^*) - h(x, \delta_x, u)| \\
&\le \sup_x (|\sup_{d \in \Delta_x} h(x, d, v^*) - h(x, \delta_x, v^*)| + |h(x, \delta_x, v^*) - h(x, \delta_x, u)|) \\
&\le (1 - c)\varepsilon + c \cdot \|v^* - u\| \le (1 - c)\varepsilon + c\varepsilon = \varepsilon.
\end{aligned}$$

5.2 Probabilistic Strategies

We use the metric coinduction rule (1) to prove the well-known result that for Markov decision processes, probabilistic strategies are no better than deterministic strategies. If $\sup_{d \in \Delta_x} h(x, d, v^*)$ is attainable for all x, then the deterministic strategy $\delta_x \stackrel{\text{def}}{=} \operatorname{argmax}_{d \in \Delta_x} h(x, d, v^*)$ is optimal, even allowing probabilistic strategies. However, if $\sup_{d \in \Delta_x} h(x, d, v^*)$ is not attainable, then it is not so obvious what to do.

For this argument, we assume that Δ_x is a measurable space and that for all fixed x and u, $h(x, d, u)$ is an integrable function of $d \in \Delta_x$. Given a probabilistic strategy $\mu : \prod_{x \in \Omega} \mathcal{M}(\Delta_x)$, the one-step utility function is $H_\mu : V \to V$ defined by the Lebesgue integral

$$H_\mu(u)(x) \stackrel{\text{def}}{=} \int_{d \in \Delta_x} h(x, d, u) \cdot \mu_x(\triangle d).$$

This integral accumulates the various individual payoffs over all choices of d weighted by the measure μ_x.

The map $H_\mu(u)$ is uniformly bounded in μ, since

$$\begin{aligned}
\|H_\mu(u)\| = \sup_x \left| \int_{d \in \Delta_x} h(x, d, u) \cdot \mu_x(\triangle d) \right| &\le \sup_x \int_{d \in \Delta_x} |h(x, d, u)| \cdot \mu_x(\triangle d) \\
&\le \sup_x \sup_d |h(x, d, u)| \cdot \int_{d \in \Delta_x} \mu_x(\triangle d) = \sup_{x,d} |h(x, d, u)|.
\end{aligned}$$

It is also a contractive map with constant of contraction c, since

$$\begin{aligned}
\|H_\mu(v) - H_\mu(u)\| &= \sup_x |H_\mu(v)(x) - H_\mu(u)(x)| \\
&= \sup_x \left| \int_{d \in \Delta_x} h(x,d,v) \cdot \mu_x(\triangle d) - \int_{d \in \Delta_x} h(x,d,u) \cdot \mu_x(\triangle d) \right| \\
&= \sup_x \left| \int_{d \in \Delta_x} (h(x,d,v) - h(x,d,u)) \cdot \mu_x(\triangle d) \right| \\
&\le \sup_x \int_{d \in \Delta_x} |h(x,d,v) - h(x,d,u)| \cdot \mu_x(\triangle d) \\
&\le \sup_x \int_{d \in \Delta_x} c \cdot \|v - u\| \cdot \mu_x(\triangle d) \\
&= c \cdot \|v - u\| \cdot \sup_x \int_{d \in \Delta_x} \mu_x(\triangle d) \\
&= c \cdot \|v - u\|.
\end{aligned}$$

Since it is a contractive map, it has a unique fixpoint v_μ.

Now take any deterministic strategy δ such that $h(x, \delta_x, v_\mu) \ge v_\mu(x)$ for all x. This is always possible, since if $h(x, d, v_\mu) < v_\mu(x)$ for all $d \in \Delta_x$, then

$$v_\mu(x) = H_\mu(v_\mu)(x) = \int_{d \in \Delta_x} h(x,d,v_\mu) \cdot \mu_x(\triangle d) < v_\mu(x),$$

a contradiction. The following lemma says that the deterministic strategy δ is no worse than the probabilistic strategy μ.

Lemma 5. $v_\delta \ge v_\mu$.

Proof. Assuming $v_\mu \le v$, we have

$$v_\mu(x) \le h(x, \delta_x, v_\mu) \le h(x, \delta_x, v) = H_\delta(v)(x),$$

the second inequality by monotonicity. As x was arbitrary, $v_\mu \le H_\delta(v)$. The result follows from the coinduction principle on the metric space V with $\varphi(v)$ the closed property $v_\mu \le v$ and contractive map H_δ.

6 Non-well-Founded Sets[1]

In classical Zermelo–Fraenkel set theory with choice (ZFC), the "element of" relation $\in$ is well-founded, as guaranteed by the axiom of foundation. Aczel [11] developed the theory of *non-well-founded sets*, in which sets with infinitely descending $\in$-chains are permitted in addition to the well-founded sets. These are precisely the sets that are explicitly ruled out of existence by the axiom of foundation.

[1] Proofs have been omitted from this section due to page limitations. A longer version with all omitted proofs is available from the authors.

In the theory of non-well-founded sets, the sets are represented by *accessible pointed graphs* (APGs). An APG is a directed graph with a distinguished node such that every node is reachable by a directed path from the distinguished node. Two APGs represent the same set iff they are bisimilar. The APGs of well-founded sets may be infinite, but may contain no infinite paths or cycles, whereas the APGs of non-well-founded sets may contain infinite paths and cycles. Equality as bisimulation is the natural analog of extensionality in ZFC; essentially, two APGs are declared equal as sets if there is no witness among their descendants that forces them not to be. The class V is the class of sets defined in this way.

Aczel [11] and Barwise and Moss [12] note the strong role that coinduction plays in this theory. Since equality between APGs is defined in terms of bisimulation, coinduction becomes a primary proof technique for establishing the equivalence of different APGs representing the same set.

In attempting to define a metric on non-well-founded sets, the classical Hausdorff distance suggests itself as a promising candidate. There are two complications. One is that we must apply the definition coinductively. Another is that ordinarily, the Hausdorff metric is only defined on compact sets, since otherwise a Hausdorff distance of zero may not imply equality, and that is the case here. However, the definition still makes sense even for non-compact sets and leads to further insights into the structure of non-well-founded sets.

In this section, we define a distance function $d : V^2 \to \mathbb{R}$ based on a coinductive application of the Hausdorff distance function and derive some properties of d. We show that (V, d) forms a compact pseudometric space. Being a pseudometric instead of a metric means that there are sets $s \neq t$ with $d(s, t) = 0$. Nevertheless, we identify a maximal family of sets that includes all the hereditarily finite sets on which d acts as a metric.

The following are our results. Define $s \approx t$ if $d(s, t) = 0$. Call a set s *singular* if the only t such that $s \approx t$ is s itself.

- A set is singular if and only if it is hereditarily finite.
- All singular sets are closed in the pseudometric topology. In particular, all hereditarily finite sets are hereditarily closed (but not vice-versa).
- A set is hereditarily closed if and only if it is closed and all elements are singular.
- All hereditarily closed sets are canonical (but not vice-versa), where a set is *canonical* if it is a member of a certain coinductively-defined class of canonical representatives of the $\approx$-classes.
- The map d is a metric on the canonical sets; moreover, the canonical sets are a maximal class for which this is true.

6.1 Coinductive Definition of Functions

Just as classical ZFC allows the definition of functions by induction over ordinary well-founded sets, there is a corresponding principle for non-well-founded sets. For any function $H : V \to V$, the equation

$$G(s) \stackrel{\text{def}}{=} \{G(u) \mid u \in H(s)\} \tag{6}$$

determines $G : V \to V$ uniquely. This is because if G and G' both satisfy (6), then the relation

$$u \; R \; v \stackrel{\text{def}}{\iff} \exists s \; u = G(s) \wedge v = G'(s)$$

is a bisimulation, therefore $G(s) = G'(s)$ for all s. In coalgebraic terms, the map G is the unique morphism from the coalgebra $(V, \{(s,t) \mid s \in H(t)\})$ to the final coalgebra $(V, \in)$; see [11, Chp. 7].

6.2 Definition of d

Let B be the Banach space of bounded real-valued functions $g : \mathrm{APG}^2 \to \mathbb{R}$ with norm

$$\|g\| \stackrel{\text{def}}{=} \sup_{s,t} |g(s,t)|.$$

Define the map $\tau : B \to B$ by

$$\tau(g)(s,t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } s,t = \varnothing \\ 1 & \text{if } s = \varnothing \Leftrightarrow t \neq \varnothing \\ \frac{1}{2} \max \begin{cases} \sup_{u \in s} \inf_{v \in t} g(u,v) \\ \sup_{v \in t} \inf_{u \in s} g(u,v) \end{cases} & \text{if } s,t \neq \varnothing. \end{cases}$$

It can be shown that $\|\tau(g) - \tau(g')\| \le \frac{1}{2}\|g - g'\|$, thus τ is contractive on B with constant of contraction $1/2$ and has a unique fixpoint $d \in B$. One can therefore use the metric coinduction rule (1) to prove properties of d.

We can show that the non-well-founded sets V form a compact (thus complete) pseudometric space under the distance function d. At the outset, it is not immediately clear that d is well-defined on V. We must argue that d is invariant on bisimulation classes; that is, for any bisimulation R, if $s \; R \; s'$ and $t \; R \; t'$, then $d(s,t) = d(s',t')$. We can use the metric coinduction rule (1) to prove this.

6.3 Canonical Sets

The map d is only a pseudometric and not a metric, since it is possible that $d(s,t) = 0$ even though $s \neq t$. For example, define $\bar{0} = \varnothing$, $\overline{n+1} = \{\bar{n}\}$. Let Ω be the unique non-well-founded set such that $\Omega = \{\Omega\}$. The sets $\{\bar{n} \mid n \ge 0\}$ and $\{\bar{n} \mid n \ge 0\} \cup \Omega$ are distinct, but distance 0 apart (Fig. 1). Nevertheless, it is possible to relate this map to the coalgebraic structure of V.

The map d defines a pseudometric topology with basic open neighborhoods $\{t \mid d(s,t) < \varepsilon\}$ for each set s and $\varepsilon > 0$, but because d is only a pseudometric, the topology does not have nice separation properties. However, if we define $s \approx t \stackrel{\text{def}}{\iff} d(s,t) = 0$, then d is well-defined on $\approx$-equivalence classes and is a metric on the quotient space.

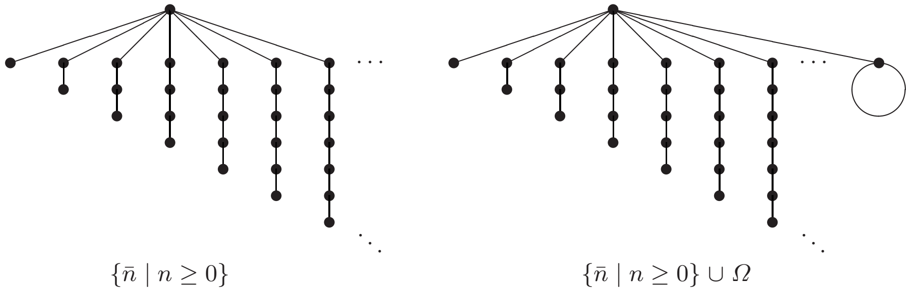


Fig. 1. Distinct sets of distance 0

More interestingly, we can identify a natural class of canonical elements, one in each $\approx$-class, such that d, restricted to canonical elements, is a metric; moreover, the canonical elements are a maximal class for which this is true. Thus the quotient space is isometric to the subspace of canonical elements. The canonical elements include all the hereditarily finite sets.

The canonical elements are defined as the images of the function $F : V \to V$, defined coinductively as follows:

$$F(s) \stackrel{\text{def}}{=} \{F(u) \mid u \in \text{cl}(s)\}, \tag{7}$$

where cl denotes closure in the pseudometric topology. The equation (7) determines F uniquely, as with (6). A set s is called *canonical* if $s = F(t)$ for some t; equivalently, by Corollary 1(ii) below, if s is a fixpoint of F.

Lemma 6. *$d(s,t) = 0$ iff $\text{cl}(s) = \text{cl}(t)$.*

Theorem 1

(i) *If $d(s,t) = 0$, then $F(s) = F(t)$.*
(ii) *For all s, $d(s, F(s)) = 0$; that is, $s \approx F(s)$.*

Corollary 1

(i) *$d(s,t) = 0$ iff $F(s) = F(t)$.*
(ii) *For all s, $F(F(s)) = F(s)$.*
(iii) *Every $\approx$-equivalence class contains exactly one canonical set, and d restricted to canonical sets is a metric. Moreover, the canonical sets are a maximal class for which this is true.*

6.4 Hereditarily Finite Sets Are Canonical

Let φ be a property of sets. We define a set to be *hereditarily* φ (Hφ) if it has an APG representation in which every node represents a set satisfying φ. Equivalently, Hφ is the largest solution of

$$\text{H}\varphi(s) \stackrel{\text{def}}{\iff} \varphi(s) \wedge \forall u \in s\ \text{H}\varphi(u).$$

The *hereditarily finite* (HF) sets are those possessing an APG representation in which every node has finite out-degree (not necessarily bounded). Note that this differs from Aczel's definition [11, p. 7]. Aczel defines a set to be hereditarily finite if it has a finite APG, which is a much stronger condition. Aczel's definition and ours coincide for well-founded sets by König's lemma, but not for non-well-founded sets in general. For example, the set $f(0)$, where f is defined coinductively by $f(n) = \{\bar{n}, f(n+1)\}$ (Fig. 2), is hereditarily finite in our sense but not Aczel's. We would prefer the term *regular* or *rational* for sets that are hereditarily finite in Aczel's sense, since they are exactly the sets that have a regular or rational tree representation [13].

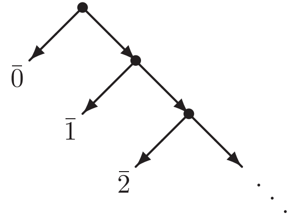


Fig. 2. $f(0)$

A set is *hereditarily closed* (HC) if it has an APG representation in which every node represents a closed set in the pseudometric topology. Recall that a set is *singular* if it forms a singleton $\approx$-class.

Theorem 2. *A set is hereditarily closed if and only if it is closed and all its elements are singular.*

Theorem 3. *A set is singular if and only if it is hereditarily finite.*

Theorem 4. *Every hereditarily finite set is heretarily closed, and every hereditarily closed set is canonical. Both implications are strict.*

7 Conclusions and Future Work

We have illustrated the use of the metric coinduction principle in four areas: infinite streams, Markov chains, Markov decision processes, and non-well-founded sets. In all these areas, metric coinduction can be used to simplify proofs or derive new insights.

Other areas are likely to be amenable to such techniques. In particular, iterated function systems seem to be a promising candidate.

Acknowledgements

Thanks to Lars Backstrom and Prakash Panangaden for valuable comments. This work was supported by NSF grant CCF-0635028. Any views and conclusions expressed herein are those of the authors and should not be interpreted as representing the official policies or endorsements of the National Science Foundation or the United States government.

References

1. Rutten, J.: Universal coalgebra: A theory of systems. Theor. Comput. Sci. 249, 3–80 (2000)
2. Fiore, M.P.: A coinduction principle for recursive data types based on bisimulation. In: Proc. 8th Conf. Logic in Computer Science (LICS'93), pp. 110–119 (1993)

3. Kozen, D.: Coinductive proof principles for stochastic processes. In: Alur, R. (ed.) Proc. 21st Symp. Logic in Computer Science (LICS'06), pp. 359–366. IEEE Computer Society Press, Los Alamitos (2006)
4. Feller, W.: An Introduction to Probability Theory and its Applications, vol. 1. Wiley, Chichester (1950)
5. Häggström, O.: Finite Markov Chains and Algorithmic Applications. Cambridge University Press, Cambridge (2002)
6. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
7. Brémaud, P.: Markov Chains, Gibbs Fields, Monte Carlo Simulation and Queues. Texts in Applied Mathematics. Springer, Heidelberg (1999)
8. Minc, H.: Nonnegative Matrices. John Wiley, Chichester (1988)
9. Dunford, N., Schwartz, J.T.: Linear Operators: Part I: General Theory. John Wiley, Chichester (1957)
10. Denardo, E.V.: Contraction mappings in the theory underlying dynamic programming. SIAM Review 9(2), 165–177 (1967)
11. Aczel, P.: Non-Well-Founded Sets. CSLI Lecture Notes, vol. 14. Stanford University (1988)
12. Barwise, J., Moss, L.: Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena. CSLI Lecture Notes, vol. 60. Center for the Study of Language and Information (CSLI), Stanford University (1996)
13. Courcelle, B.: Fundamental properties of infinite trees. Theor. Comput. Sci. 25, 95–169 (1983)
14. Isaacson, D., Madsen, R.: Markov Chains: Theory and Applications. John Wiley and Sons, Chichester (1976)

The Goldblatt-Thomason Theorem for Coalgebras

Alexander Kurz[1,⋆] and Jiří Rosický[2,⋆⋆]

[1] University of Leicester, UK
[2] Masaryk University, Brno, Czech Republic

Abstract. Goldblatt and Thomason's theorem on modally definable classes of Kripke frames and Venema's theorem on modally definable classes of Kripke models are generalised to coalgebras.

1 Introduction

The Goldblatt-Thomason theorem [11] states that a class of Kripke frames closed under ultrafilter extensions is modally definable if and only if it reflects ultrafilter extensions and is closed under generated subframes, homomorphic images and disjoint unions. The proof is based on the duality between Boolean algebras and sets

$$\mathsf{BA} \underset{\Sigma}{\overset{\Pi}{\leftrightarrows}} \mathsf{Set}^{\mathrm{op}} \tag{1}$$

where Π is powerset and Σ assigns to a BA the set of ultrafilters. Σ is left-adjoint to Π but, of course, this adjunction does *not* form a dual equivalence. The price we have to pay for this is that going from Set to BA and back leaves us with $\Sigma\Pi X$: If X is the carrier of a Kripke frame, then its ultrafilter extension has carrier $\Sigma\Pi X$, which explains why ultrafilter extensions appear in the theorem.

Our generalisation from Kripke frames to T-coalgebras works as follows. Set and BA are completions (with filtered colimits) of the categories Set_ω of finite sets and BA_ω of finite Boolean algebras, respectively. BA_ω and Set_ω are dually equivalent. Now, given a functor T on Set that preserves finite sets, we can restrict T to Set_ω. Via the dual equivalence $\mathsf{BA}_\omega \simeq \mathsf{Set}_\omega^{\mathrm{op}}$, this gives us a functor on BA_ω, which we can then lift to a functor $L : \mathsf{BA} \to \mathsf{BA}$.

$$\begin{array}{ccc} L \circlearrowright \mathsf{BA} & \underset{\Sigma}{\overset{\Pi}{\leftrightarrows}} & \mathsf{Set}^{\mathrm{op}} \circlearrowleft T \\ \uparrow & & \uparrow \\ \mathsf{BA}_\omega & \leftrightarrows & \mathsf{Set}_\omega^{\mathrm{op}} \end{array} \tag{2}$$

⋆ Partially supported by EPSRC EP/C014014/1.
⋆⋆ Supported by the Ministry of Education of the Czech Republic under the project 1M0545.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 342–355, 2007.
© Springer-Verlag Berlin Heidelberg 2007

[17] showed the following: (i) L has a presentation and therefore determines a logic for T-coalgebras, (ii) Π extends to a functor $\mathsf{Coalg}(T) \to \mathsf{Alg}(L)$, (iii) if T weakly preserves cofiltered limits, then Σ extends to a map on objects $\mathsf{Alg}(L) \to \mathsf{Coalg}(T)$. This note shows that the classical Goldblatt-Thomason theorem generalises to those T-coalgebras where $\Sigma : \mathsf{BA} \to \mathsf{Set}$ can be extended to a *functor* $\mathsf{Alg}(L) \to \mathsf{Coalg}(T)$.

$$\mathsf{Alg}(L) \underset{\Sigma}{\overset{\Pi}{\leftrightarrows}} \mathsf{Coalg}(T)^{\mathrm{op}} \tag{3}$$

The same argument also generalises a similar definability result for Kripke models due to Venema [22].

Related Work. An algebraic semantics for logics for coalgebras and its investigation via the adjunction between BA and Set has been given in Jacobs [13]. The idea that a logic for T-coalgebras is a functor L on BA appears in [5,15] and can be traced back to Abramsky [1,2] and Ghilardi [10]. It has been further developed in [6,16]. The general picture underlying diagram (2) has been discussed in Lawvere [19] where it is attributed to Isbell. The implications of this Isbell-conjugacy for logics for coalgebras are explained in [17]. For topological spaces, which can be seen as particular coalgebras, the Goldblatt-Thomason theorem is due to Gabelaia [9] and ten Cate et al [7].

2 Coalgebras and Their Logics

Definition 2.1. *The category* $\mathsf{Coalg}(T)$ *of coalgebras for a functor* T *on a category* $\mathcal{X}$ *has as objects arrows* $\xi : X \to TX$ *in* $\mathcal{X}$ *and morphisms* $f : (X, \xi) \to (X', \xi')$ *are arrows* $f : X \to X'$ *such that* $Tf \circ \xi = \xi' \circ f$.

Examples of functors of interest to us in this paper are described by

Definition 2.2 (gKPF). *A generalised Kripke polynomial functor (gKPF)* $T : \mathsf{Set} \to \mathsf{Set}$ *is built according to*

$$T ::= Id \mid K_C \mid T + T \mid T \times T \mid T \circ T \mid \mathcal{P} \mid \mathcal{H}$$

where Id *is the identity functor,* K_C *is the constant functor that maps all sets to a finite set* C*,* $\mathcal{P}$ *is covariant powerset and* $\mathcal{H}$ *is* $2^{2^{-}}$.

Remark 2.3. The term 'Kripke polynomial functor' was coined in Rößiger [20]. We add the functor $\mathcal{H}$. $\mathcal{H}$-coalgebras are known as neighbourhood frames in modal logic and are investigated, from a coalgebraic point of view, in Hansen and Kupke [12].

We describe logics for coalgebras by functors L on the category BA of Boolean algebras. Although this approach differs conceptually from Jacobs's [13], the equations appearing in the example below are the same as his.

Example 2.4. We describe functors $L : \mathsf{BA} \to \mathsf{BA}$ or $L : \mathsf{BA} \times \mathsf{BA} \to \mathsf{BA}$ by generators and relations as follows.

1. $L_{K_C}(A)$ is the free BA given by generators $c \in C$ and satisfying $c_1 \wedge c_2 = \bot$ for all $c_1 \neq c_2$ and $\bigvee_{c \in C} c = \top$.
2. $L_+(A_1, A_2)$ is generated by $[\kappa_1]a_1$, $[\kappa_2]a_2$, $a_i \in A_i$ where the $[\kappa_i]$ preserve finite joins and binary meets and satisfy $[\kappa_1]a_1 \wedge [\kappa_2]a_2 = \bot$, $[\kappa_1]\top \vee [\kappa_2]\top = \top$, $\neg[\kappa_1]a_1 = [\kappa_2]\top \vee [\kappa_1]\neg a_1$, $\neg[\kappa_2]a_2 = [\kappa_1]\top \vee [\kappa_2]\neg a_2$.
3. $L_\times(A_1, A_2)$ is generated by $[\pi_1]a_1$, $[\pi_2]a_2$, $a_i \in A_i$ where $[\pi_i]$ preserve Boolean operations.
4. $L_{\mathcal{P}}(A)$ is generated by $\Box a$, $a \in A$, and $\Box$ preserves finite meets.
5. $L_{\mathcal{H}}(A)$ is generated by $\Box a$, $a \in A$ (no equations).

Informally, the equations in the 2nd item are justified as follows. Take A_1, A_2 to be the collections of subsets of two sets X_1, X_2, take $[\kappa_i]a_i$ to be the direct image of the injection $\kappa_i : X_i \to X_1 + X_2$ and describe how the $[\kappa_i]$ interact with the Boolean operations, interpreting $\wedge$ as $\cap$, etc.

More formally, we recall that sets and Boolean algebras are related by two functors

$$\mathsf{BA} \underset{\Sigma}{\overset{\Pi}{\rightleftarrows}} \mathsf{Set}^{\mathrm{op}} \tag{4}$$

where Π maps a set to its powerset and Σ a Boolean algebra to the set of its ultrafilters. On arrows, both functors are given by inverse image.

The justification for the presentations is now given, in essence, by the following isomorphisms. For Boolean algebras A, A_1, A_2, we have $L_{K_C}(A) \cong \Pi C$; $L_+(A_1, A_2) \cong A_1 \times A_2$; $L_\times(A_1, A_2) \cong A_1 + A_2$. For finite sets X, we have $L_{\mathcal{P}}(\Pi X) \cong \Pi\mathcal{P}X$; $L_{\mathcal{H}}(\Pi X) \cong \Pi\mathcal{H}X$. We will make this more precise in Definition 2.6 and Proposition 2.8.

Definition 2.5 (L_T). *For each gKPF (see Definition 2.2) $T : \mathsf{Set} \to \mathsf{Set}$ we define L_T by the corresponding constructions of Example 2.4.*

Example 2.4 illustrates how (a presentation of) a functor on BA describes the syntax and proof system of a modal logic. The semantics is given by a natural transformation

$$L\Pi X \xrightarrow{\delta_X} \Pi T X, \tag{5}$$

since this is exactly what is needed to define the extension $[\![-]\!]$ of formulas via the unique morphism from the initial L-algebra $LI \to I$. In detail, given a coalgebra (X, ξ) we let $[\![-]\!]$ be as in

$$\begin{array}{ccccc} I & \longleftarrow & & & LI \\ {\scriptstyle [\![-]\!]}\downarrow & & & & \downarrow{\scriptstyle L[\![-]\!]} \\ \Pi X & \xleftarrow[\Pi\xi]{} & \Pi T X & \xleftarrow[\delta_X]{} & L\Pi X \end{array} \tag{6}$$

In our examples, for gKPFs T, we define $\delta_T : L_T\Pi \to \Pi T$ as follows.

Definition 2.6 (δ_T). *We define Boolean algebra morphisms*

1. $L_{K_C} \Pi X \to \Pi C$ *by* $c \mapsto \{c\}$,
2. $L_+(\Pi X_1, \Pi X_2) \to \Pi(X_1 + X_2)$ *by* $[\kappa_i]a_i \mapsto a_i$,
3. $L_\times(\Pi X, \Pi Y) \to \Pi(X_1 \times X_2)$ *by* $[\pi_1]a_1 \mapsto a_1 \times X_2$, $[\pi_2]a_2 \mapsto X_1 \times a_2$,
4. $L_\mathcal{P} \Pi X \to \Pi \mathcal{P} X$ *by* $\Box a \mapsto \{b \subseteq X \mid b \subseteq a\}$,
5. $L_\mathcal{H} \Pi X \to \Pi \mathcal{H} X$ *by* $\Box a \mapsto \{s \in \mathcal{H}X \mid a \in s\}$.

and extend them inductively to $\delta_T : L_T \Pi \to \Pi T$ *for all gKPF* T.

The definition exploits that BA-morphisms are determined by their action on the generators.

Example 2.7. Together with (6), item 4 and 5 of Definition 2.6 give rise to the Kripke and neighbourhood semantics of modal logic:

– For $\xi : X \to \mathcal{P}X$ and $\Box\varphi$ in the initial $L_\mathcal{P}$-algebra, we have $[\![\Box\varphi]\!] = \{x \in X \mid \xi(x) \subseteq [\![\varphi]\!]\}$;
– For $\xi : X \to \mathcal{H}X$ and $\Box\varphi$ in the initial $L_\mathcal{H}$-algebra, we have $[\![\Box\varphi]\!] = \{x \in X \mid [\![\varphi]\!] \in \xi(x)\}$.

The justification for the definition of L_T and δ_T is now given by the following proposition. It says that (L, δ) completely captures the action of T on finite X; and more can hardly be expected from a finitary logic of T.

Proposition 2.8. *Let* T *be a gKPF. Then* $(\delta_T)_X : L_T \Pi X \to \Pi T X$ *is an isomorphism for all finite sets* X.

Proof. For finite X, $(\delta_T)_X : L_T \Pi X \to \Pi T X$ is an isomorphism in all of the 5 cases of Definition 2.6. The result then follows by induction, using that all the functors involved restrict to finite sets and finite BAs. □

The property of Proposition 2.8, namely

$$L \Pi X \cong \Pi T X \qquad \text{for all finite sets } X, \tag{7}$$

or, equivalently, $LA \cong \Pi T \Sigma A$ for finite A, is of central importance as it sets up the relationship between the logic (=functors L given by a presentation) and the semantics (=functor T). (7) can be read in two different ways: If the logic (ie L and $L\Pi \to \Pi T$) is given, then (7) is a requirement; on the other hand, given T, we can take (7) also as a definition of L (up to isomorphism) and look for a presentation of L, which then gives us a syntax and proof system of a logic for T-coalgebras.[1]

To summarise, we might say the whole point of the paper is to show that, once we presented a functor L satisfying (7), everything else flows from this: syntax and proof system are determined by the presentation and the semantics is determined by (7). This also means that the approach presented in the next section is not restricted to gKPFs.

[1] For a general definition of 'presentation of a functor' and how presentations give rise to logics see [6]. Further investigations can be found in [17] showing, for example, that an endofunctor on BA has a finitary presentation iff it preserves filtered colimits.

3 The Goldblatt-Thomason Theorem for Coalgebras

To clarify the relationship between L-algebras and T-coalgebras in diagram (2), we review the categorical analysis given in [17], before returning the special case of Boolean algebras and sets.

3.1 Algebras and Coalgebras on Ind- and Pro-completions

The general picture[2] underlying the situation discussed in the introduction is

$$\begin{array}{ccc} \mathsf{Ind}\mathcal{C} & \underset{\Sigma}{\overset{\Pi}{\rightleftarrows}} & (\mathsf{Ind}\mathcal{C}^{\mathrm{op}})^{\mathrm{op}} \\ \uparrow{\scriptstyle(\hat{-})} & & \uparrow{\scriptstyle(\bar{-})} \\ \mathcal{C} & \rightleftarrows & \mathcal{C}^{\mathrm{op}} \end{array} \tag{8}$$

where $\mathcal{C}$ is a finitely complete and cocomplete category, $\mathsf{Ind}\mathcal{C}$ is the full subcategory of $\mathsf{Set}^{\mathcal{C}^{\mathrm{op}}}$ of finite limit preserving functors, $(\hat{-})$ and $(\bar{-})$ are the Yoneda embeddings. It is well-known that, under these assumptions, $\mathsf{Ind}\mathcal{C}$ is the completion of $\mathcal{C}$ with filtered colimits, see eg [14, Chapter VI]. Dually,

$$\mathsf{Pro}\mathcal{C} \stackrel{\mathrm{def}}{=} (\mathsf{Ind}\mathcal{C}^{\mathrm{op}})^{\mathrm{op}}$$

is the completion of $\mathcal{C}$ with cofiltered limits. Furthermore, we let Σ be the left Kan-extension of $(\bar{-})$ along $(\hat{-})$, and Π the right Kan-extension of $(\hat{-})$ along $(\bar{-})$ (in particular, $\Sigma\hat{C} \cong \bar{C}$, $\Pi\bar{C} \cong \hat{C}$). Σ is left adjoint to Π.

Example 3.1.
1. $\mathcal{C} = \mathsf{BA}_\omega$ (finite Boolean algebras = finitely presentable Boolean algebras), $\mathsf{Ind}\mathcal{C} = \mathsf{BA}$, $\mathsf{Pro}\mathcal{C} = \mathsf{Set}^{\mathrm{op}}$. ΣA is the set of ultrafilters over A and Π is (contravariant) powerset.
2. $\mathcal{C} = \mathsf{DL}_\omega$ (finite distributive lattices = finitely presentable distributive lattices), $\mathsf{Ind}\mathcal{C} = \mathsf{DL}$, $\mathsf{Pro}\mathcal{C} = \mathsf{Poset}^{\mathrm{op}}$. ΣA is the set of prime filters over A and Π gives the set of upsets.
3. In fact, (8) can be instantiated with any locally finite variety for $\mathsf{Ind}\mathcal{C}$. (A variety is locally finite if finitely generated free algebras are finite.)

We are interested in coalgebras over $(\mathsf{Ind}\mathcal{C}^{\mathrm{op}})$, ie, algebras over $\mathsf{Pro}\mathcal{C} = (\mathsf{Ind}\mathcal{C}^{\mathrm{op}})^{\mathrm{op}}$. Consider

$$L \circlearrowright \mathsf{Ind}\mathcal{C} \underset{\Sigma}{\overset{\Pi}{\rightleftarrows}} \mathsf{Pro}\mathcal{C} \circlearrowleft T \tag{9}$$

where we assume that L and T agree on $\mathcal{C}$, that is,

$$L\Pi\bar{C} \cong L\hat{C} \cong \Pi T\bar{C} \qquad \Sigma L\hat{C} \cong T\bar{C} \cong T\Sigma\hat{C} \tag{10}$$

[2] Actually, the general picture is even more general, see Lawvere [19, Section 7], an interesting special case of which is investigated in [18,21].

Example 3.2. For $\mathsf{Ind}\mathcal{C} = \mathsf{BA}$ and $\mathsf{Pro}\mathcal{C} = \mathsf{Set}^{\mathrm{op}}$, the gKPF T and the $L = L_T$ satisfy (10) by Proposition 2.8.

Remark 3.3. We will usually denote by the same symbol a functor and its dual, writing eg $T : \mathcal{K} \to \mathcal{K}$ and $T : \mathcal{K}^{\mathrm{op}} \to \mathcal{K}^{\mathrm{op}}$.

In order to lift Π and Σ to algebras, we extend the natural isomorphisms (10) from $\mathcal{C}$ to $\mathsf{Ind}\mathcal{C}$ and $\mathsf{Pro}\mathcal{C}$, respectively. As a result of the procedure below, the lifted $L\Pi \to \Pi T$ and $T\Sigma \to \Sigma L$ will in general not be isomorphisms, the second may even fail to be natural.

The natural transformation $\delta : L\Pi \to \Pi T$. ΠX is a filtered colimit $\hat{C}_i \to \Pi X$. If L preserves filtered colimits we therefore obtain $L\Pi \to \Pi T$ as in

$$\begin{array}{ccccc} \Pi X & & L\Pi X & \xrightarrow{\delta_X} & \Pi T X \\ \uparrow{\scriptstyle c_i} & & \uparrow{\scriptstyle Lc_i} & & \uparrow{\scriptstyle \Pi T c_i^\sharp} \\ \hat{C}_i & & L\hat{C}_i & \xrightarrow{=} & \Pi T \Sigma \hat{C}_i \end{array} \tag{11}$$

where $c_i^\sharp : \Sigma\hat{C}_i \to X$ is the transpose of $c_i : \hat{C}_i \to \Pi X$. δ allows us to lift Π to a functor

$$\mathsf{Alg}(L) \xleftarrow{\tilde{\Pi}} \mathsf{Coalg}(T)^{\mathrm{op}} \tag{12}$$

mapping a T-algebra (X, ξ) to the L-algebra $(\Pi X, \xi \circ \delta_X)$.

Example 3.4. For $\mathsf{Ind}\mathcal{C} = \mathsf{BA}$, $\mathsf{Pro}\mathcal{C} = \mathsf{Set}^{\mathrm{op}}$, and T being one of $\mathcal{P}$ or $\mathcal{H}$, δ has been given explicitly in Definition 2.6.

The transformation $h : T\Sigma \to \Sigma L$. We will need that there exists h such that the following diagram commutes in $\mathsf{Pro}\mathcal{C}$ (where the d_k are the filtered colimit approximating A).

$$\begin{array}{ccccc} A & & T\Sigma A & \xrightarrow{h_A} & \Sigma L A \\ \uparrow{\scriptstyle d_k} & & \uparrow & & \uparrow \\ \hat{A}_k & & T\Sigma\hat{A}_k & \xrightarrow{=} & \Sigma L \hat{A}_k \end{array} \tag{13}$$

Remark 3.5. A sufficient condition for the existence of h is that T weakly preserves filtered colimits in $\mathsf{Pro}\mathcal{C}$, or, equivalently, weakly preserves cofiltered limits in $(\mathsf{Pro}\mathcal{C})^{\mathrm{op}}$. If T preserves these limits (non weakly) then h is natural.

Example 3.6. For gKPFs excluding $\mathcal{H}$, the maps h have been described by Jacobs [13, Definition 5.1]. We detail the definitions of the following to cases.

1. $h_A : \Sigma L_{\mathcal{P}} A \to \mathcal{P}\Sigma A$ maps $v \in \Sigma L_{\mathcal{P}} A$ to $\{u \in \Sigma A \mid \Box a \in v \Rightarrow a \in u\}$.
2. $h_A : \Sigma L_{\mathcal{H}} A \to \mathcal{H}\Sigma A$ maps $v \in \Sigma L_{\mathcal{H}} A$ to $\{\hat{a} \in 2^{\Sigma A} \mid \Box a \in v\}$.

Remark 3.7. There is a systematic way of calculating h from δ. For $A \in \mathcal{C}$, denoting the unit and counit of the adjunction $\Sigma \dashv \Pi$ by η and ε, h_A is given in $(\mathsf{Pro}\mathcal{C})^{\mathrm{op}}$ (thinking of Set) by

$$\Sigma LA \xrightarrow{(\Sigma L\eta_A)^\circ} \Sigma L\Pi\Sigma A \xrightarrow{(\Sigma\delta_{\Sigma A})^\circ} \Sigma\Pi T\Sigma A \xrightarrow{(\varepsilon_{T\Sigma A})^\circ} T\Sigma A \tag{14}$$

Here we use that the arrows above are isos and we can take their inverse, denoted by $^\circ$. The calculations showing that Example 3.6 derives directly from (14) are detailed[3] in the appendix.

In general, h_A is not uniquely determined by (13) and we cannot assume it to be natural. Nevertheless, in the cases we are aware of h is natural.

Proposition 3.8. *For gKPFs T, the map*

$$h : \Sigma L_T \to T\Sigma$$

in Set *is natural.*

Proof. The type constructors $K_C, +, \times$ preserve cofiltered limits, hence the corresponding map h defined by (13) is uniquely determined and therefore natural. In the other two cases, $T = \mathcal{P}$ and $T = \mathcal{H}$, we take Example 3.6 as the definition of h and verify that it is natural and satisfies (13). We detail this for $T = \mathcal{H}$. Note first that $h_A : \Sigma LA \to \mathcal{H}\Sigma A$ is $\nu_{\Sigma A} \circ in_A^{-1}$ where in_A is the insertion of generators $A \to LA, a \mapsto \Box a$, and $\nu_X : X \to \mathcal{H}X$ maps x to $\{a \subseteq X \mid x \in a\}$. Now both the commutativity of (13) and the naturality of h follow from naturality of in and ν. $\Box$

To finish the category theoretic part of our development, we note that h allows us to lift Σ to

$$\mathsf{Alg}(L) \xrightarrow[\tilde{\Sigma}]{} \mathsf{Coalg}(T)^{\mathrm{op}} \tag{15}$$

via $(LA \to A) \mapsto (\Sigma A \to \Sigma LA \to T\Sigma A)$. If h is natural, then this map is a functor.

3.2 The Goldblatt-Thomason Theorem for Coalgebras

We used the general categorical framework to clarify the relationship between the functors T and L. We will now return to the special case discussed in the introduction. In particular, $\mathsf{Ind}\mathcal{C} = \mathsf{BA}$ and $\mathsf{Ind}\mathcal{C}^{\mathrm{op}} = \mathsf{Set}$; $\Pi : \mathsf{Set} \to \mathsf{BA}$ maps X to 2^X and $\Sigma : \mathsf{BA} \to \mathsf{Set}$ maps a Boolean algebra A to the set of ultrafilters over A.

We say that a functor $T : \mathsf{Set} \to \mathsf{Set}$ *preserves finite sets* if T maps finite sets to finite sets.

[3] We hope these calculations show that *isomorphisms do work*. This balances Conor McBride's view, from a programming perspective, that *isomorphisms cost*.

Definition 3.9 (modal logic of a functor). *The modal logic of a functor* $T : \mathsf{Set} \to \mathsf{Set}$ *is the pair* $(L, \delta : L\Pi \to \Pi T)$ *where* $L = \Pi T \Sigma$ *on finite Boolean algebras and* L *is continuously extended to all of* BA*.* δ *is then given as in (11).*

Remark 3.10. 1. The definition of L does not require T to preserve finite sets. This condition, which implies the right-hand side of (10), is needed for h in (13).

2. For gKPFs T, the modal logic corresponding to (L, δ) has been described explicitly in Example 2.4. But we know from [17] that any $L : \mathsf{BA} \to \mathsf{BA}$ arising from Definition 3.9 has such a presentation by modal operators and axioms.

The notion of a modal theory now arises from the initial, or free, L-algebra, see diagram (6).

Definition 3.11 (modal theory). *Consider a functor* $T : \mathsf{Set} \to \mathsf{Set}$ *with its associated modal logic* (L, δ) *and a* T*-coalgebra* (X, ξ)*.*

1. *Let* I *be the initial* L*-algebra and* $[\![-]\!] : I \to \Pi(X, \xi)$ *be the unique morphism. Then the* variable-free modal theory *of* (X, ξ) *is* $\{\varphi \in I \mid [\![\varphi]\!] = X\}$*.*
2. *Let* I_P *be the free* L*-algebra over the free Boolean algebra generated by a countable set* P *of propositional variables. Let* $[\![-]\!]_v : I_P \to \Pi(X, \xi)$ *be the unique morphism extending a valuation* $v : P \to \Pi X$ *of the propositional variables. Then the* modal theory *of* (X, ξ) *is* $\{\varphi \in I_P \mid [\![\varphi]\!]_v = X$ *for all* $v : P \to \Pi X\}$*.*

The next proposition provides the first main ingredient to the Goldblatt-Thomason theorem, namely that modally definable classes 'reflect' ultrafilter extensions. In case of variable-free theories, definable classes are also closed under ultrafilter extensions.

Proposition 3.12. *Let* $T : \mathsf{Set} \to \mathsf{Set}$ *preserve finite sets and assume that* h *as in (13) exists. Then*

1. (X, ξ) *and* $\Sigma\Pi(X, \xi)$ *have the same variable-free modal theory,*
2. (X, ξ) *satisfies the modal theory of* $\Sigma\Pi(X, \xi)$*.*

Proof. (1): By construction of the logic from L, a formula φ is an element of the initial L-algebra and $(X, \xi) \models \varphi$ iff the unique morphism $[\![-]\!]$ from the initial L-algebra to $\Pi(X, \xi)$ maps φ to $X \in \Pi(X, \xi) = 2^X$. Therefore[4], to show that $(X, \xi) \models \varphi \Leftrightarrow \Pi(X, \xi) \models \varphi$, it suffices to establish that the map $\iota : \Pi(X, \xi) \to \Pi\Sigma\Pi(X\xi)$ is an injective algebra morphism. This follows from (the proof of) Theorem 5.3 in [17] and Stone's representation theorem for Boolean algebras. (2): Suppose there is a valuation v showing that φ does not hold in (X, ξ), that is, $[\![\varphi]\!]_v \neq X$. Then $\iota \circ [\![-]\!]_v(\varphi) \neq \Sigma\Pi X$, that is, there is a valuation showing that φ does not hold in $\Sigma\Pi(X, \xi)$. □

[4] Note that the top-element X of the BA $\Pi(X, \xi)$ is preserved by algebra morphisms.

The second main ingredient (of the algebraic proof) of the Goldblatt-Thomason theorem is Birkhoff's variety theorem stating that a class of algebras is definable by equations iff it is closed under homomorphic images (H), subalgebras (S), and products (P). A set of equations is called *ground* if it does not contain any variables. This corresponds to the absence of propositional variables in a modal theory, or, in other words, to treating Kripke models instead of Kripke frames. The lesser expressivity of ground equations is reflected algebraically by also closing under embeddings (E). Closure under H, S, P (and E) is equivalent to closure under HSP ($EHSP$).

Theorem 3.13 (Birkhoff's variety theorem). *A class of algebras is definable by a set of*

1. *ground equations iff it is closed under $EHSP$,*
2. *equations iff it is closed under HSP.*

Proof. We sketch the proof of the less well-known 1st statement. It is routine to check that a definable class of algebras enjoys the required closure properties. Conversely, let $\mathcal{K}$ be a class of algebras closed under $EHSP$ and let Φ be the ground theory of $\mathcal{K}$. Consider an algebra A with $A \models \Phi$. We have to show that $A \in \mathcal{K}$. Since $\mathcal{K}$ is closed under SP, the quotient $Q = I/\Phi$ of the initial algebra I by Φ is in $\mathcal{K}$. $A \models \Phi$ then means that there is a morphism $Q \to A$, hence $A \in \mathcal{K}$ by closure under EH.

The dual of closure under S and E is closure under quotients and domains of quotients. This is equivalent to closure under 'co-spans' $(X, \xi) \twoheadrightarrow \bullet \twoheadleftarrow (X', \xi')$, or bisimilarity:

Definition 3.14 (bisimilarity). *Two coalgebras (X, ξ), (X', ξ') are bisimilar if there are surjective coalgebra morphisms $(X, \xi) \twoheadrightarrow \bullet \twoheadleftarrow (X', \xi')$.*

We can now generalise to coalgebras the Goldblatt-Thomason theorem [11] for Kripke frames and Venema's corresponding result for Kripke models [22]. For a textbook account of the former see [3,4]. [4, Theorem 5.54] gives an excellent account of the algebraic proof that we generalise, [4, Theorem 3.19] presents an alternative model theoretic proof, and [4, Theorem 2.75] gives a version for pointed Kripke models.

We say that a class $\mathcal{K}$ of coalgebras is *closed under ultrafilter extensions* if $(X, \xi) \in \mathcal{K} \Rightarrow \Sigma\Pi(X, \xi) \in \mathcal{K}$ and that it *reflects ultrafilter extensions* if $\Sigma\Pi(X, \xi) \in \mathcal{K} \Rightarrow (X, \xi) \in \mathcal{K}$.

The first part of the theorem below is the definability result for coalgebras as generalisations of Kripke models, the second part treats definability for coalgebras as generalisations of Kripke frames. The difference in the formulation, apart from replacing closure under bisimilarity by closure under quotients, is due to the fact that all modally definable classes of Kripke models but not all modally definable classes of Kripke frames are closed under ultrafilter extensions (compare the two items of Proposition 3.12).

Theorem 3.15. *Let* $T : \mathsf{Set} \to \mathsf{Set}$ *preserve finite sets and assume there is a natural transformation* h *satisfying (13).*

1. *A class* $\mathcal{K} \subseteq \mathsf{Coalg}(T)$ *is definable by a variable-free modal theory iff* $\mathcal{K}$ *is closed under subcoalgebras, bisimilarity, coproducts and ultrafilter extensions and* $\mathcal{K}$ *reflects ultrafilter extensions.*
2. *A class* $\mathcal{K} \subseteq \mathsf{Coalg}(T)$ *closed under ultrafilter extensions is definable by a modal theory iff* $\mathcal{K}$ *is closed under subcoalgebras, quotients and coproducts and* $\mathcal{K}$ *reflects ultrafilter extensions.*

Proof. (1): For 'if' let X be a coalgebra that is a model of the theory of $\mathcal{K}$, that is, by Theorem 3.13.1, $\Pi X \in EHSP(\Pi\mathcal{K})$ where $\Pi\mathcal{K} = \{\Pi Y \mid Y \in \mathcal{K}\}$. We have to show $X \in \mathcal{K}$. ΠX embeds a quotient of a subalgebra of a product $\prod_i \Pi(X_i)$, $X_i \in \mathcal{K}$. Since Π is right adjoint, we obtain $\prod_i \Pi(X_i) \cong \Pi(\coprod_i X_i)$. Since Σ maps injective maps to surjective maps and vice versa, we have

$$\Sigma\Pi X \twoheadrightarrow \bullet \hookrightarrow \bullet \twoheadleftarrow \Sigma\Pi(\coprod X_i).$$

The stipulated closure properties now imply $X \in \mathcal{K}$. For 'only if', using that ground equationally definable classes of algebras are closed under $EHSP$, it is enough to observe (i) that Π maps surjective maps to injective maps and vice versa, (ii) that Π maps coproducts to products, (iii) Proposition 3.12.1.
(2): The proof is a straightforward variation of the previous one. For 'if' let X be a coalgebra that is a model of the theory of $\mathcal{K}$, that is, by Theorem 3.13.2, $\Pi X \in HSP(\Pi\mathcal{K})$ where $\Pi\mathcal{K} = \{\Pi Y \mid Y \in \mathcal{K}\}$. We have to show $X \in \mathcal{K}$. ΠX is a quotient of a subalgebra of a product $\prod_i \Pi(X_i)$, $X_i \in \mathcal{K}$. We have

$$\Sigma\Pi X \hookrightarrow \bullet \twoheadleftarrow \Sigma\Pi(\coprod X_i).$$

The stipulated closure properties now imply $X \in \mathcal{K}$. For 'only if', we use (i) and (ii) as in part 1 and (iii) Proposition 3.12.2. □

Before deriving our main result as a corollary, let us analyse the hypotheses needed for Theorem 3.15 in terms of the general setting discussed in Section 3.1.

Remark 3.16. The following ingredients are used in the proof of Theorem 3.15.

1. $\mathcal{C}$ in diagram (8) has all finite limits and finite colimits. This is a strong requirement. But it holds if $\mathsf{Ind}\mathcal{C}$ is a locally finite variety and $\mathcal{C}$ is the subcategory of finitely presentable algebras. This includes BA and DL.
2. $A \to \Sigma\Pi A$ is injective. This is unlikely to hold in the generality of diagram (8) but it seems to be a rather mild requirement: For example, it holds for (subvarieties of) BA and DL.
3. T preserves finite sets (or, more generally, T restricts to $\mathcal{C}^{\mathrm{op}}$). This is needed in diagram 13. It excludes polynomial functors with infinite constants and the probability distribution functor. For a further discussion and the connection with strong completeness see [17].

4. h exists and is natural. The status of this requirement remains somewhat unclear. As emphasised by the corollary, it is satisfied in important examples. Let us note here that h is certainly natural if T preserves cofiltered limits. This is the case for all polynomial functors. The example we are aware of where the existence of h fails, is if T is the finite powerset functor (the ultrafilter extension of a Kripke frame is not finitely branching).

The main result of the paper is the following corollary. The second part generalises the Goldblatt-Thomason theorem from Kripke frames to all gKPF-coalgebras and the first part generalises Venema's definability theorem for Kripke models to all gKPF-coalgebras.

Corollary 3.17. *Let T be a gKPF.*

1. *A class $\mathcal{K} \subseteq \mathsf{Coalg}(T)$ is definable by a variable-free modal theory iff $\mathcal{K}$ is closed under subcoalgebras, bisimilarity, coproducts and ultrafilter extensions and $\mathcal{K}$ reflects ultrafilter extensions.*
2. *A class $\mathcal{K} \subseteq \mathsf{Coalg}(T)$ closed under ultrafilter extensions is definable by a modal theory iff $\mathcal{K}$ is closed under subcoalgebras, quotients and coproducts and $\mathcal{K}$ reflects ultrafilter extensions.*

Remark 3.18. 1. As far as we know, the special case of $\mathcal{H}$-coalgebras (neighbourhood frames) is a new result.

2. In the statement of the theorem, we can replace "$\mathsf{Coalg}(T)$" by a modally definable full subcategory of $\mathsf{Coalg}(T)$. For example, the theorem holds for monotone neighbourhood frames or topological neighbourhood frames. For topological spaces, the result is due to Gabelaia [9, Theorem 2.3.4], but see also ten Cate et al [7].

The original formulation of Venema's theorem [22] has closure under surjective bisimulations instead of closure under subcoalgebras and bisimilarity. The relationship between the two formulations is as follows. For functors T that preserve weak pullbacks, one can use 'spans' $(X, \xi) \twoheadleftarrow \bullet \twoheadrightarrow (X', \xi')$ in the definition of bisimilarity instead of co-spans $(X, \xi) \twoheadrightarrow \bullet \twoheadleftarrow (X', \xi')$ as above. Closure under subcoalgebras (or generated submodels in the parlance of [22]) is incorporated in the notion of surjective bisimulation by not forcing the left-hand projection of the span to be surjective: A class $\mathcal{K}$ is closed under *surjective bisimulations* iff for all $(X, \xi) \in \mathcal{K}$ and all $(X, \xi) \leftarrow \bullet \twoheadrightarrow (X', \xi')$ also $(X', \xi') \in \mathcal{K}$. Since $\mathcal{H}$ is the only ingredient of a gKPF that does not preserve weak pullbacks, we obtain the following generalisation of Venema's definability theorem for Kripke models.

Corollary 3.19. *Let T be a KPF (ie a gKPF built without using $\mathcal{H}$). A class $\mathcal{K} \subseteq \mathsf{Coalg}(T)$ is definable by a variable-free modal theory iff $\mathcal{K}$ is closed under coproducts and surjective bisimulations and $\mathcal{K}$ reflects ultrafilter extensions.*

4 Conclusion

The basic idea underlying this (and previous) work is to consider the logics for coalgebras as functors L on a category of propositional logics such as BA. Let us summarise some features of this approach.

- The functor L packages up all the information about modal operators and their axioms. In this way the functor L acts as an interface to the syntax, which is given by a presentation of L.
- As long as we only use abstract properties of L, such as (7), we can prove theorems about modal logics in a syntax free way, see Corollary 3.17 or the Jónsson-Tarski theorem [17, Thm 5.3] for examples. This gives rise not only to simpler proofs, but also to more general results.
- If we instantiate our abstract categories and functors with concrete presentations, we not only get back all the riches of syntax, but find that the categorical constructions actually do work for us. For example, in diagram (8), if we let $\mathcal{C} = \mathsf{BA}_\omega$ and $\mathcal{C}^{\mathrm{op}} = \mathsf{Set}_\omega$, then the fact that Π is contravariant powerset and Σ is ultrafilters, follows from the end/coend formula for right/left Kan extensions. Another example of this phenomenon is detailed in the appendix.

Another point is that the generality of our approach suggests further work. Let us mention the following:

- It is possible to replace Boolean algebras by distributive lattices. It could be of interest to look at the details.
- It should be possible to alleviate the restriction to finite constants insofar as infinite 'input' constants C as in T^{K_C} can be allowed. But the restriction to finite 'output' sets is important, see Friggens and Goldblatt [8] for a detailed discussion.
- Is it possible to generalise definability results for pointed models or frames in the same framework?
- It should be of interest to instantiate $\mathsf{Ind}\mathcal{C}$ in diagram (8) with other locally finite varieties.
- Diagram (8) can also be varied in many directions, for example, considering other completions than Ind or going to an enriched setting (for example, for posets (ie enrichment over 2) the Galois closed subsets of the adjunction $\Sigma \dashv \Pi$ describe the elements of the MacNeille completion of $\mathcal{C}$).

Finally, and from the point of view of logics for coalgebras, most importantly: Can we find a similarly nice and abstract account for functors T that do not preserve finite sets?

Acknowledgements. Our greatest debts are to David Gabelaia whose sharp eye allowed us to correct a serious mistake in the statement of our main theorem. He also suggested numerous other improvements and made us aware of Venema's paper on definability for Kripke models [22]. The first author would like to thank Dmitry Sustretov for the discussions during ESSLLI 2006 in Malaga, which prompted the writing of this paper; and Neil Ghani for relating Conor McBride's statement about isomorphisms, see footnote 3. We are also grateful to the referees for their suggestions.

References

1. Abramsky, S.: Domain theory in logical form. Annals of Pure and Applied Logic 51 (1991)
2. Abramsky, S.: A Cook's tour of the finitary non-well-founded sets. In: We Will Show Them: Essays in Honour of Dov Gabbay. College Publications, 2005. Presented at BCTCS (1988)
3. van Benthem, J.: Modal Logic and Classical Logic. Bibliopolis (1983)
4. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. CUP (2001)
5. Bonsangue, M., Kurz, A.: Duality for logics of transition systems. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, Springer, Heidelberg (2005)
6. Bonsangue, M., Kurz, A.: Presenting functors by operations and equations. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006 and ETAPS 2006. LNCS, vol. 3921, Springer, Heidelberg (2006)
7. ten Cate, B., Gabelaia, D., Sustretov, D.: Modal languages for topology: expressivity and definability. arXiv 0610357 (2006)
8. Friggens, D., Goldblatt, R.: A modal proof theory for final polynomial coalgebras. Theoret. Comput. Sci. 360 (2006)
9. Gabelaia, D.: Modal definability in topology. Master's thesis, University of Amsterdam (2001)
10. Ghilardi, S.: An algebraic theory of normal forms. Ann. Pure Appl. Logic, 71 (1995)
11. Goldblatt, R.I., Thomason, S.K.: Axiomatic classes in propositional modal logic. In: Algebra and Logic (1974)
12. Hansen, H., Kupke, C.: A coalgebraic perspective on monotone modal logic. In: CMCS'04. ENTCS, vol. 106 (2004)
13. Jacobs, B.: Many-sorted coalgebraic modal logic: a model-theoretic study. Theor. Inform. Appl. 35 (2001)
14. Johnstone, P.: Stone Spaces. Cambridge University Press, Cambridge (1982)
15. Kupke, C., Kurz, A., Pattinson, D.: Algebraic semantics for coalgebraic logics. In: CMCS'04. ENTCS, vol. 106 (2004)
16. Kupke, C., Kurz, A., Pattinson, D.: Ultrafilter extensions of coalgebras. In: Fiadeiro, J.L., Harman, N., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, Springer, Heidelberg (2005)
17. Kurz, A., Rosick ý, J.: Strongly complete logics for coalgebras (July 2006) Submitted
18. Lawvere, F.: Metric spaces, generalized logic and closed categories. Rendiconti del Seminario Matematico e Fisico di Milano, XLIII, Also appeared as TAC reprints No. 1 (1973)
19. Lawvere, F.: Taking categories seriously. Revista Colombiana de Matemáticas XX (1986) Also appeared as TAC reprints No. 8.
20. Rößiger, M.: Coalgebras and modal logic. In: CMCS'00. ENTCS, vol. 33 (2000)
21. Rutten, J.: Weighted colimits and formal balls in generalized metric spaces. Topology and its Applications 89 (1998)
22. Venema, Y.: Model definability, purely modal. In: JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday, Amsterdam University Press (1999)

A Appendix

We show that the h in Example 3.6 are derived from (14). To this end we first state a lemma on ultrafilters, which is a straightforward consequence of the respective definitions.

Notation. $\eta : Id \to \Pi\Sigma$ and $\varepsilon : Id \to \Sigma\Pi$ are the (co)unit of the adjunction $\Sigma \dashv \Pi$ (note that we wrote ε here as an arrow in Set and not in $\mathsf{Set}^{\mathrm{op}}$). f° denotes the converse of a bijection f. For $a \in A$ we abbreviate $\eta(a) = \{u \in \Pi\Sigma A \mid a \in u\}$ by $\hat{a}$. The complement $X \setminus S$ of a subset S of X is written as $-S$.

Lemma A.1. *Let A be a finite* BA*, Y a finite set and L one of $L_{\mathcal{P}}$, $L_{\mathcal{H}}$.*

1. *Every ultrafilter $u \in \Sigma A$ has a smallest element given by $\bigwedge_{a \in u} a$.*
2. *$(\varepsilon_Y)^\circ : \Sigma\Pi Y \to Y$ maps u to y, where $\{y\}$ is the smallest element of u.*
3. *Every ultrafilter $v \in \Sigma LA$ is determined by the set $\{\Box a \mid a \in A, \Box a \in v\}$.*
4. *The smallest element of $v \in \Sigma LA$ is given by $\bigwedge_{\Box a \in v} \Box a \wedge \bigwedge_{\Box a \notin v} \neg \Box a$.*

Also note that for isos f we have that $(\Sigma f)^\circ$ is the direct image map of f. We obtain for $L = L_{\mathcal{P}}$:

$(\Sigma L \eta_A)^\circ$ maps v to the ultrafilter determined by $\{\Box\hat{a} \mid \Box a \in v\}$,
$(\Sigma \delta_{\Sigma A})^\circ$ maps $\{\Box\hat{a} \mid \Box a \in v\}$ to $\{\{t \in \mathcal{P}\Sigma A \mid t \subseteq \hat{a}\} \mid \Box a \in v\}$,
$(\varepsilon_{\mathcal{P}\Sigma A})^\circ$ maps the ultrafilter determined by $\{\{t \in \mathcal{P}\Sigma A \mid t \subseteq \hat{a}\} \mid \Box a \in v\}$ to $\bigcap_{\Box a \in v}\{t \in \mathcal{P}\Sigma A \mid t \subseteq \hat{a}\} \cap \bigcap_{\Box a \notin v} -\{t \in \mathcal{P}\Sigma A \mid t \subseteq \hat{a}\}$.

It is now a straightforward verification to check that this last set contains exactly one t which is $\{u \in \Sigma A \mid \Box a \in v \Rightarrow a \in u\}$. Hence we obtained the h described in Example 3.6.1.

For $L = L_{\mathcal{H}}$ we have:

$(\Sigma L \eta_A)^\circ$ maps v to the ultrafilter determined by $\{\Box\hat{a} \mid \Box a \in v\}$,
$(\Sigma \delta_{\Sigma A})^\circ$ maps $\{\Box\hat{a} \mid \Box a \in v\}$ to $\{\{t \in \mathcal{H}\Sigma A \mid \hat{a} \in t\} \mid \Box a \in v\}$,
$(\varepsilon_{\mathcal{H}\Sigma A})^\circ$ maps the ultrafilter determined by $\{\{t \in \mathcal{H}\Sigma A \mid \hat{a} \in t\} \mid \Box a \in v\}$ to $\bigcap_{\Box a \in v}\{t \in \mathcal{H}\Sigma A \mid \hat{a} \in t\} \cap \bigcap_{\Box a \notin v} -\{t \in \mathcal{H}\Sigma A \mid \hat{a} \in t\}$.

It is now a straightforward verification that this last set contains exactly one t which is $\{\hat{a} \in 2^{\Sigma A} \mid \Box a \in v\}$. Hence we obtained the h described in Example 3.6.2.

Specification-Based Testing for CoCasl's Modal Specifications

Delphine Longuet and Marc Aiguier

IBISC CNRS FRE 2873 - University of Évry Val d'Essonne
523 place des terrasses de l'Agora, F-91000 Évry
{delphine.longuet,marc.aiguier}@ibisc.univ-evry.fr

Abstract. Specification-based testing is a particular case of black-box testing, which consists in deriving test cases from an analysis of a formal specification. We present in this paper an extension of the most popular and most efficient selection method widely used in the algebraic framework, called axiom unfolding, to coalgebraic specifications, using the modal logic provided by the CoCasl specification language.

Keywords: Specification-based testing, axiom unfolding, coalgebraic specifications, modal logic, CoCasl.

Black-box testing refers to any method used to validate software systems independently of their implementation. Specification-based testing is a particular case of black-box testing, which consists of the dynamic verification of a system with respect to its specification [1,2,3]. The system under test is executed on a finite subset of its possible input data to check its conformance with respect to the specification requirements.

The testing process is classically divided into two principal phases:

1. The *selection* phase where some selection criteria are defined to split test sets into subsets in order to manage their size.
2. The *generation* phase where some techniques and tools based on constraint solving are defined in order to generate some test cases in each test set to be submitted to the system under test.

In this paper, we are interested in the selection phase. More particularly, we will extend to CoCasl specifications a very popular and very efficient selection method, called *axiom unfolding*, which has extensively been studied in the framework of algebraic specifications [1,2,3,4,5,6,7,8,9].

CoCasl is a coalgebraic extension of the algebraic specification language Casl that allows to specify processes as coalgebraic types dealing with data defined as algebraic types [10]. CoCasl's modal logic is syntactical sugar to express properties on such processes, like safety and fairness properties. We then propose in this paper a selection method for testing dynamic systems specified with CoCasl's modal logic.

The usual approach of black-box testing for dynamic systems is conformance testing [11,12,13,14,15]. In conformance testing, specifications, systems and test

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 356–371, 2007.
© Springer-Verlag Berlin Heidelberg 2007

purposes are classically represented by input output transition systems. Test cases are then execution traces selected in the specification by using classic techniques from the automata theory such as synchronised product, symbolic evaluation, etc. Recently, some selection methods from test purposes expressed as temporal properties has been investigated (e.g. see [16]). Taking advantage of the fact that specifications are transition systems, model-checking techniques have been used to select trace sets. Here, CoCasl specifications are logical theories. Hence, our selection method, based on axiom unfolding, will be algorithmically defined by defining a search proof strategy. This strategy will enable one to bound the search space for proofs to a given class of trees having a specific structure (see Section 3). However, the aim of the unfolding procedure will not be to find the entire proof of a test purpose φ, but rather to stretch further the execution of the unfolding procedure in order to make increasingly big proof whose remaining lemmas will define a "partition" of φ. Hence, the procedure will be able to be stopped at any time when the obtained partition will be fine enough according to tester's judgement or needs. Completeness of the unfolding procedure will then be established by showing that derivability restricted to the unfolding strategy coincides with the full derivability (i.e. without any specific proof strategy).

The paper is organised as follows. Section 1 briefly presents CoCasl specification language, especially cotype definition. Then CoCasl's modal logic is introduced, and is given a sequent calculus. To set the framework we work within, Section 2 recalls the relevant definitions from [3] we will use in this paper, such as exhaustive test set, and selection criteria and their associated properties. We also prove in this section the important result of the existence of a reference exhaustive test set, allowing to start the selection procedure with. After having recalled in Section 3.1 the general notions of test set and constrained test set from [17], Section 3.2 introduces the unfolding procedure from which is defined a family of selection criteria for CoCasl's modal specifications. Selection criteria thus defined are proved to be sound and complete in Section 3.3.

1 CoCasl's Modal Logic

CoCasl extends Casl specification language by enriching basic specifications with dual forms of algebraic constructs used in Casl to define inductive datatypes. The basic dual form is the cotype construct which is used to specify processes. A cotype declaration defines a coinductive process by declaring *selectors*, also called *observers*, and constructors. Unlike in Casl specifications, constructors here are optional. For example, the two following cotypes can be declared in CoCasl:

spec Moore =
  sorts *In*, *Out*
  cotype *State* ::= (*next* : *In* → *State*;
                          observe : *Out*)
end

spec List =
  sort *Nat*
  cotype *List* ::= *empty* |
                        insert(*head* :? *Nat*;
                                 tail :? *List*)
end

The first declaration declares the two observers $next : In \times State \rightarrow State$ and $observe : State \rightarrow Out$. The second similarly declares observers *head* and *tail* over the cotype *List*, but also constructors $empty : List$ and $insert : Nat \times List \rightarrow List$, where *Nat* is an imported sort from the local environment. The parts of the declaration separated by vertical bars are called alternatives. For instance, in the LIST specification, alternatives are defined by both constructors *empty* and *insert*. Observers may be unary like *observe*, or may have additional parameters, which have to come from the local environment, like *next*. Both observers and constructors may be partial. Observers are partial as soon as the cotype is defined by several alternatives. As cotypes are dual for types, cotype declarations can be strengthened by declaring a cogenerated cotype to restrict the class of models to fully abstract ones, or a cofree cotype to restrict models to the terminal one. For a complete presentation of COCASL language, the reader may refer to [10].

To express properties on processes declared in COCASL, a multi-sorted modal logic has been defined in [10], where modalities are defined from observers used to describe system evolutions. All the sorts defined in the cotype are called *non-observable*, while sorts from the local environment are called *observable*. The set of non-observable sorts defines a multi-sorted state space, with observers either directly producing an observable value, or making the system state evolve.

Actually, the modal logic presented here is both a restriction and an extension of the one presented in [10]. This is a restriction because we only consider here quantifier-free formulae. But the logic we present is also an extension because atomic formulae are not restricted to equations but may involve any predicate. The restriction to quantifier-free formulae is due to the fact that existentially quantified formulae are impossible to deal with from a testing point of view. As a matter of fact, testing a formula of the form $\exists x\, \varphi(x)$ requires to exhibit a witness value a such that $\varphi(a)$ is evaluated as "true" by the system under test. Of course, there is no general way to find out such a relevant value, but to simply prove that the system satisfies the property. This led us to conclude that existential properties are not testable [8].

Syntax. A COCASL *signature* $\Sigma = (S, F, P, V)$ consists of a set S of sorts with a partition S_{obs} and T of observable and non-observable sorts respectively, a set F of operation names, each one equipped with an arity in $S^* \times S$, a set P of predicate names, each one equipped with an arity in S^+ and an S-indexed set V of variables. For all operations $f : s_1 \times \ldots \times s_n \rightarrow s$ in F and all predicates $p : s_1 \times \ldots \times s_n$ in P, there exists at most one i, $1 \leq i \leq n$, such that $s_i \in T$. We make a distinction between operations coming from the local environment, i.e. operations $f : s_1 \times \ldots \times s_n \rightarrow s$ with $s_1, \ldots, s_n, s \in S_{obs}$ on the one hand, and constructors and observers, that are operations $f : s_1 \times \ldots \times s_n \times s \rightarrow s'$ with $s \in T$ on the other hand. Constructors have a non-observable result sort, while observers may be with observable result sort $s' \in S_{obs}$ (they are also called attributes) or with non-observable result sort $s' \in T$ (these are also called methods). Constructors and methods are only distinguished from each other thanks to the cotype declaration: the above LIST declaration declares *empty* and *insert* as constructors, *head* as an observer with observable sort, and *tail* as

an observer with non-observable sort. We call an observer $f : s_1 \times \ldots \times s_n \times s \to s'$ observer of cotype s. The set F of operations names is then a partition $F = F_{obs} \amalg F_\Omega \amalg (F_s)_{s \in T}$ where F_{obs} is the set of operations from the local environment, F_Ω is the set of constructors and for all $s \in T$, F_s is the set of observers of cotype s. Since a cotype may be declared using several alternatives, observers for a given cotype are actually defined for a given alternative of this cotype. For a cotype s having m alternatives, we then have $F_s = \coprod_{1 \leq j \leq m} F_{s,j}$ where $F_{s,j}$ is the set of observers for the jth alternative of cotype s. The set P of predicates is also a partition $P_{obs} \amalg (P_s)_{s \in T}$ where P_{obs} is the set of predicates only involving observable sorts , and for each $s \in T$, P_s is the set of predicates $p : s_1 \times \ldots \times s_n \times s$. The above List declaration gives the following CoCasl signature.

$S_{obs} = \{Nat\} \qquad T = \{List\}$
$F_\Omega = \{empty : List, insert : Nat \times List \to List\} \qquad P_{List} = \{def : List\}$
$F_{List,1} = \emptyset$
$F_{List,2} = \{head : List \to Nat, tail : List \to List\}$

where alternative 1 corresponds to the empty list, and alternative 2 to a list built with constructor *insert*.

Given a signature $\Sigma = (S, F, P, V)$, $T_\Sigma(V)$ is the S-set of *terms with variables* in V defined inductively from variables in V and operations of F: for each operation $f : s_1 \times \ldots \times s_n \to s \in F_{obs} \cup F_\Omega$, $f(t_1, \ldots, t_n) \in T_\Sigma(V)_s$, where each $t_i \in T_\Sigma(V)_{s_i}$, $1 \leq i \leq n$; for each observer $f : s_1 \times \ldots \times s_n \times s \to s'$, $f(t_1, \ldots, t_n) \in T_\Sigma(V)_{s'}$, where each $t_i \in T_\Sigma(V)_{s_i}$, $1 \leq i \leq n$. Notice that, for observers, the sort s has been removed. This allows to consider states as implicit, as usual with modal logic. The set of *ground terms* T_Σ is defined as the set of terms built over the empty set of variables $T_\Sigma(\emptyset)$. A *substitution* is any mapping $\sigma : V \to T_\Sigma(V)$ that preserves sorts. Substitutions are naturally extended to terms with variables and then to formulae.

Σ-atomic formulae are sentences of the form $p(t_1, \ldots, t_n)$ where $p : s_1 \times \ldots \times s_n \in P_{obs}$ or $p : s_1 \times \ldots \times s_n \times s \in P_s$, and $t_i \in T_\Sigma(V)_{s_i}$ for each i, $1 \leq i \leq n$. A term t with non-observable sort leads to modalities $[t]$, $\langle t \rangle$, $[t*]$ and $\langle t* \rangle$, intuitively meaning "all next state", "some next state", "always" and "eventually", respectively. Modalities can be extended to finite sequences $\{t_1, \ldots, t_n\}$, where $[\{t_1, \ldots, t_n\}]\varphi$ and $\langle \{t_1, \ldots, t_n\} \rangle \varphi$ stand respectively for the conjunction and the disjunction of the modal formulae obtained for the corresponding individual modalities. *Formulae* are then built following the syntax:

$$\begin{aligned} \varphi, \psi ::= \; & true \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid [t]\varphi \mid \langle t \rangle \varphi \mid [t*]\varphi \mid \langle t* \rangle \varphi \\ & \mid [\{t_1, \ldots, t_n\}]\varphi \mid \langle \{t_1, \ldots, t_n\} \rangle \varphi \mid [\{t_1, \ldots, t_n\}*]\varphi \mid \langle \{t_1, \ldots, t_n\}* \rangle \varphi \end{aligned}$$

The set of modalities is denoted by $M_\Sigma(V)$. $For(\Sigma)$ is the set of all Σ-formulae. A *specification* $Sp = (\Sigma, Ax)$ consists of a signature Σ and a set Ax of formulae often called *axioms*. The List declaration above generates, besides the signature we gave, the following axioms, as well as the five axioms specifying that the equality predicate is the existential equality:

$$\begin{array}{lll} \neg def(head(empty)) & & head(insert(n,l)) = n \\ \neg def(tail(empty)) & & tail(insert(n,l)) = l \\ x = x & t = t' \Rightarrow t' = t & t = t' \wedge t' = t'' \Rightarrow t = t'' \\ \multicolumn{3}{l}{t_1 = t'_1 \wedge \cdots \wedge t_n = t'_n \wedge def(f(t_1,\ldots,t_n)) \Rightarrow f(t_1,\ldots,t_n) = f(t'_1,\ldots,t'_n)} \\ def(f(t_1,\ldots,t_n)) \Rightarrow def(t_i) & \text{(strictness)} & t = t' \Rightarrow def(t) \quad \text{(definability)} \end{array}$$

Semantics. Given a signature $\Sigma = (S, F, P, V)$, we denote by Σ_{obs} the "observable subsignature" $(S, F_{obs} \amalg F_\Omega, P_{obs}, V)$ of Σ. A Σ_{obs}-model $\mathcal{A}$ is then a first-order structure, that is an S-indexed set A, equipped for each operation name $f : s_1 \times \ldots \times s_n \to s \in F_{obs} \amalg F_\Omega$ with a mapping $f^{\mathcal{A}} : A_{s_1} \times \ldots \times A_{s_n} \to A_s$, and for each predicate name $p : s_1 \times \ldots \times s_n \in P_{obs}$ with an n-ary relation $p^{\mathcal{A}} \subseteq A_{s_1} \times \ldots \times A_{s_n}$.

Since several cotypes can be declared in CoCasl, the set of states E is said multi-sorted and is defined as a product $E = \prod_{s \in T} E_s$ where for each $s \in T$, $E_s = A_s$. Σ-models are then coalgebras $(E, \alpha : E \to \mathcal{F}E)$ of the functor $\mathcal{F}$ such that $\mathcal{F}E = \prod_{s \in T} \mathcal{F}E_s$ and which, for each $s \in T$, associates to E_s the set $\mathcal{F}E_s$ defined as follows:

$$\mathcal{F}E_s = \coprod_{1 \le j \le m} \left(\prod_{\substack{f : s_1 \times \ldots \times s_n \times s \to s' \in F_{s,j} \\ s' \in S_{obs}}} A_{s'}^{A_{s_1} \times \ldots \times A_{s_n}} \times \prod_{\substack{f : s_1 \times \ldots \times s_n \times s \to s' \in F_{s,j} \\ s' \in T}} E_{s'}^{A_{s_1} \times \ldots \times A_{s_n}} \right) \times \prod_{p : s_1 \times \ldots \times s_n \times s \in P_s} 2^{A_{s_1} \times \ldots \times A_{s_n}}$$

where sort s is defined by m alternatives. [1] We denote by $Mod(\Sigma)$ the category whose objects are Σ-models, i.e. the category $\mathsf{Coalg}(\mathcal{F})$ of coalgebras over $\mathcal{F}$.

Given a Σ-model (E, α) over a first-order structure $\mathcal{A}$, we denote by $_^A : T_{\Sigma_{obs}} \to A$ the unique homomorphism that maps any Σ_{obs} ground term $f(t_1, \ldots, t_n)$ to its value $f^{\mathcal{A}}(t_1^A, \ldots, t_n^A)$. A Σ-model is said *reachable on data* if $_^A$ is surjective.

Given a Σ-model (E, α), a Σ-interpretation in A is any mapping $\nu : V \to A$ preserving sorts. Given an interpretation of variables ν and a state $e = (e_s)_{s \in T} \in E$, the interpretation of terms in $T_\Sigma(V)$ $\nu_e^\natural : T_\Sigma(V) \to M$ is built in the usual way for variables and operations in $F_{obs} \cup F_\Omega$, and in the following way for observers: if $f : s_1 \times \ldots \times s_n \times s \to s' \in F_{s,j}$ is an observer with observable result sort then $\nu_e^\natural(f(t_1, \ldots, t_n)) = (\pi_f \circ \kappa_j \circ \pi_s \circ \alpha)(e)(\nu_e^\natural(t_1), \ldots, \nu_e^\natural(t_n))$, where: $\pi_s : E \to E_s$ is the canonical projection to the s-sorted part of a state; assuming that the sort s is declared by j alternatives, κ_j is the canonical injection to alternative j; and π_f is the canonical projection from alternative j of E_s to the interpretation of f; if $f : s_1 \times \ldots \times s_n \times s \to s' \in F_{s,j}$ is an observer with non-observable result sort, then $\nu_e^\natural(f(t_1, \ldots, t_n)) = e'$ such that $e' = (e'_s)_{s \in T} \in E$ with $e'_{s''} = e_{s''}$ for all $s'' \neq s'$, and $e_{s'} = (\pi_f \circ \kappa_j \circ \pi_s \circ \alpha)(e)(\nu_e^\natural(t_1), \ldots, \nu_e^\natural(t_n))$. By abuse of notation, the extension $\nu_e^\natural$ of ν will be denoted by ν_e.

The *satisfaction* of a Σ-formula φ by (E, α) for an interpretation ν and a state e, denoted by $(E, \alpha) \models_{\nu,e} \varphi$, is inductively defined on the structure of φ: $(E, \alpha) \models_{\nu,e}$ *true* always holds; $(E, \alpha) \models_{\nu,e} p(t_1, \ldots, t_n)$ for $p \in P_{obs}$ if and

[1] If A and B are two sets, we denote by B^A the set of all mappings from A to B.

only if $(\nu_e(t_1), \ldots, \nu_e(t_n)) \in p^{\mathcal{A}}$; $(E, \alpha) \models_{\nu,e} p(t_1, \ldots, t_n)$ for $p \in P_s$ if and only if $(\nu_e(t_1), \ldots, \nu_e(t_n)) \in \pi_p \circ \pi_s(e)$; $(E, \alpha) \models_{\nu,e} [t]\psi$ if and only if for all $e' \in E$ such that $\nu_e(t) = e'$, $(E, \alpha) \models_{\nu,e'} \psi$. The other modalities can be defined as derived notions. Actually, we have the following elementary equivalences:[2] $\langle t\rangle\varphi \equiv \neg[t]\neg\varphi$; $[t*]\varphi \equiv \varphi \wedge [t][t*]\varphi$; $[\{t_1, \ldots, t_n\}]\varphi \equiv [t_1]\varphi \wedge \ldots \wedge [t_n]\varphi$. Boolean connectives are interpreted as usual. (E, α) *validates* a formula φ, denoted by $(E, \alpha) \models \varphi$, if and only if for every interpretation $\nu : V \to A$ and every state $e \in E$, $(E, \alpha) \models_{\nu,e} \varphi$. Given $\Psi \subseteq \mathit{For}(\Sigma)$ and two Σ-models (E, α) and (E', α'), (E, α) is *Ψ-equivalent to* (E', α'), denoted by $(E, \alpha) \equiv_\Psi (E', \alpha')$, if and only if we have: $\forall \varphi \in \Psi$, $(E, \alpha) \models \varphi \Leftrightarrow (E', \alpha') \models \varphi$. Given a specification $Sp = (\Sigma, Ax)$, a Σ-model (E, α) is an *Sp-model* if for every $\varphi \in Ax$, $(E, \alpha) \models \varphi$. $\mathit{Mod}(Sp)$ is the full subcategory of $\mathit{Mod}(\Sigma)$, objects of which are all Sp-models. A Σ-formula φ is a *semantic consequence* of a specification $Sp = (\Sigma, Ax)$, denoted by $Sp \models \varphi$, if and only if for every Sp-model (E, α), we have $(E, \alpha) \models \varphi$. $Sp^\bullet$ is the set of all semantic consequences.

Calculus. A calculus for quantifier-free modal CoCasl specifications is defined by the following inference rules, where $\Gamma \vdash\!\!\sim \Delta$ is a sequent such that Γ and Δ are two sets of Σ-formulae:

$$\frac{}{\Gamma,\varphi \vdash\!\!\sim \Delta,\varphi}\,\mathsf{Taut} \qquad \frac{\Gamma \vdash\!\!\sim \Delta \in Sp}{\Gamma \vdash\!\!\sim \Delta}\,\mathsf{Ax} \qquad \frac{\Gamma \vdash\!\!\sim \Delta,\varphi}{\Gamma,\neg\varphi \vdash\!\!\sim \Delta}\,\mathsf{Left}\text{-}\neg \qquad \frac{\Gamma,\varphi \vdash\!\!\sim \Delta}{\Gamma \vdash\!\!\sim \Delta,\neg\varphi}\,\mathsf{Right}\text{-}\neg$$

$$\frac{\Gamma,\varphi,\psi \vdash\!\!\sim \Delta}{\Gamma,\varphi\wedge\psi \vdash\!\!\sim \Delta}\,\mathsf{Left}\text{-}\wedge \qquad \frac{\Gamma \vdash\!\!\sim \Delta,\varphi \quad \Gamma \vdash\!\!\sim \Delta,\psi}{\Gamma \vdash\!\!\sim \Delta,\varphi\wedge\psi}\,\mathsf{Right}\text{-}\wedge \qquad \frac{\Gamma \vdash\!\!\sim \Delta,\varphi \quad \Gamma,\psi \vdash\!\!\sim \Delta}{\Gamma,\varphi\Rightarrow\psi \vdash\!\!\sim \Delta}\,\mathsf{Left}\text{-}\Rightarrow$$

$$\frac{\Gamma,\varphi \vdash\!\!\sim \Delta \quad \Gamma,\psi \vdash\!\!\sim \Delta}{\Gamma,\varphi\vee\psi \vdash\!\!\sim \Delta}\,\mathsf{Left}\text{-}\vee \qquad \frac{\Gamma \vdash\!\!\sim \Delta,\varphi,\psi}{\Gamma \vdash\!\!\sim \Delta,\varphi\vee\psi}\,\mathsf{Right}\text{-}\vee \qquad \frac{\Gamma,\varphi \vdash\!\!\sim \Delta,\psi}{\Gamma \vdash\!\!\sim \Delta,\varphi\Rightarrow\psi}\,\mathsf{Right}\text{-}\Rightarrow$$

$$\frac{\Gamma \vdash\!\!\sim \varphi}{[t]\Gamma \vdash\!\!\sim [t]\varphi}\,\mathsf{Nec} \qquad \frac{\Gamma \vdash\!\!\sim \Delta}{\sigma(\Gamma) \vdash\!\!\sim \sigma(\Delta)}\,\mathsf{Subs} \qquad \frac{\Gamma \vdash\!\!\sim \Delta,\varphi \qquad \Gamma',\varphi \vdash\!\!\sim \Delta'}{\Gamma,\Gamma' \vdash\!\!\sim \Delta,\Delta'}\,\mathsf{Cut}$$

where $[t]\Gamma = \{[t]\gamma \mid \gamma \in \Gamma\}$, $\langle t\rangle\Gamma = \{\langle t\rangle\gamma \mid \gamma \in \Gamma\}$ and $\sigma(\Gamma) = \{\sigma(\gamma) \mid \gamma \in \Gamma\}$. This calculus is the standard Gentzen sequent calculus for modal logic **K** which underlies CoCasl's logic, from which we removed the axiom scheme called Kripke distribution axiom: $[t](\varphi \Rightarrow \psi) \Rightarrow ([t]\varphi \Rightarrow [t]\psi)$, since it is of no interest for our unfolding procedure. From rule Nec, we can derive the following rules, which will be helpful later:

$$\frac{\Gamma \vdash\!\!\sim \varphi}{[t*]\Gamma \vdash\!\!\sim [t*]\varphi}\,\mathsf{Nec*} \qquad \frac{\Gamma \vdash\!\!\sim \varphi}{[\{t_1,\ldots,t_n\}]\Gamma \vdash\!\!\sim [\{t_1,\ldots,t_n\}]\varphi}\,\mathsf{Nec}^n$$

$$\frac{\Gamma \vdash\!\!\sim \varphi,\Delta}{[t]\Gamma \vdash\!\!\sim [t]\varphi,\langle t\rangle\Delta} \qquad \frac{\Gamma,\varphi \vdash\!\!\sim \Delta}{[t]\Gamma,\langle t\rangle\varphi \vdash\!\!\sim \langle t\rangle\Delta} \qquad \frac{\Gamma \vdash\!\!\sim \Delta}{[t]\Gamma \vdash\!\!\sim \langle t\rangle\Delta}$$

In order to manipulate less complex formulae, we take advantage of the fact that the inference rules associated to Boolean connectives define an automatic process that allows to transform any sequent $\vdash\!\!\sim \varphi$, where φ is a modal formula, into a set of sequents $\Gamma \vdash\!\!\sim \Delta$ where every formula in Γ and Δ is of the form

[2] Two formulae φ and ψ are said elementarily equivalent, denoted by $\varphi \equiv \psi$, if and only if for each Σ-model (E, α), for each interpretation ν and every state e, $(E, \alpha) \models_{\nu,e} \varphi \Leftrightarrow (E, \alpha) \models_{\nu,e} \psi$.

$\alpha_1 \dots \alpha_n \psi$, where $\alpha_i \in M_\Sigma(V)$ for all i, $1 \leq i \leq n$, and $\psi \in For(\Sigma)$ is a formula not beginning with a modality. Let us call such sequents *normalised sequents*.

More precisely, these normalised sequents are obtained by eliminating every boolean connectives which is not in the scope of a modal operator with the help of the above sequent calculus. Such a syntactic transformation can be done since the inference rules associated to boolean connectives are reversible: given an inference rule $\frac{\varphi_1 \cdots \varphi_n}{\varphi}$ amongst {Left-@, Right-@} where $@ \in \{\neg, \wedge, \vee, \Rightarrow\}$, we have $\bigwedge_{1 \leq i \leq n} \varphi_i \equiv \varphi$. Then, applying reversed inference rules for boolean connectives to any sequent leads to an equivalent set of normalised sequents, which allows to only deal with normalised sequents. Therefore, in the following, we will suppose that specification axioms are normalised sequents. These transformations enable us to remove the rules associated to boolean connectives from the unfolding procedure.

Example 1 (Running Example)
To illustrate our approach, we continue here the specification of the List cotype. We specify two additional observers $odd : List \to List$ and $even : List \to List$ which give a list containing all the elements occurring in oddly numbered places of the original list, in evenly numbered places respectively. We have the following modal axioms:[3]

- $head = n \Leftrightarrow \langle odd \rangle head = n$
- $[odd][tail]\varphi \Leftrightarrow [tail][tail][odd]\varphi$
- $[even]\varphi \Leftrightarrow [tail][odd]\varphi$

We don't specify the data part here, since we are only interested in specifying the process part. Axioms are then transformed into normalised sequents, as explained above. For example, the first axiom $head = n \Leftrightarrow \langle odd \rangle head = n$, which is equivalent to the formula $head = n \Rightarrow \langle odd \rangle head = n \wedge \langle odd \rangle head = n \Rightarrow head = n$, leads to the two sequents $head = n \mathrel{|\!\sim} \langle odd \rangle head = n$ and $\langle odd \rangle head = n \mathrel{|\!\sim} head = n$.

1. $head = n \mathrel{|\!\sim} \langle odd \rangle head = n$
2. $\langle odd \rangle head = n \mathrel{|\!\sim} head = n$
3. $[odd][tail]\varphi \mathrel{|\!\sim} [tail][tail][odd]\varphi$
4. $[tail][tail][odd]\varphi \mathrel{|\!\sim} [odd][tail]\varphi$
5. $[even]\varphi \mathrel{|\!\sim} [tail][odd]\varphi$
6. $[tail][odd]\varphi \mathrel{|\!\sim} [even]\varphi$

Due to lack of space, we don't give a more complex and larger example here, but another example dealing with the CoCasl's modal specification of a cash machine may be found in the long version of this paper [18].

2 Testing from Logical Specifications

The work presented in Section 3 comes within the general framework of testing from formal specifications defined in [3]. So that the paper is as self-contained as possible, we succinctly introduce this framework and we instantiate it to the CoCasl's formalism presented in Section 1.

[3] The second and third axioms actually are axiom schemes, i.e. they denote the sets of all their instances with any formula substituted for φ.

Following previous works [1,3,7,9,19], given a specification $Sp = (\Sigma, Ax)$, the basic assumption is that the system under test can be assimilated to a model of the signature Σ. Test cases are then Σ-formulae which are semantic consequences of the specification Sp (i.e. elements of $Sp^\bullet$). As these formulae are to be submitted to the system, test case interpretation is defined in terms of formula satisfaction. When a test case is submitted to a system, it has to yield a verdict (success or failure). Hence, test cases have to be directly interpreted as "true" or "false" by a computation of the system. Obviously, systems can't deal with formulae containing non-instantiated variables, so test cases have to be ground formulae, that is formulae where all variables have been replaced with actual values. These "executable" formulae are called *observable*. Then a test case is any observable semantic consequence. If we denote by $Obs \subseteq For(\Sigma)$ the set of observable formulae, then a *test set* T is any subset of $Sp^\bullet \cap Obs$. Since the system under test is considered to be a Σ-model P, T is said to be *successful* for P if and only if $\forall \varphi \in T, P \models \varphi$.

The interpretation of test cases submission as a success or failure is related to the notion of system correctness. Following an observational approach [20], to be qualified as correct with respect to a specification Sp, a system is required to be observationally equivalent to a model of $Mod(Sp)$ up to the observable formulae of Obs, that is, they have to validate exactly the same observable formulae.

Definition 1 (Correctness). *P is* correct *for Sp via Obs, denoted by $Correct_{Obs}(P, Sp)$, if and only if there exists a model $\mathcal{M}$ in $Mod(Sp)$ such that $\mathcal{M} \equiv_{Obs} P$.*[4]

A test set allowing to establish the system correctness is said *exhaustive*. Formally, an exhaustive set is defined as follows:

Definition 2 (Exhaustive test set). *Let $\mathcal{K} \subseteq Mod(\Sigma)$. A test set T is* exhaustive *for $\mathcal{K}$ with respect to Sp and Obs if and only if*

$$\forall P \in \mathcal{K}, P \models T \iff Correct_{Obs}(P, Sp)$$

The existence of an exhaustive test set means that systems belonging to the class $\mathcal{K}$ are testable with respect to Sp via Obs, since correctness can be asymptotically approached by submitting a (possibly infinite) test set. Hence, an exhaustive test set is appropriate to start the process of selecting test sets. However, such an exhaustive set does not necessarily exist, depending on the nature of both specifications and systems (whence the usefulness of subclass $\mathcal{K}$ of systems in Definition 2), and on the chosen set of observable formulae. For instance, we will need here to assume that the system under test is reachable on data. Among all the test sets, the biggest one is the set $Sp^\bullet \cap Obs$ of observable semantic consequences of the specification. Hence, to start the selection phase of the testing process, we first have to show that $Sp^\bullet \cap Obs$ is exhaustive. This holds for every system reachable on data as stated by Theorem 1.

[4] Equivalence of Σ-models with respect to a set of formulae is defined in Section 1.

Theorem 1. *Let $Sp = (\Sigma, Ax)$ be a specification. Then the test set $Sp^\bullet \cap Obs$ is exhaustive for every model reachable on data.*

Idea of the proof. Considering a system $\mathcal{S}$ reachable on data, we use classic results of the coalgebra theory [21] to build a final coalgebra elementary equivalent to $\mathcal{S}$ with respect to Obs, and then show that a well-chosen subcoalgebra of it (also elementary equivalent to $\mathcal{S}$ up to Obs) is a model of specification Sp.

The entire proof may be found in [18]. □

The challenge, when dealing with specifications defined as logical theories, consists in managing the size of $Sp^\bullet \cap Obs$, which is most of the time infinite. In practice, experts apply some selection criteria in order to extract a set of test cases of sufficiently reasonable size to be submitted to the system. The underlying idea is that all test cases satisfying a considered selection criterion reveal the same class of incorrect systems, intuitively those corresponding to the fault model captured by the criterion. For example, the criterion called *uniformity hypothesis* states that test cases in a test set all have the same power to make the system fail.

A classic way to select test data with a selection criterion C consists in splitting a given starting test set T into a family of test subsets $\{T_i\}_{i \in I_{C(T)}}$ such that $T = \cup_{i \in I_{C(T)}} T_i$ holds. A test set satisfying such a selection criterion simply contains at least one test case for each non-empty subset T_i. The selection criterion C is then a coverage criterion according to the way C is splitting the initial test set T into the family $\{T_i\}_{i \in I_{C(T)}}$. This is the method that we will use in this paper to select test data, known under the term of *partition testing*.

Definition 3 (Selection criterion). *A* selection criterion *C is a mapping $\mathcal{P}(Sp^\bullet \cap Obs) \to \mathcal{P}(\mathcal{P}(Sp^\bullet \cap Obs))$.*[5] *For a test set T, we note $|C(T)| = \cup_{i \in I_{C(T)}} T_i$ where $C(T) = \{T_i\}_{i \in I_{C(T)}}$. T'* satisfies *C* applied to *T, noted by $T' \sqsubset C(T)$ if and only if: $\forall i \in I_{C(T)}, T_i \neq \emptyset \Rightarrow T' \cap T_i \neq \emptyset$.*

To be pertinent, a selection criterion should ensure some properties between the starting test set and the resulting family of test sets:

Definition 4 (Properties). *Let C be a selection criterion and T be a test set. C is said* sound for *T if and only if $|C(T)| \subseteq T$. C is said* complete for *T if and only if $|C(T)| \supseteq T$.*

These properties are essential for an adequate selection criterion: soundness ensures that test cases will be selected within the starting test set (i.e. no test is added) while completeness ensures that no test from the initial test set is lost. A sound and complete selection criterion then preserves exactly all the test cases of the initial test set, up to the notion of equivalent test cases.

[5] For a given set X, $\mathcal{P}(X)$ denotes the powerset of X.

3 Selection Criteria Based on Axiom Unfolding

In this section, we study the problem of test case selection for quantifier-free modal CoCasl specifications, by adapting a selection criteria based on unfolding of quantifier-free first-order formulae recently defined in the first-order specifications setting [17].

3.1 Test Sets for Modal CoCasl Formulae

We recall here general definitions of test sets from [17]. The selection method that we are going to define takes inspiration from classic methods that split the initial test set of any formula considered as a test purpose.

Succinctly, for a modal CoCasl formula φ, our method consists in splitting the initial test set for φ into many test subsets, called *constrained test sets for* φ, and choosing any input in each non-empty subset. First, let us define what test set and constrained test set for a modal CoCasl formula are.

Definition 5 (Test set). *Let* φ *be a modal formula, called* test purpose. *The* test set for φ, *denoted by* T_φ, *is the set defined as follows:*

$$T_\varphi = \{\rho(\varphi) \mid \rho : V \to T_\Sigma, \rho(\varphi) \in Sp^\bullet \cap Obs\}$$

Note that φ may be any formula, not necessarily in $Sp^\bullet$. When $\varphi \notin Sp^\bullet$ then $T_\varphi = \emptyset$. Constrained test sets will be sets generated by our unfolding procedure. They are defined as follows.

Definition 6 (Constrained test set). *Let* φ *be a modal formula (the test purpose),* $\mathcal{C}$ *be a set of modal formulae called* Σ*-constraints, and* $\sigma : V \to T_\Sigma(V)$ *be a substitution. A* test set for φ with respect to $\mathcal{C}$ and σ, *denoted by* $T_{(\mathcal{C},\sigma),\varphi}$, *is the set of ground formulae defined by:*

$$T_{(\mathcal{C},\sigma),\varphi} = \{\rho(\sigma(\varphi)) \mid \rho : V \to T_\Sigma, \forall\psi \in \mathcal{C}, \rho(\psi) \in Sp^\bullet \cap Obs\}$$

The couple $\langle(\mathcal{C},\sigma),\varphi\rangle$ *is called a* constrained test purpose.

Note that the test purpose φ of Definition 5 can be seen as the constrained test purpose $\langle(\{\varphi\}, Id),\varphi\rangle$.

3.2 Unfolding Procedure

Given a test purpose φ, the unfolding procedure will then replace the initial constrained test purpose $\langle(\{\varphi\}, Id),\varphi\rangle$ with a set of constrained test purposes $\langle(\mathcal{C},\sigma),\varphi\rangle$. This will be achieved by matching (up to unification), formulae in $\mathcal{C}$ for any constrained test purpose $\langle(\mathcal{C},\sigma),\varphi\rangle$ with the specification axioms. Hence, step by step, we will see that the unfolding procedure is building a proof tree of conclusion φ having the following structure :

- no instance of cut occurs over instances of substitution and necessitation
- no instance of substitution occurs over instances of necessitation

– there is no instance of cut with two instances of cut occurring over it.

Hence, the unfolding procedure will only involve cut, substitution and necessitation rules. In order to allow many applications of the necessitation rule at each step of the unfolding procedure, let us define the following relation $\mathcal{R}$ over tuples of modality sequences.

Definition 7. *Let $p, q \in \mathbb{N}$. $\mathcal{R} \subseteq (M_\Sigma(V)^*)^p \times (M_\Sigma(V)^*)^q$ is defined for all $(M_1, \ldots, M_p) \in (M_\Sigma(V)^*)^p$ and $(N_1, \ldots, N_q) \in (M_\Sigma(V)^*)^q$ as follows:*
$(M_1, \ldots, M_p)\mathcal{R}(N_1, \ldots, N_q)$ if and only if

1. *there exists $n \in \mathbb{N}$ such that for all i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, there exists $\alpha^i_1, \ldots, \alpha^i_n$ and $\beta^i_1, \ldots, \beta^i_n$ such that $M_i = \alpha^i_1 \ldots \alpha^i_n$ and $N_j = \beta^j_1 \ldots \beta^j_n$*
2. *for all l, $1 \leq l \leq n$, $\alpha^1_l, \ldots, \alpha^p_l$ and $\beta^1_l, \ldots, \beta^q_l$ are such that:*
   (a) there exists $t \in T_\Sigma(V)$ such that for all i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, α^i_l and β^j_l all equal to $[t]$ or $\langle t \rangle$, or α^i_l and β^j_l all equal to $[t*]$ or $\langle t* \rangle$
   (b) for all i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, $\alpha^i_l = [t]$ and $\beta^j_l = \langle t \rangle$ (resp. $\alpha^i_l = [t*]$ and $\beta^j_l = \langle t* \rangle$), except perhaps either for one k, $1 \leq k \leq p$, such that $\alpha^k_l = \langle t \rangle$ (resp. $\alpha^k_l = \langle t* \rangle$), or for one k, $1 \leq k \leq q$, such that $\beta^k_l = [t]$ (resp. $\beta^k_l = [t*]$).

This relation then ensures the following proposition.

Proposition 1. *Let $\gamma_1, \ldots, \gamma_p \mid\!\sim \delta_1, \ldots, \delta_q$ be any sequent. Let $(M_1, \ldots, M_p) \in (M_\Sigma(V)^*)^p$ and $(N_1, \ldots, N_q) \in (M_\Sigma(V)^*)^q$ such that $(M_1, \ldots, M_p)\mathcal{R}(N_1, \ldots, N_q)$. Then there exists a proof tree of conclusion $M_1\gamma_1, \ldots, M_p\gamma_p \mid\!\sim N_1\delta_1, \ldots, N_q\delta_q$ composed only of instances of the necessitation rule.*

We can now proceed with the presentation of the unfolding procedure. The procedure inputs are:

– a modal CoCasl specification $Sp = (\Sigma, Ax)$ where axioms of Ax have been transformed into normalised sequents (see Section 1)
– a modal formula φ representing the test purpose $\langle(\{\varphi\}, Id), \varphi\rangle$
– a family Ψ of couples $(\mathcal{C}, \sigma)$ where $\mathcal{C}$ is a set of Σ-constraints in the form of normalised sequents, and σ is a substitution $V \to T_\Sigma(V)$.

Test sets for φ with respect to couples $(\mathcal{C}, \sigma)$ are naturally extended to Ψ as follows: $T_{\Psi,\varphi} = \bigcup_{(\mathcal{C},\sigma)\in\Psi} T_{(\mathcal{C},\sigma),\varphi}$. The first set Ψ_0 only contains the couple composed of the set of normalised sequents obtained from the modal formula φ under test and the identity substitution.

The unfolding procedure is expressed by the two following rules:[6]

$$\textbf{Reduce}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{\Gamma \mid\!\sim \Delta\}, \sigma')\}}{\Psi \cup \{(\sigma(\mathcal{C}), \sigma \circ \sigma')\}} \quad \exists\gamma \in \Gamma, \exists\delta \in \Delta \text{ s.t. } \sigma(\gamma) = \sigma(\delta),\ \sigma \text{ mgu}$$

[6] The most general unifier (or mgu) of two terms γ and δ is the most general substitution σ such that $\sigma(\gamma) = \sigma(\delta)$.

$$\textbf{Unfold}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{\phi\}, \sigma')\}}{\Psi \cup \bigcup_{(c,\sigma)\in Tr(\phi)} \{(\sigma(\mathcal{C}) \cup c, \sigma \circ \sigma')\}}$$

where $Tr(\phi)$ for $\phi = \gamma_1, \dots, \gamma_m \mathrel{|\!\!\sim} \delta_1, \dots, \delta_n$ is the set of couples

$$\{(c, \sigma) \mid Cond(c, \sigma)\}$$

where, for each couple, c is the following set

$$\begin{array}{c}\{\sigma(\gamma_{p+1}), \dots, \sigma(\gamma_m), \sigma(N'_i\zeta_i) \mathrel{|\!\!\sim} \sigma(\delta_{q+1}), \dots, \sigma(\delta_n)\}_{1\leq i\leq k} \\ \bigcup \\ \{(\sigma(\gamma_{p+1}), \dots, \sigma(\gamma_m) \mathrel{|\!\!\sim} \sigma(M'_i\xi_i), \sigma(\delta_{q+1}), \dots, \sigma(\delta_n)\}_{1\leq i\leq l}\end{array}$$

and the condition on (c, σ) denoted by $Cond(c, \sigma)$ is the following: there exists an axiom $\psi_1, \dots, \psi_p, \xi_1, \dots, \xi_l \mathrel{|\!\!\sim} \zeta_1, \dots, \zeta_k, \varphi_1, \dots, \varphi_q \in Ax$ with $k, l \in \mathbb{N}$, $1 \leq p \leq m$ and $1 \leq q \leq n$, and there exists a unifier σ such that

- for all $1 \leq i \leq p$, there exists $M_i \in M_\Sigma(V)^*$ such that $\sigma(M_i\psi_i) = \sigma(\gamma_i)$,
- for all $1 \leq i \leq q$, there exists $N_i \in M_\Sigma(V)^*$ such that $\sigma(N_i\varphi_i) = \sigma(\delta_i)$,
- for all $1 \leq i \leq l$ and for all $1 \leq j \leq k$, $M'_i, N'_j \in M_\Sigma(V)^*$ with $(M_1, \dots, M_p, M'_1, \dots, M'_l)\mathcal{R}(N_1, \dots, N_q, N'_1, \dots, N'_k)$

The **Reduce** rule eliminates tautologies from constraints sets (up to substitution), which are without interest for the unfolding procedure. The **Unfold** rule is closely related to the one given in [17] although more complicated because of modalities. Roughly speaking, this rule consists in replacing the formula ϕ with the set c of constraints, ϕ being the conclusion of an instance of the Cut rule, and the constraints in c being the premisses of this rule instance which do not directly come from a substitution (up to some applications of the necessitation rule) of an axiom of the specification. The relevance of the method is due to the fact that testing $\sigma(\phi)$ comes to test the formulae of c, which will be proved in the next subsection. The particular case where no formula has to be cut is taken into account, since k and l may be equal to zero. $Tr(\phi)$ is then a couple $(\emptyset, \sigma)$, and it is the last step of unfolding for this formula.

Each unification with an axiom leads to as much couples (c, σ) as there are ways to instantiate $M'_1, \dots, M'_l$ and $N'_1, \dots, N'_k$ so that $(M_1, \dots, M_p, M'_1, \dots, M'_l)$ and $(N_1, \dots, N_q, N'_1, \dots, N'_k)$ belong to $\mathcal{R}$. So the initial formula ϕ is replaced with, at least, as much sets of formulae as there are axioms to which it can be unified. The definition of $Tr(\phi)$ being based on unification, this set is computable if the specification Sp has a finite set of axioms. Therefore, given an atomic formula ψ, we have the selection criterion C_ψ that maps any $T_{(\mathcal{C},\sigma'),\varphi}$ to $(T_{(\sigma(\mathcal{C}\setminus\{\phi\})\cup c,\sigma\circ\sigma'),\varphi})_{(c,\sigma)\in Tr(\phi)}$ if $\phi \in \mathcal{C}$, and to $T_{\mathcal{C},\varphi}$ otherwise.

We write $\langle\Psi, \varphi\rangle \vdash_U \langle\Psi', \varphi\rangle$ to mean that Ψ' can be derived from Ψ by applying **Reduce** or **Unfold**. An unfolding procedure is then any program, whose inputs are a CoCasl's modal specification Sp and a modal formula φ, and uses the above inference rules to generate the sequence $\langle\Psi_0, \varphi\rangle \vdash_U \langle\Psi_1, \varphi\rangle \vdash_U \langle\Psi_2, \varphi\rangle \dots$

Termination of the unfolding procedure is unlikely, since it is not checked, during its execution, whether the formula φ is a semantic consequence of the specification or not. Actually, this will be done during the generation phase, not handled in this paper. As we already explained in the introduction, the aim of the unfolding procedure is not to find the complete proof of formula φ, but to make a partition of T_φ increasingly fine. Hence the procedure can be stopped at any moment, when the obtained partition is fine enough according to the judgement or the needs of the tester. The idea is to stretch further the execution of the procedure in order to make increasingly big proof trees whose remaining lemmas are constraints. If φ is not a semantic consequence of Sp, then this means that, among remaining lemmas, some of them are not true, and then the associated test set is empty.

Example 2 (Lists). Let us suppose that we want to test the formula $[even][tail]head = a \Rightarrow [tail][tail][even]head = b$. Then, to perform the first step of the unfolding procedure on the initial family of couples:

$$\Psi_0 = \{(\{[even][tail]head = a \mathrel{|\!\sim} [tail][tail][even]head = b\}, Id)\}$$

leads to the following family of couples:

$$\begin{array}{lr}
\Psi_1 = \{(\{[even][tail]\langle odd\rangle head = n_0 \mathrel{|\!\sim} [tail][tail][even]head = m_0\}, \sigma_1), & (1)\\
\quad (\{\langle even\rangle[tail]\langle odd\rangle head = n_0 \mathrel{|\!\sim} [tail][tail][even]head = m_0\}, \sigma_1), & (1)\\
\quad (\{[even]\langle tail\rangle\langle odd\rangle head = n_0 \mathrel{|\!\sim} [tail][tail][even]head = m_0\}, \sigma_1), & (1)\\
\quad (\{\langle even\rangle\langle tail\rangle\langle odd\rangle head = n_0 \mathrel{|\!\sim} [tail][tail][even]head = m_0\}, \sigma_1), & (1)\\
\quad (\{[tail][odd][tail]head = n_0 \mathrel{|\!\sim} [tail][tail][even]head = m_0\}, \sigma_1), & (5)\\
\quad (\{[even][tail]head = n_0 \mathrel{|\!\sim} [tail][tail][even]\langle odd\rangle head = m_0\}, \sigma_2), & (2)\\
\quad (\{[even][tail]head = n_0 \mathrel{|\!\sim} [tail][tail][tail][odd]head = m_0\}, \sigma_2)\} & (6)
\end{array}$$

where $\sigma_1 : a \mapsto n_0, b \mapsto m_0, n \mapsto n_0$ and $\sigma_2 : a \mapsto n_0, b \mapsto m_0, n \mapsto m_0$. Each couple of Ψ_1 is labelled by the number of the axiom used for the unfolding of the initial formula.

The first four couples of Ψ_1 come from the unification of the initial formula with axiom (1). Since $\sigma_1(M_1\psi_1) = \sigma_1(\gamma_1)$, where $M_1 = [even][tail]$, ψ_1 is the formula $head = n$ and γ_1 is $head = a$, the resulting constraints are the sequents $\sigma_1(N_1'\zeta_1) \mathrel{|\!\sim} \sigma_1(\delta_1)$ where ζ_1 is the formula $\langle odd\rangle head = n$, δ_1 is $[tail][tail][even]head = b$, and N_1' must be such that $M_1 \mathcal{R} N_1$. According to the definition of $\mathcal{R}$, several N_1 suit, namely $[even][tail]$, $\langle even\rangle[tail]$, $[even]\langle tail\rangle$ and $\langle even\rangle\langle tail\rangle$, whence the four constraints generated by the unification with axiom (1).

Notice that the formula under test is a consequence of the specification if and only if $a = b$. The unfolding may then generate two kinds of constrained test sets: those whose substitution σ is such that $\sigma(a) = \sigma(b)$, which will lead to test cases since they are consequences of the specification, and those where $\sigma(a) \neq \sigma(b)$, which are not test cases. Here, when a constraint is unified with both sides of axiom (1) or (2), the substitution collapses a and b and the resulting constrained test set is a potential test case.

The unfolding procedure can not distinguish between these two kinds of constrained test sets, but this distinction will be done before submitting them to the

system, by applying a ground substitution ρ to any formula in constrained test purposes. Since, by definition, $\rho(\psi)$ has to be a consequence of the specification, constrained test sets where $\sigma(a) \neq \sigma(b)$ will not be submitted to the system.

The application of the procedure on another example may be found in [18].

Until now, the unfolding procedure has been defined in order to cover the behaviours of one test purpose, represented by the formula φ. When we are interested in covering more widely the exhaustive set $Sp^\bullet \cap Obs$, a strategy consists in ordering modal formulae with respect to their size, as follows:

$$\Phi_0 = \{ \mid\!\sim p(x_1, \ldots, x_n) \mid p : s_1 \times \ldots \times s_n \in P, \forall i, 1 \leq i \leq n, x_i \in V_{s_i} \}$$

$$\Phi_{n+1} = \{ \mid\!\sim \neg\psi,\ \mid\!\sim [m]\psi,\ \mid\!\sim \psi_1 @ \psi_2 \mid m \in M_\Sigma(V), @ \in \{\wedge, \vee, \Rightarrow\}, \psi, \psi_1, \psi_2 \in \Phi_n \}$$

Then, to manage the size (often infinite) of $Sp^\bullet \cap Obs$, we start by choosing $k \in \mathbb{N}$, and then we apply for every i, $1 \leq i \leq k$, the above unfolding procedure to each formula belonging to Φ_i. Of course, this requires that signatures are finite so that each set Φ_i is finite too.

3.3 Soundness and Completeness

Here, we prove the two properties that make the unfolding procedure relevant for the selection of appropriate test cases, i.e. that the selection criterion defined by the procedure is sound and complete for the initial test set we defined.

Theorem 2. *If $\langle \Psi, \varphi \rangle \vdash_U \langle \Psi', \varphi \rangle$, then $T_{\Psi,\varphi} = T_{\Psi',\varphi}$.*

Idea of the proof. To prove the soundness of the procedure comes to prove that the initial formula φ can be derived from the constraints replacing it in the procedure. Thus we prove that the test set obtained by the application of the procedure does not add new test cases. Then, to prove the completeness of the procedure, we prove that there necessarily exists a proof tree of conclusion φ having a certain structure, and then that the procedure generates all possible constraints for testing φ. We thus prove that no test cases are lost. As explained just before, we can observe that our unfolding procedure defines a proof search strategy that enables to limit the search space to the class of proof trees having the following structure:

- no instance of cut occurs over instances of substitution and necessitation
- no instance of substitution occurs over instances of necessitation
- there is no instance of cut with two instances of cut occurring over it.

We then have to prove that the derivability defined by our unfolding strategy coincides with the full derivability. We then define basic transformations to rewrite proof trees into ones having the above structure, and show that the induced global proof tree transformation is weakly normalising.

The entire proof may be found in [18]. □

4 Conclusion

In this paper, we have extended the method for selecting test cases known as axiom unfolding to coalgebraic specifications of dynamic systems. As in the algebraic specifications setting, our unfolding procedure consists in dividing the initial test set for a formula into subsets. The generation of a test set for this formula then arises from the selection of one test case in each resulting subset. We have proved this procedure to be sound and complete, so that test cases are preserved at each step. We have also proved the exhaustiveness of the set of observable consequences of the specification for every reachable system, and proposed a strategy to cover this exhaustive test set.

Ongoing research concerns several aspects. First, we have to specialize our unfolding procedure by handling (strong and existential) equality in a more efficient way. We lose here the strong equality, and the advantage of equality being a congruence. Then we have to extend this work to the very recent extension of CoCasl logic [22]. This logic deals with modalities at a more abstract level than the one presented here, using Pattinson's predicate liftings. This extension of CoCasl allows to specify in several modal logics that were not handled with basic CoCasl, such as probabilistic modal logic. Defining testing for such an extension of CoCasl would allow us to handle a larger variety of modal formalisms in our framework. Another important future work will be to include structuration, such as provided by Casl and CoCasl languages, in our framework, both on its first-order side, by extending our work developed in [17], and on its coalgebraic side, by extending the present work. This work will surely take inspiration from [6,23].

References

1. Bernot, G., Gaudel, M.C., Marre, B.: Software testing based on formal specifications: a theory and a tool. Software Engineering Journal 6(6), 387–405 (1991)
2. Gaudel, M.C.: Testing can be formal, too. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 82–96. Springer, Heidelberg (1995)
3. Le Gall, P., Arnould, A.: Formal specification and test: correctness and oracle. In: Haveraaen, M., Dahl, O.-J., Owe, O. (eds.) Recent Trends in Data Type Specification. LNCS, vol. 1130, pp. 342–358. Springer, Heidelberg (1996)
4. Marre, B.: LOFT: a tool for assisting selection of test data sets from algebraic specifications. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 799–800. Springer, Heidelberg (1995)
5. Aiguier, M., Arnould, A., Boin, C., Le Gall, P., Marre, B.: Testing from algebraic specifications: test data set selection by unfolding axioms. In: Grieskamp, W., Weise, C. (eds.) FATES 2005. LNCS, vol. 3997, pp. 203–217. Springer, Heidelberg (2006)
6. Machado, P., Sannella, D.: Unit testing for Casl architectural specifications. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 506–518. Springer, Heidelberg (2002)

7. Arnould, A., Le Gall, P., Marre, B.: Dynamic testing from bounded data type specifications. In: Hlawiczka, A., Simoncini, L., Silva, J.G.S. (eds.) Dependable Computing - EDCC-2. LNCS, vol. 1150, pp. 285–302. Springer, Heidelberg (1996)
8. Aiguier, M., Arnould, A., Le Gall, P.: Exhaustive test sets for algebraic specification correctness. Technical report, IBISC - Université d'Évry-Val d'Essonne (2006)
9. Arnould, A., Le Gall, P.: Test de conformité: une approche algébrique. Technique et Science Informatiques, Test de logiciel 21, 1219–1242 (2002)
10. Mossakowski, T., Schröder, L., Roggenbach, M., Reichel, H.: Algebraic-coalgebraic specification in CoCasl. Journal of Logic and Algebraic Programming 67(1-2), 146–197 (2006)
11. Yannakakis, M., Lee, D.: Testing finite state machines. In: Symposium on Theory of Computing (STOC'91), pp. 476–485. ACM Press, New York (1991)
12. Tretmans, J.: Testing labelled transition systems with inputs and outputs. In: International Workshop on Protocols Test Systems (IWPTS'95) (1995)
13. Rusu, V., du Bousquet, L., Jéron, T.: An approach to symbolic test generation. In: Integrated Formal Methods (IFM '00), pp. 338–357. Springer, Heidelberg (2000)
14. Frantzen, L., Tretmans, J., Willemse, T.: Test generation based on symbolic specifications. In: Grabowski, J., Nielsen, B. (eds.) FATES 2004. LNCS, vol. 3395, pp. 1–15. Springer, Heidelberg (2005)
15. Gaston, C., Le Gall, P., Rapin, N., Touil, A.: Symbolic execution techniques for test purpose definition. In: Uyar, M.Ü., Duale, A.Y., Fecko, M.A. (eds.) TestCom 2006. LNCS, vol. 3964, pp. 1–18. Springer, Heidelberg (2006)
16. Ammann, P., Ding, W., Xu, D.: Using a model checker to test safety properties. In: International Conference on Engineering of Complex Computer Systems (ICECCS'01), pp. 212–221 (2001)
17. Aiguier, M., Arnould, A., Le Gall, P., Longuet, D.: Test selection criteria for quantifier-free first-order specifications. In: Fundamentals of Software Engineering (FSEN'07). Lecture Notes in Computer Science (to appear)
18. Longuet, D., Aiguier, M.: Specification-based testing for CoCasl's modal specifications. Technical report, IBISC - Université d'Évry-Val d'Essonne (2007) Available at http://www.ibisc.fr/~dlonguet/publications_gb.html
19. Bernot, G.: Testing against formal specifications: a theoretical view. In: Abramsky, S. (ed.) TAPSOFT 1991, CCPSD 1991, and ADC-Talks 1991. LNCS, vol. 494, pp. 99–119. Springer, Heidelberg (1991)
20. Hennicker, R., Wirsing, M., Bidoit, M.: Proof systems for structured specifications with observability operators. Theoretical Computer Science 173(2), 393–443 (1997)
21. Rutten, J.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249, 3–80 (2000)
22. Schröder, L., Mossakowski, T.: Coalgebraic modal logic in CoCasl. In: Recent Trends in Algebraic Specification Techniques (WADT'06). LNCS, vol. 4409, pp. 128–142 (2007)
23. Machado, P.: Testing from structured algebraic specifications. In: Rus, T. (ed.) AMAST 2000. LNCS, vol. 1816, pp. 529–544. Springer, Heidelberg (2000)

CIRC: A Circular Coinductive Prover*

Dorel Lucanu[1] and Grigore Roşu[2]

[1] Faculty of Computer Science
Alexandru Ioan Cuza University, Iaşi, Romania
dlucanu@info.uaic.ro
[2] Department of Computer Science
University of Illinois at Urbana-Champaign, USA
grosu@cs.uiuc.edu

Abstract. CIRC is an automated circular coinductive prover implemented as an extension of Maude. The circular coinductive technique that forms the core of CIRC is discussed, together with a high-level implementation using metalevel capabilities of rewriting logic. To reflect the strength of CIRC in automatically proving behavioral properties, an example defining and proving properties about infinite streams of infinite binary trees is shown. CIRC also provides limited support for automated inductive proving, which can be used in combination with coinduction.

1 Introduction

Behavioral abstraction in algebraic specification appears under various names in the literature such as *hidden algebra* [5], *observational logic* [8], *swinging types* [11], *coherent hidden algebra* [2], *hidden logic* [12], and so on. Most of these approaches appeared as a need to extend algebraic specifications to ease the process of specifying and verifying designs of systems and also for various other reasons, such as, to naturally handle infinite types[1], to give semantics to the object paradigm, to specify finitely otherwise infinitely axiomatizable abstract data types, etc. The main characteristic of these approaches is that sorts are split into *visible* (or *observational*) for data and *hidden* for states, and the equality is behavioral, in the sense that two states are *behaviorally equivalent* if and only if they *appear* to be the same under any visible *experiment*.

Coalgebraic *bisimulation* [9] and *context induction* [7] are sound proof techniques for behavioral equivalence. Unfortunately, both need human intervention: coinduction to pick a "good" bisimulation relation, and context induction to invent and prove lemmas. *Circular coinduction* [3,12] is an automatic proof technique for behavioral equivalence. By circular coinduction one can prove, for example, the equality $\mathit{zip}(\mathit{zeros}, \mathit{ones}) = \mathit{blink}$ on streams as follows (zeros is the stream 0^ω, ones is 1^ω, blink is $(01)^\omega$, zip merges two streams):

* This work is partially supported by the CNCSIS project 1162/2007, by grants NSF CCF-0448501 and NSF CNS-0509321, and by Microsoft Research gifts.

[1] I.e., types whose values are infinite structures.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 372–378, 2007.
© Springer-Verlag Berlin Heidelberg 2007

1. check that the two streams have the same head, 0;
2. take the tail of the two streams and generate the new goal $zip(ones, zeros) =$ *1:blink*; this becomes the next task;
3. check that the two new streams have the same head, 1;
4. take the tail of the two new streams; after simplification one gets the new goal $zip(zeros, ones) = blink$, which is nothing but the original proof task;
5. conclude that $zip(zeros, ones) = blink$ holds.

The intuition for the above "proof" is that the two streams have been exhaustively tried to be distinguished by iteratively checking their heads and taking their tails. Ending up in circles (we obtained the same new proof task as the original one) means that the two streams are indistinguishable, so equal.

Circular coinduction can be explained and proved correct by reducing it to either bisimulation or context induction: it iteratively constructs a bisimulation, but it also discovers all lemmas needed by a context induction proof. Since the behavioral equivalence problem is Π_2^0-complete even for streams [13], there is *no* algorithm complete for behavioral equality in general, as well as *no* algorithm complete for inequality of streams. Therefore, the best we can do is to focus our efforts on exploring heuristics or deduction rules to prove or disprove equalities of streams that *work well on examples of interest* rather than in general.

BOBJ [3,12] was the first system supporting circular coinduction. Hausmann, Mossakowski, and Schröder [6] also developed circular coinductive techniques and tactics in the context of CoCASL. In this paper we propose CIRC, an automated circular coinductive prover implemented in Full Maude as a behavioral extension of the Maude system [1], making heavy use of meta-level and reflection capabilities of rewriting logic. Maude is by now a very mature system, with many uses, a high-performance rewrite engine, and a broad spectrum of analysis tools. Maude's current meta-level capabilities were not available when the BOBJ system was developed; consequently, BOBJ was a heavy system, with rather poor parsing and performance. By allowing the entire Maude system visible to the user, CIRC inherits all Maude's uses, performance and analysis tools.

CIRC implements the *circularity principle*, which generalizes circular coinductive deduction [4] and can be informally described as follows (a formalization of this principle, together with the related technical details will be discussed elsewhere). Assume that each equation of interest (to be proved) e admits a *frozen form* $fr(e)$, which is an equation associated to e defined possibly over an extended signature, and a set $Der(e)$ of equations, also defined possibly over an extended signature, called its *derivatives*. The circularity principle requires that the following rule be valid: if from the hypotheses $\mathcal{H}$ together with $fr(e)$ we can deduce $Der(e)$, then e is a consequence of $\mathcal{H}$. When $fr(e)$ freezes the equation at the top as in [4], the circularity principle becomes circular coinduction. Interestingly, when the equation is frozen at the bottom on a variable, then it becomes a structural induction (on that variable) derivation rule. This way, CIRC supports both coinduction and induction as projections of a more general principle. In this paper, we concentrate only on CIRC's coinductive capabilities.

2 Behavioral Algebraic Specifications

We assume the reader familiar with algebraic specification and Maude [1]. Here we intuitively present behavioral algebraic specification using an example. Infinite binary trees can be specified using three *behavioral operations* (observers): `node(`T`)` returning the information from the root of T, `left(`T`)` and `right(`T`)` returning the left child and the right child of T, respectively. Examples of experiments for the sort `BTree` of infinite binary trees are `node(*:BTree)`, `node(left(*:BTree))`, `node(right(*:BTree))`, and so on. The sort `BTree` is called *hidden*, and the sort `Elt` (i.e., the result sort of `node`) is called *visible* w.r.t. `BTree`. The result sort of the experiments is always visible. We next define a "mirror" operation over infinite binary trees. We further consider streams of infinite binary trees. A stream is specified by two behavioral operations: `hd(`S`)` returning the first element of stream S (in our case an infinite binary tree), and `tl(`S`)` returning the stream obtained by removing the first element of S. Examples of stream experiments are `hd(*:TStream)`, `hd(tl(*:TStream))`, `hd(tl(tl(*:TStream)))`, and so on. Note that `BTree` is visible w.r.t. `TStream`. We define the following stream operations: a constructor `cons(`T, S`)` inserting the tree T in front of S, $blink(T_1, T_2)$ defining the stream $(T_1, T_2, T_1, T_2, \ldots)$, and $const(T)$ defining the constant stream $(T, T, T, \ldots)$. These infinite data structures can be defined in Full Maude as follows:

```
(th TSTREAM is
  sorts Elt BTree TStream .
  var S : TStream . var E : Elt .
  vars T T1 T2 : BTree .
  ops left right : BTree -> BTree .
  op node : BTree -> Elt .
  op mirror : BTree -> BTree .
  eq left(mirror(T)) =
     mirror(right(T)) .
  eq right(mirror(T)) =
     mirror(left(T)) .
  eq node(mirror(T)) = node(T) .
  op hd : TStream -> BTree .
  op tl : TStream -> TStream .
  op cons : BTree TStream -> TStream .
  eq hd(cons(T, S)) = T .
  eq tl(cons(T, S)) = S .
  op blink : BTree BTree -> TStream .
  eq hd(blink(T1, T2)) = T1 .
  eq tl(blink(T1, T2)) =
     cons(T2, blink(T1,T2)) .
  op const : BTree -> TStream .
  eq hd(const(T)) = T .
  eq tl(const(T)) = const(T) .
endth)
```

Trees T_1, T_2 are *behaviorally equivalent* iff `node(`T_1`)` $=$ `node(`T_2`)`, `node(left(`T_1`))` $=$ `node(left(`T_2`))`, `node(right(`T_1`))` $=$ `node(right(`T_2`))`, and so on. Similarly, streams S_1 and S_2 are behaviorally equivalent iff `hd(`S_1`)` $=$ `hd(`S_2`)`, `hd(tl(`S_1`))` $=$ `hd(tl(`S_2`))`, `hd(tl(tl(`S_1`)))` $=$ `hd(tl(tl(`S_2`)))`, and so on. For instance, `mirror(mirror(`T`))` and T are behaviorally equivalent. We write `TSTREAM` $\models$ `mirror(mirror(`T`))` $= T$. Also, `TSTREAM` $\models$ `blink(`S, S`)` $= S$.

Hence, a *behavioral specification* is an algebraic specification together with a specified subset of *behavioral operators*, which are used to define the crucial notion of *behavioral equivalence* as "indistinguishability under experiments."

3 CIRC

We here describe the underlying proving technique of CIRC and show how one can use the system. The CIRC prover can be downloaded from its website at [10].

As already mentioned, CIRC implements what we call "the principle of circularity," which generalizes both structural induction and circular coinduction, and which will be discussed in more detail elsewhere. We here only focus on its coinductive instance. Circular coinduction [3,12,4] is a sound proof calculus for $\models\!\!\!\equiv$, which can be defined as an instance of the principle of circularity as follows:

- for any sort s, let us extend the signature with a new operation $fr : s \to \texttt{new}$, where `new` is a new sort;
- for each equation e of the form "$(\forall X)\ t = t'$ `if` c," let
  - $fr(e)$ be the equation "$(\forall X) fr(t) = fr(t')$ `if` c," which we call the "frozen" form of e; and
  - $Der(e)$ be the set of equations $\{(\forall X)\ fr(\delta[t/*{:}h]) = fr(\delta[t'/*{:}h])\ \texttt{if}\ c \mid \delta$ behavioral for $h = sort(t)\}$, which we call the "derivatives" of e.

We say that $fr(e)$ "freezes e at the top" in the context of coinduction; this freezing operation ensures the sound use of the coinduction hypotheses in the procedure below, because it prevents the application of congruence inference steps [12,4]. We take the liberty to also call $fr(e)$ *visible* when e is visible.

Any automatic proving procedure based on circular coinduction is parametric in a procedure for equational deduction. In CIRC we use the standard rewriting-based semi-decision procedure to derive equations "$(\forall X)\ t = t'$ `if` c": add the variables X as constants, then add the conditions in c to the set of equations, and then reduce t, t' to their normal forms $nf(t)$ and $nf(t')$, respectively, orienting all the equations into rewrite rules. In what follows we let $\mathcal{E} \vdash e$ denote the fact that e can be deduced from $\mathcal{E}$ using this standard approach ($\mathcal{E}$ is any set of equations). By $\mathcal{E} \not\vdash e$ we mean "knowingly incapable of proving it," that is, that the rewrite engine reduced the two terms to normal forms, but those were not equal. Obviously, this does not necessarily mean that the equation is not true.

Suppose that $\mathcal{B}$ is a behavioral specification whose equations form a set E and that e is an equation to prove. CIRC takes $\mathcal{B}$ and e as input and aims at proving $\mathcal{B} \models\!\!\!\equiv e$. CIRC maintains and iteratively reduces a pair of sets of equations of the form $(\mathcal{E}, \mathcal{G})$, where $\mathcal{E}$ is the set of equations that are assumed to hold and $\mathcal{G}$ is the set of "goals," that is, the set of equations that still need to be proved. Therefore, CIRC aims at reducing the pair $(E, fr(e))$ to a pair of the form $(\mathcal{E}', \emptyset)$, i.e., one whose set of goals is empty. If successful, then $\mathcal{B} \models\!\!\!\equiv e$; moreover, (the unfrozen variants of) all the equations in $\mathcal{E}'$ are behavioral consequences of $\mathcal{B}$. While trying to perform its task, CIRC's procedure can also fail, in which case we conclude that it could not prove $\mathcal{B} \models\!\!\!\equiv e$, or it can run forever. Here are the reduction rules currently supported by CIRC:

[Equational Reduction] :

$(\mathcal{E}, \mathcal{G} \cup \{fr(e)\}) \Rightarrow (\mathcal{E}, \mathcal{G}) \quad \text{if } \mathcal{E} \vdash fr(e)$

[Coinduction Failure] :

$(\mathcal{E}, \mathcal{G} \cup \{fr(e)\}) \Rightarrow fail \quad \text{if } \mathcal{E} \not\vdash fr(e) \text{ and } e \text{ is visible}$

[Circular Coinduction] :

$(\mathcal{E}, \mathcal{G} \cup \{fr(e)\}) \Rightarrow (\mathcal{E} \cup \{nf(fr(e))\}, \mathcal{G} \cup Der(e)) \text{ if } \mathcal{E} \not\vdash fr(e) \text{ and } e \text{ is hidden.}$

$nf(e)$ denotes the equation e whose left-hand and right-hand sides are reduced to normal forms. [Equational Reduction] removes a goal if it can be proved using ordinary equational reduction. [Coinduction Failure] says that the procedure fails whenever it finds a visible goal which cannot be proved using ordinary equational reduction. Finally, [Circular Coinduction] implements the circularity principle: when a hidden equation cannot be proved using ordinary equational reduction, its frozen form (its normal form is an equivalent variant) is added to the specification and its derivatives are added to the set of goals.

The termination of the CIRC procedure above, i.e., reaching of a configuration of the form $(\mathcal{E}', \emptyset)$, is not guaranteed. Since the behavioral entailment problem is Π_2^0 [12,13], we know that there can be no procedure to decide behavioral equalities or inequalities in general.

The remainder of this section shows, by means of our example with streams of infinite trees, how CIRC can be used in practice. Since a behavioral specification includes more information than a usual Full Maude specification, we designed an interface allowing the user to introduce behavioral operations, case sentences, goals, and CIRC commands. A typical scenario of using CIRC is as follows. The objects whose behavioral properties are investigated are described using Full Maude specifications, like the module `TSTREAM` given above. Note that a Full Maude specification does not explicitly say which are the behavioral operations. Therefore we consider a new type of module, delimited by the keywords `cmod` and `endcm`, used to describe the behavioral specifications. Such a module includes importing commands (for object specifications), behavioral operation (derivatives) declarations, and case sentences (not discussed here):

```
(cmod B-TSTREAM is
  importing TSTREAM .
  derivative left(*:BTree) right(*:BTree) node(*:BTree) .
  derivative hd(*:TStream) derivative tl(*:TStream) .
endcm)
```

To prove $blink(mirror(mirror(T)), T) = const(T)$, we introduce it as a new goal:

```
Maude>(add goal blink(mirror(mirror(T:BTree)),T:BTree)=const(T:BTree) .)
Goal blink(mirror(mirror(T:BTree)),T:BTree) =  const(T:BTree) added.
```

Then we can specify the tactic we want CIRC to use; for example:

```
Maude> (coinduction .)
Maude> rewrites: ... (ommited for space reasons)
Proof succeeded.
```

This command triggers the iterative execution of the three reduction rules. This property requires two coinduction proofs: one for streams and the other one for infinite trees. Note that the technique in [6] fails to automatically prove the above property because the user must explicitly tell the system where the second coinduction proof starts. Since the termination is not guaranteed, the "`coinduction`" tactic can also be given a depth (no depth needed here).

Several other examples have been experimented with in CIRC and can be found and run on CIRC's webpage at [10], including the following: proving behavioral equalities, i.e., language equivalence of regular expressions extended with complement; proving a series of known properties of infinite streams and operations on them; proving properties of powerlists; proving equivalent definitions of Fibonacci numbers equivalent; proving non-trivial properties about the Morse sequence; proving inductive and coinductive properties about finite or infinite trees. In all these examples, the important property to prove has been captured as an equation to be entailed by a specification extended with derivatives.

4 Conclusion

We presented CIRC, an automated prover supporting the principle of circularity and in particular circular coinduction. CIRC is implemented as an extension of Maude using its metalevel programming capabilities. An example was also discussed, reflecting the strength of CIRC in automatically proving behavioral properties. The technical details and proofs will be discussed elsewhere.

Acknowledgment. We warmly thank Andrei Popescu for help with the implementation of the first version of CIRC and for ideas on combining induction and coinduction via the principle of circularity, and to the anonymous referees for insightful comments.

References

1. Clavel, M., Durán, F.J., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: Specification and Programming in Rewriting Logic. Theoretical Computer Science 285, 187–243 (2002), Also extended Maude manual available at http://maude.csl.sri.com, http://maude.cs.uiuc.edu
2. Diaconescu, R., Futatsugi, K.: Behavioral coherence in object-oriented algebraic specification. JUCS 6(1), 74–96 (2000)
3. Goguen, J., Lin, K., Roşu, G.: Circular coinductive rewriting. In: Proceedings of Automated Software Engineering 2000, pp. 123–131. IEEE Computer Society Press, Los Alamitos (2000)
4. Goguen, J., Lin, K., Roşu, G.: Conditional circular coinductive rewriting with case analysis. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) Recent Trends in Algebraic Development Techniques. LNCS, vol. 2755, pp. 216–232. Springer, Heidelberg (2003)
5. Goguen, J., Malcolm, G.: A hidden agenda. J. of TCS 245(1), 55–101 (2000)

6. Hausmann, D., Mossakowski, T., Schröder, L.: Iterative circular coinduction for CoCASL in Isabelle/HOL. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 341–356. Springer, Heidelberg (2005)
7. Hennicker, R.: Context induction: a proof principle for behavioral abstractions. Formal Aspects of Computing 3(4), 326–345 (1991)
8. Hennicker, R., Bidoit, M.: Observational logic. In: Haeberer, A.M. (ed.) AMAST 1998. LNCS, vol. 1548, pp. 263–277. Springer, Heidelberg (1998)
9. Jacobs, B., Rutten, J.: A tutorial on (co)algebras and (co)induction. Bulletin of the European Association for Theoretical Computer Science 62, 222–259 (1997)
10. Lucanu, D., Roşu, G.: CIRC prover http://fsl.cs.uiuc.edu/index.php/Circ
11. Padawitz, P.: Swinging data types: Syntax, semantics, and theory. In: Haveraaen, M., Dahl, O.-J., Owe, O. (eds.) Recent Trends in Data Type Specification. LNCS, vol. 1130, pp. 409–435. Springer, Heidelberg (1996)
12. Roşu, G.: Hidden Logic. PhD thesis, University of California at San Diego (2000)
13. Roşu, G.: Equality of streams is a Π_2^0-complete problem. In: the 11th ACM SIGPLAN Int. Conf. on Functional Programming (ICFP'06), ACM Press, New York (2006)

Observing Distributed Computation. A Dynamic-Epistemic Approach

Radu Mardare

Microsoft Research - University of Trento
Centre for Computational and Systems Biology, Trento, Italy
mardare@cosbi.eu

Abstract. We propose a new logic designed for modelling and reasoning about information flow and information exchange between spatially located interconnected *agents* witnessing a distributed computation. The intention is to trace the process of knowledge acquisition and its dynamics in the context of distributed systems. Underpinning on the dual algebraical-coalgebraical characteristics of process calculi, we design a decidable and completely axiomatized logic that combines the process-algebraical/equational and the modal/coequational features and is developed for process-algebraical semantics.

1 Introduction

Observation is fast becoming an important topic in computer science. In which manner can observation (in the broad sense of the word) be used for computing? In which way can the partial information available to an external observer of a computational system be used in deriving knowledge about the overall complete system? We will approach these problems by developing a logic designed to handle (partial) information flow and information exchange between external observers (agents) of a distributed system.

In the context of (parallel) distributed computation, a concurrent computational system can be thought of as being composed of a number of *modules*, i.e. spatially localized and independently observable units of behavior and computation (e.g. programs or processors running in parallel), organized in networks of subsystems and being able to interact, collaborate, communicate and interrupt each other. In this context we shall consider *agents* - *external observers* of the modules. As an external observer, an agent witnesses the computation of its module and interacts with the whole system only by means of it. Thus it derives its knowledge about the overall system from the observed behavior of its subsystem and from epistemic reasoning on the knowledge (and reactions) of the other agents witnessing (different) parts of the same computational process.

In this context we are interested in specifying when agents can receive, communicate or protect truthful information, when they improve their knowledge, when they are aware of the knowledge of the others and how they can construct strategies for influencing the others knowledge. Hence, the problem is

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 379–393, 2007.
© Springer-Verlag Berlin Heidelberg 2007

related to issues of privacy, trust, secured communications, authentication, etc. covering many different areas, with potential applications: in Secure Communication (checking privacy and authentication for given communication protocols by studying the knowledge acquisition strategies that an intruder might take), in Debugging and Performance analysis (checking for the cause of errors or of high computational costs in systems where only some modules are accessible), in Artificial Intelligence (endowing artificial agents with good and flexible tools to reason about their changing environment and about each other), in designing and improving strategies for knowledge acquisition over complex networks (such as the Internet), etc.

In approaching this problem we have chosen the process-algebraical representation of (mobile) distributed systems and we developed a logic of information flow for process-algebraical semantics. Taking process calculi as semantics is theoretically challenging due to their dual algebraical/coalgebraical nature. While the algebraical features of processes are naturally approached in equational fashion (that reflects, on logical level, the program constructors), the coalgebraical features (intrinsically related to transition systems via the denotational and the operational semantics of process calculi) ask for a modal/coequational treatment. The modal approach is also needed for the epistemic reasoning.

Consequently, our paper combines two logical paradigms to information flow in distributed systems: *dynamic-epistemic (and doxastic) logics* [14,9,12], semantically based on epistemic-doxastic Kripke models; and the spatial logics for concurrency [6], for which the semantics is usually given in terms of process algebra.

Epistemic/doxastic logics [14,9] are multimodal logics that formalize the epistemic notions of *knowledge*, or *belief*, possessed by an agent, or a group of agents, using modalities indexed by agents. We have modalities like $K_A\phi$ (*"A knows that* ϕ*"*) or $\Box_A\phi$ (*"A justifiably believes that* ϕ*"*) for any agent A. The models associate to each basic modality a binary relation interpreted as *"indistinguishability"* relation $\xrightarrow{A}$ for each agent A. It expresses the agent's uncertainty about the current state of the system. The states s' such that $s \xrightarrow{A} s'$ are the epistemic alternatives of s to agent A, i.e. if the current state is s, A thinks that any of the alternatives s' may be the current state.

Dynamic logics [12] are closer to process calculi as they have names for *"programs"*, or *"actions"*, and ways to combine them. In this case we have modalities indexed on a signature $\mathbb{A}$ (the set of programs). A dynamic modality $[\pi]\phi$ captures the weakest precondition of such a program w.r.t. a given post-specification ϕ, and the accessibility relations are interpreted as transitions induced by programs. These logics already combine the coalgebraical features (modalities) with algebraical ones (the modalities have algebraic structures: programs are built using basic program constructors such as sequential composition or iteration).

Dynamic Epistemic Logics [1,2,11,3,8] are a class of logics that combine the dynamic and epistemic formalisms for specifying properties of evolving knowledge and beliefs in dynamic systems. The high level of expressivity reaches here a low complexity (decidability and complete axiomatizations).

Spatial Logics. Process semantics for modal logics can be considered as a special case of Kripke semantics, since, via transition systems, we can structure a class of processes as a Kripke model, by endowing it with accessibility relations induced by action transitions. Further we can use the standard clauses of Kripke semantics (e.g. Hennessy-Milner logic [13]). In addition, temporal, mobile and concurrent features have been added [22,7,20]. Spatial Logics [6] are the most expressive logics in this class containing equational operators to express spatial properties, such as the *parallel operator* $\phi|\psi$ and the *guarantee operator* $\phi \triangleright \psi$ (the adjoint of parallel), or operators for expressing the *"fresh name features"* inspired by the Gabbay-Pitts quantifier [10], etc.

The intention of this paper is to develop and study a logic that combines these two paradigms proposing a unified one. The new logic combines well with the process algebraical modelling of information flow and can directly express agent-dependent partial information features and their dynamics. We give a spatial interpretation of epistemic modalities in CCS. The intuition is to associate to each "agent" A the process P that describes the behavior of the module observed by A. The agent observing a process (possibly running in parallel with many other processes) *"knows"* only the activity and actions of its own process. *"Knowledge"* is thus defined as "information (about the overall, global process) that is locally available (to an agent)". In effect, this organizes any class $\mathcal{M}$ of processes (thought of as "states") as an epistemic Kripke model, with indistinguishability relations $\xrightarrow{A}$ for each agent A. Thus, if A observes the subprocess P then $P|P' \xrightarrow{A} P|P''$ for any P', P''. Since these are equivalence relations, we obtain a notion of "(truthful) knowledge". The resulting Kripke modality, $K_A\phi$, read *"agent A knows ϕ"*, holds at a given state (process) R iff the process P is active (as a subprocess) at R and property ϕ holds in any context in which P is active.

The resulting logic is completely axiomatizable and decidable. The Hilbert-style axiomatics we propose for it presents this logic as an authentic dynamic-epistemic logic. The classical axioms of knowledge will be present in our system together with spatial-like axioms.

1.1 Case Study: A Security Attack

For illustrating the problem we approach in this paper, we propose a toy example: a simplified *"Man-in-the-Middle"* type of cryptographic attack. Alice wants to communicate to Bob a secret over some communication channel. More concrete, she wants to inform Bob that a certain event p happened. Before receiving the message from Alice, Bob considers both alternatives, p and $\neg p$ equally possible. For communicating, Alice uses a key k to encrypt her messages while Bob (and only Bob) knows how to decrypt them ($\overline{k}$)[1]. But the communication channel is not secure: an evil outsider, Eve, has also access and her purpose is to make

[1] In a public-key cryptographic implementation, one could think of k as being Bob's public key, while $\overline{k}$ is Bob's private key (for decryption). In a different context, k might be Alice's password for communicating with Bob, which can only be authenticated by Bob using $\overline{k}$.

Bob believe $\neg p$. Suppose that Eve is also in possession of k (either because k was Bob's public key, or because Eve has somehow succeeded to steal Alice's password). Hence, she can present herself as Alice and is trying to convince Bob that $\neg p$. The communication of the secret event fails if Bob believes that the received message was from Alice and consequently $\neg p$ happened. In fact Eve manipulated Bob.

We are not concerned here primarily with the cryptographic details of the encryption method, but with the informational, "epistemic" features of this protocol. The main goal of it is to understand the epistemic status of the agents involved. What does Alice know? What does Bob know? What does Alice think that Bob knows? Does Alice know that her communication was unsuccessful? Does Eve know that her attack was successful? In which way the evolution of the system (of the processes involved) influences the information state of the agents? For proving the success or the failure of the protocol one has to show how Bob's knowledge can be influenced by Eve's actions and what can be done in order to avoid this.

One can use process algebras to describe such a scenario and logics for processes to specify properties of this protocol. But for answering to the previous questions, a way of arguing directly on the epistemic status of the agents is needed. We will prove further that despite of the apparent complexity of the epistemic reasoning on such frameworks there is a general approach that can formalize in a decidable manner the agents' reasoning in the above situation.

2 On Processes

In this section we introduce a fragment of CCS [19] calculus, that is "the core" of process calculi and will be used for defining the process-algebraical semantics for the logic. For the proofs of the results presented in this section and for additional results on this subject, the reader is referred to [16,18].

2.1 CCS Processes

Definition 1. *Let $\mathbb{A}$ be a denumerable signature. The syntax of the calculus is given by a grammar with one non-terminal symbol P and the productions $P := 0 \mid \alpha.P \mid P|P$, where $\alpha \in \mathbb{A}$. We denote by $\mathbb{P}$ the language generated by this grammar. We call the elements of $\mathbb{A}$* (basic) actions *and the objects in $\mathbb{P}$* processes.

Definition 2. *Let $\equiv \subseteq \mathfrak{P} \times \mathfrak{P}$ be the smallest equivalence relation on $\mathbb{P}$ s.t.*
- *$(\mathbb{P}, |, 0)$ is a commutative monoid with respect to $\equiv$;*
- *if $P' \equiv P''$ then $\alpha.P' \equiv \alpha.P''$ and $P'|P \equiv P''|P$, for any $\alpha \in \mathbb{A}$ and $P \in \mathbb{P}$.*

We call $\equiv$ structural congruence.

Definition 3. *We call a process P* guarded *if $P \equiv \alpha.Q$ for some $\alpha \in \mathbb{A}$. We denote $P^0 \stackrel{def}{=} 0$ and $P^k \stackrel{def}{=} \underbrace{P|...|P}_{k}$.*

Definition 4. *We consider, on $\mathfrak{P}$, the labelled transition system defined by the rules in Table 1. We denote $P \longrightarrow Q$ if $P \stackrel{\alpha}{\longrightarrow} Q$ or $P \stackrel{(\alpha,\overline{\alpha})}{\longrightarrow} Q$ for some $\alpha \in \mathbb{A}$.*

Table 1. The transition system

$$\frac{}{\alpha.P \stackrel{\alpha}{\longrightarrow} P} \qquad \frac{P \equiv Q \qquad P \stackrel{\alpha}{\longrightarrow} P'}{Q \stackrel{\alpha}{\longrightarrow} P'} \qquad \frac{P \stackrel{\alpha}{\longrightarrow} P'}{P|Q \stackrel{\alpha}{\longrightarrow} P'|Q} \qquad \frac{P \stackrel{\alpha}{\longrightarrow} P' \qquad Q \stackrel{\overline{\alpha}}{\longrightarrow} Q'}{P|Q \stackrel{(\alpha,\overline{\alpha})}{\longrightarrow} P'|Q}$$

We write $P \stackrel{Q:\alpha}{\longrightarrow} P'$ whenever $P \equiv Q|R$, $P' \equiv Q'|R$ and $Q \stackrel{\alpha}{\longrightarrow} Q'$. We write $P|Q \stackrel{P:\alpha,Q:\overline{\alpha}}{\longrightarrow} P'|Q'$ to denote the case when $P \stackrel{P:\alpha}{\longrightarrow} P'$ and $Q \stackrel{Q:\alpha}{\longrightarrow} Q'$. We call $(Q:\alpha)$ and $(P:\alpha, Q:\overline{\alpha})$ composed actions*.*

Definition 5. *We define for any process P, its set of actions $Act(P) \subset \mathbb{A}$:*
$1.Act(0) \stackrel{def}{=} \emptyset \quad 2.Act(\alpha.P) \stackrel{def}{=} \{\alpha\} \cup Act(P) \quad 3.Act(P|Q) \stackrel{def}{=} Act(P) \cup Act(Q)$
For $M \subset \mathbb{P}$ we define $Act(M) \stackrel{def}{=} \bigcup_{P \in M} Act(P)$.

Definition 6. *We call* action substitution *any mapping $\sigma : \mathbb{A} \longrightarrow \mathbb{A}$. We extend it, syntactically, to processes, $\sigma : \mathbb{P} \longrightarrow \mathbb{P}$, by*
1. $\sigma(0) \stackrel{def}{=} 0$ 2. $\sigma(P|Q) \stackrel{def}{=} \sigma(P)|\sigma(Q)$ 3. $\sigma(\alpha.P) \stackrel{def}{=} \sigma(\alpha).\sigma(P)$
Let $act(\sigma) \stackrel{def}{=} \{\alpha, \beta \in \mathbb{A} \mid \alpha \neq \beta,\ \sigma(\alpha) = \beta\}$ and for $M \subset \mathbb{P}$, $\sigma(M) \stackrel{def}{=} \{\sigma(P) \mid P \in M\}$.

We will also use M^σ, P^σ for denoting $\sigma(M)$ and $\sigma(P)$.

2.2 Size of a Process

Definition 7. *The* size *$[\![P]\!] = (h, w)$ of a process $P \in \mathbb{P}$ is given by:*
1. $[\![0]\!] \stackrel{def}{=} (0, 0)$
2. $[\![P]\!] \stackrel{def}{=} (h, w)$ *iff* $P \equiv (\alpha_1.Q_1)^{k_1}|...|(\alpha_j.Q_j)^{k_j}$ *with* $\alpha_i.Q_i \not\equiv \alpha_j.Q_j$ *for* $i \neq j$, *where* $h = 1 + max(h_1, .., h_k)$, $w = max(k_1, .., k_j, w_1, .., w_j)$ *for* $[\![Q_i]\!] = (h_i, w_i)$.
We write $(h_1, w_1) \leq (h_2, w_2)$ for $h_1 \leq h_2$ and $w_1 \leq w_2$ and $(h_1, w_1) < (h_2, w_2)$ for $h_1 < h_2$ and $w_1 < w_2$.

The intuition is that the size (h, w) of a process is given by the depth of its syntactic tree (*height h*) and by the maximum number of bisimilar processes that can be found in a node of the syntactic tree (*width w*). By construction, the size of a process is unique up to structural congruence.

Definition 8. *For a set $M \subset \mathbb{P}$ we define*[2] $[\![M]\!] \stackrel{def}{=} max\{[\![P]\!] \mid P \in M\}$.

[2] The size of a set of processes is not always well-defined. An infinite set, for example, might not have the maximum required. However we will use this definition only where it is well-defined.

2.3 Structural Bisimulation

We introduce the *structural bisimulation*, a relation on processes that is an approximation of the structural congruence defined on size. It analyzes the behavior of a process focusing on a boundary of its syntactic tree. This relation is similar with the pruning relation proposed in [4].

Definition 9. *Let $P, Q \in \mathbb{P}$. We define $P \approx_h^w Q$ inductivelly by:*
$P \approx_0^w Q$ always
$P \approx_{h+1}^w Q$ iff $\forall i \in 1..w$ and $\forall \alpha \in \mathbb{A}$ we have
- *if $P \equiv \alpha.P_1|...|\alpha.P_i|P'$ then $Q \equiv \alpha.Q_1|...|\alpha.Q_i|Q'$ with $P_j \approx_h^w Q_j$, $j = 1..i$*
- *if $Q \equiv \alpha.Q_1|...|\alpha.Q_i|Q'$ then $P \equiv \alpha.P_1|...|\alpha.P_i|P'$ with $Q_j \approx_h^w P_j$, $j = 1..i$*

We call $\approx_h^w$ structural bisimulation *on dimension (h, w).*

Proposition 1. *$\approx_h^w$ is a congruence relation on processes having the properties:*
1.(Antimonotonicity) if $P \approx_h^w Q$ and $(h', w') \leq (h, w)$ then $P \approx_{h'}^{w'} Q$.
2.(Inversion) if $P'|P'' \approx_h^{w_1+w_2} Q$ then $Q \equiv Q'|Q''$ and $P' \approx_h^{w_1} Q'$, $P'' \approx_h^{w_2} Q''$.

Proposition 2. *1. If $[\![P]\!] \leq (h, w)$ and $[\![P']\!] \leq (h, w)$ then $P \approx_h^w P'$ iff $P \equiv P'$.*
2. If $P \approx_h^w Q$ and $[\![P]\!] < (h, w)$ then $P \equiv Q$.

Hence, for a well-chosen size which depends on the processes involved, the structural bisimulation guarantees the structural congruence. Reverse, the structural congruence implies the structural bisimulation.

Proposition 3 (Behavioral simulation). *Let $P \approx_h^w Q$.*
1. If $P \xrightarrow{\alpha} P'$ then there exists a transition $Q \xrightarrow{\alpha} Q'$ s.t. $P' \approx_{h-1}^{w-1} Q'$.
2. If $[\![R]\!] < (h, w)$ and $P \xrightarrow{R:\alpha} P'$ then $Q \xrightarrow{R:\alpha} Q'$ and $P' \approx_{h-1}^{w-1} Q'$.

This states that the structural bisimulation is preserved by transitions with the price of decreasing the size.

2.4 Bound Pruning Processes

In this subsection we prove that for a given process P and a given size (h, w) we can always find a process Q, having the size at most equal with (h, w), such that $P \approx_h^w Q$. We will present a method for constructing Q from P, by pruning the syntactic tree of P to the given size.

Theorem 1 (Bound pruning theorem). *For any process $P \in \mathbb{P}$ and any size (h, w) there exists a process $Q \in \mathbb{P}$ with $P \approx_h^w Q$ and $[\![Q]\!] \leq (h, w)$.*

Proof. We construct [3] Q inductively on h.

Case $h = 0$: we take $Q \equiv 0$, as $P \approx_0^w Q$ and $[\![0]\!] = (0, 0)$.

Case $h + 1$: suppose $P \equiv \alpha_1.P_1|...|\alpha_n.P_n$.

Let P_i' be the result of pruning P_i by (h, w) (the inductive step of construction)

[3] This construction is not necessarily unique.

and $P' \equiv \alpha_1.P'_1|...|\alpha_n.P'_n$. As for any $i = 1..n$ we have $P_i \approx^w_h P'_i$ (by the inductive hypothesis), we obtain, using Proposition 1, that $\alpha_i.P_i \approx^w_{h+1} \alpha_i.P'_i$, hence $P \approx^w_{h+1} P'$. Consider now $P' \equiv (\beta_1.Q_1)^{k_1}|...|(\beta_m.Q_m)^{k_m}$. Let $l_i = min(k_i, w)$ for $i = 1..m$. Further we define $Q \equiv (\beta_1.Q_1)^{l_1}|...|(\beta_m.Q_m)^{l_m}$. Obviously $Q \approx^w_{h+1} P'$ and as $P \approx^w_{h+1} P'$, we obtain $P \approx^w_{h+1} Q$. By construction, $[\![Q]\!] \leq (h+1, w)$.

Definition 10. *For a process P and a tuple (h, w) we denote by $P_{(h,w)}$ the process obtained by pruning P to the size (h, w) by the method described in the proof of Theorem 1.*

3 Sets of processes

In this section we study the *closed sets of processes* that will play an essential role in proving the finite model property for the logic we will introduce. Intuitively, a closed set of processes is a set that whenever contains a process contains also any future "state" of that process and any "observable" subpart of it (what an observer might see from it). Syntactically this means that whenever we have a process in a closed set, we will also have all the processes that can be obtained by arbitrarily pruning the syntactic tree of our process. For the proofs of the results presented in this section the reader is referred to [18].

Definition 11. *For $M, N \subset \mathbb{P}$ and $\alpha \in \mathbb{A}$ we define:*
$\alpha.M \stackrel{def}{=} \{\alpha.P \mid P \in M\}$ $\quad\quad$ $M|N \stackrel{def}{=} \{P|Q \mid P \in M, Q \in N\}$.

Definition 12. *For $P \in \mathbb{P}$ we define $\pi(P) \subset \mathbb{P}$ inductively by:*
1. $\pi(0) \stackrel{def}{=} \{0\}$ $\quad$ 2. $\pi(\alpha.P) \stackrel{def}{=} \{0\} \cup \alpha.\pi(P)$ $\quad$ 3. $\pi(P|Q) \stackrel{def}{=} \pi(P)|\pi(Q)$
We extend the definition of π to sets of processes $M \subset \mathbb{P}$ by $\pi(M) \stackrel{def}{=} \bigcup_{P \in M} \pi(P)$.

Thus, we associate to each process P the set $\pi(P)$ of all processes obtained by arbitrarily pruning the syntactic tree of P.

Definition 13. *A set of processes $\mathcal{M} \subseteq \mathbb{P}$ is closed if it satisfies the conditions*
1. if $P \in \mathcal{M}$ and $P \longrightarrow P'$ then $P' \in \mathcal{M}$ $\quad$ 2. if $P \in \mathcal{M}$ then $\pi(P) \subset \mathcal{M}$.
We say that $\mathcal{M}$ is the closure of $M \subset \mathbb{P}$ if $\mathcal{M}$ is the smallest closed set of processes that contains M. We write $\overline{M} = \mathcal{M}$.
For any closed set $\mathcal{M}$ and any (h, w) we define $\mathcal{M}_{(h,w)} \stackrel{def}{=} \overline{\{P_{(h,w)} \mid P \in \mathcal{M}\}}$.
For $A \subset \mathbb{A}$ we define $\mathfrak{M}^A_{(h,w)} \stackrel{def}{=} \{\overline{M} \subset \mathbb{P} \mid Act(M) \subseteq A,\ [\![M]\!] \leq (h, w)\}$.

Lemma 1. *If $A \subset \mathbb{A}$ is a finite set of actions, then the following hold:*
1. If $\mathcal{M} \in \mathfrak{M}^A_{(h,w)}$ then $\mathcal{M}$ is a finite closed set of processes.
2. $\mathfrak{M}^A_{(h,w)}$ is finite.

The previous result shows that the set of closed sets of processes with actions from a given finite signature A and dimension not bigger than (h, w) is finite.

Definition 14. *Let $\mathcal{M}, \mathcal{N} \subset \mathbb{P}$ be closed sets. We write $\mathcal{M} \approx_h^w \mathcal{N}$ iff*

1. *for any $P \in \mathcal{M}$ there exists $Q \in \mathcal{N}$ with $P \approx_h^w Q$*
2. *for any $Q \in \mathcal{N}$ there exists $P \in \mathcal{M}$ with $P \approx_h^w Q$*

We write $(\mathcal{M}, P) \approx_h^w (\mathcal{N}, Q)$ when $P \in \mathcal{M}$, $Q \in \mathcal{N}$, $P \approx_h^w Q$ and $\mathcal{M} \approx_h^w \mathcal{N}$.

Further we state that having a closed set $\mathcal{M}$ with actions from A and a dimension (h, w) we can always find, in the finite set $\mathfrak{M}^A_{(h,w)}$, a closed set $\mathcal{N}$ structural bisimilar with $\mathcal{M}$ at the dimension (h, w).

Proposition 4. *For any closed set of processes $\mathcal{M}$, and any size (h, w) we have $\mathcal{M}_{(h,w)} \approx_w^h \mathcal{M}$.*

Theorem 2 (Bound pruning theorem). *Let $\mathcal{M}$ be a closed set of processes. Then for any (h, w) there is a closed set $\mathcal{N} \in \mathfrak{M}^{Act(\mathcal{M})}_{(h,w)}$ such that $\mathcal{M} \approx_h^w \mathcal{N}$.*

4 The Logic $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$

In this section we introduce the logic $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$ with multimodal operators indexed by "epistemic agents" from a signature $\mathfrak{A}$ and by "transition actions" from a signature $\mathbb{A}$. The proofs of the results presented further can be consulted in [18].

4.1 Syntax of $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$

Definition 15. *Consider a set $\mathcal{A}$ and its extension $\mathcal{A}^+$ generated, for arbitrary $\alpha \in \mathbb{A}$ and $e \in \mathcal{A}$, by $E := e \mid \alpha.E \mid E|E$. In addition, on $\mathcal{A}^+$ it is defined the smallest congruence relation $\equiv$ for which $|$ is commutative and associative. We call the $\equiv$-equivalence classes of $\mathcal{A}^+$* epistemic agents *and we call* atomic agents *the classes corresponding to elements of $\mathcal{A}$. For $E \in \mathcal{A}^+$ we denote by $\overline{E}$ its $\equiv$-equivalence class.*

A society of agents *is a set $\mathfrak{A}$ of epistemic agents satisfying the conditions*

1. *if $\overline{E_1|E_2} \in \mathfrak{A}$ then $\overline{E_1}, \overline{E_2} \in \mathfrak{A}$*
2. *if $\overline{\alpha.E} \in \mathfrak{A}$ then $\overline{E} \in \mathfrak{A}$*

Hereafter we denote by $A, A', A_1, ...$ arbitrary epistemic agents and we consider the canonical extension of the operators $|$ and $\alpha.$ from $\mathcal{A}^+$ to the epistemic agents.

Definition 16. *Let $\mathfrak{A}$ be a society of epistemic agents defined for the set $\mathbb{A}$ of actions. We define the language $\mathcal{F}_{\mathbb{A}}^{\mathfrak{A}}$ of $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$, for $A, A' \in \mathfrak{A}$ and $\alpha \in \mathbb{A}$ by:*

$\phi := \ 0 \mid \top \mid \neg\phi \mid \phi \wedge \phi \mid \phi|\phi \mid \langle a \rangle \phi \mid K_A \phi,$ *where*

$\langle a \rangle := \langle \alpha \rangle \mid \langle \alpha, \overline{\alpha} \rangle \mid \langle A : \alpha \rangle \mid \langle A, A' : \alpha \rangle.$

4.2 Process Semantics

A formula of $\mathcal{F}_{\mathbb{A}}^{\mathfrak{A}}$ will be evaluated to processes in a given closed set of processes, by using the satisfaction relation $\mathcal{M}, P \models \phi$.

Definition 17. *A model of* $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$ *is a pair* $(\mathcal{M}, I)$ *where* $\mathcal{M}$ *is a closed set of processes and* $I : (\mathfrak{A}, |, \alpha.) \longrightarrow (\mathcal{M}, |, \alpha.)$ *a homomorphism*[4] *of structures such that* $I(A) = 0$ *iff* $A = \overline{e}$ *for some* $e \in \mathcal{A}$.

We denote $P \stackrel{I(A):\alpha}{\longrightarrow} Q$ *by* $P \stackrel{A:\alpha}{\longrightarrow} Q$ *and* $P \stackrel{(I(A):\alpha, I(B):\overline{\alpha})}{\longrightarrow} Q$ *by* $P \stackrel{A,B:\alpha}{\longrightarrow} Q$. *Let* $\mathbb{A}^{\mathfrak{A}} = \mathbb{A} \cup \{(\alpha, \overline{\alpha}) \mid \alpha \in \mathbb{A}\} \cup \{(A : \alpha), (A, A' : \alpha) \mid \alpha \in \mathbb{A}, A, A' \in \mathfrak{A}\}$ *and* $a \in \mathbb{A}^{\mathfrak{A}}$ *an arbitrary element. For* $P \in \mathcal{M}$ *and* $\phi \in \mathcal{F}_{\mathbb{A}}^{\mathfrak{A}}$ *we define* $\mathcal{M}, P \models \phi$ *by:*
$\mathcal{M}, P \models \top$ *always.*
$\mathcal{M}, P \models 0$ *iff* $P \equiv 0$.
$\mathcal{M}, P \models \neg\phi$ *iff* $\mathcal{M}, P \not\models \phi$.
$\mathcal{M}, P \models \phi \wedge \psi$ *iff* $\mathcal{M}, P \models \phi$ *and* $\mathcal{M}, P \models \psi$.
$\mathcal{M}, P \models \phi|\psi$ *iff* $P \equiv Q|R$ *and* $\mathcal{M}, Q \models \phi$, $\mathcal{M}, R \models \psi$.
$\mathcal{M}, P \models \langle a \rangle \phi$ *iff there exists a transition* $P \stackrel{a}{\longrightarrow} P'$ *such that* $\mathcal{M}, P' \models \phi$.
$\mathcal{M}, P \models K_A \phi$ *iff* $P \equiv I(A)|R$ *and for all* $I(A)|R' \in \mathcal{M}$ *we have* $\mathcal{M}, I(A)|R' \models \phi$.

Definition 18 (Derived operators). *In addition to the classical boolean operators, we introduce other derived operators:*

$$1 \stackrel{def}{=} \neg((\neg 0) \mid (\neg 0)) \qquad \alpha.\psi \stackrel{def}{=} (\langle \alpha \rangle \psi) \wedge 1$$
$$[a]\phi \stackrel{def}{=} \neg(\langle a \rangle (\neg \phi)) \qquad \widetilde{K}_A \phi \stackrel{def}{=} \neg K_A \neg \phi.$$

We use the precedence order $\neg$, K_A, $\langle a \rangle$, $|, \wedge, \vee, \rightarrow$ *for the operators, where* $\neg$ *has precedence over all.*

The semantics of the derived operators will be:
$\mathcal{M}, P \models [a]\phi$ iff for any transition $P \stackrel{a}{\longrightarrow} P'$ (if any) we have $\mathcal{M}, P' \models \phi$
$\mathcal{M}, P \models 1$ iff $P \equiv 0$ or $P \equiv \alpha.Q$
$\mathcal{M}, P \models \alpha.\phi$ iff $P \equiv \alpha.Q$ and $\mathcal{M}, Q \models \phi$
$\mathcal{M}, P \models \widetilde{K}_A \phi$ iff either $P \not\equiv I(A)|R$ for any R, or $\exists I(A)|S \in \mathcal{M}$ such that $\mathcal{M}, I(A)|S \models \phi$

Remark the interesting semantics of the operators K_A and $\widetilde{K}_A$ for $A \in I^{-1}(0)$:
$\mathcal{M}, P \models K_A \phi$ iff $\forall Q \in \mathcal{M}$ we have $\mathcal{M}, Q \models \phi$
$\mathcal{M}, P \models \widetilde{K}_A \phi$ iff $\exists Q \in \mathcal{M}$ such that $\mathcal{M}, Q \models \phi$

Hence $K_A \phi$ and $\widetilde{K}_A \phi$ for an atomic agent A encode, in syntax, the validity and the satisfiability with respect to a given model.

4.3 Bounded Finite Model Property

Definition 19. *The* sizes of a formula *(height and width)* $(\!|\phi|\!) = (h, w)$ *w.r.t. the homomorphism* I *is given inductively in Table 2.*

Lemma 2. *If* $(\!|\phi|\!) = (h, w)$, $\mathcal{M}, P \models \phi$ *and* $(\mathcal{M}, P) \approx_h^w (\mathcal{N}, Q)$ *then* $\mathcal{N}, Q \models \phi$.

Hence ϕ is *"sensitive"* via satisfaction only up to size $(\!|\phi|\!)$. In other words, the relation $\mathcal{M}, P \models \phi$ is conserved by substituting the pair (M, P) with any other

[4] The function I associates to each agent the process it observes. An atomic agent sees always the process 0.

Table 2. Sizes of formulas

Suppose $(\!|\phi|\!) = (h, w)$, $(\!|\psi|\!) = (h', w')$, $[\![I(A)]\!] = (h_A, w_A)$ and $[\![I(A,)]\!] = (h_{A'}, w_{A'})$, then
1.$(\!|0|\!) = (\!|\top|\!) \stackrel{def}{=} (0, 0)$ 2.$(\!|\neg\phi|\!) \stackrel{def}{=} (\!|\phi|\!)$ 3.$(\!|\phi \wedge \psi|\!) \stackrel{def}{=} (max(h, h'), max(w, w'))$
4.$(\!|\phi|\psi|\!) \stackrel{def}{=} (max(h, h'), w + w')$ 5.$(\!|\langle\alpha\rangle\phi|\!) = (\!|\langle\alpha, \overline{\alpha}\rangle\phi|\!) \stackrel{def}{=} (1 + h, 1 + w)$
6.$(\!|\langle A : \alpha\rangle\phi|\!) \stackrel{def}{=} (1 + max(h, h_A), 1 + max(w, w_A))$
7.$(\!|K_A\phi|\!) \stackrel{def}{=} (1 + max(h, h_A), 1 + max(w, w_A))$
8.$(\!|\langle A, A' : \alpha\rangle\phi|\!) \stackrel{def}{=} (1 + max(h, h_A, h_{A'}), 1 + max(w, w_A, w_{A'}))$

pair (N, P) structurally bisimilar to it at the size $(\!|\phi|\!)$. Using this result, we conclude that if a process satisfies ϕ w.r.t. a given closed set of processes, then by pruning the process and the closed set on the size $(\!|\phi|\!)$, we preserve the satisfiability for ϕ. Indeed the theorems 1 and 4 prove that if $(\!|\phi|\!) = (h, w)$ then $(\mathcal{M}, P) \approx_w^h (\mathcal{M}_{(\!|\phi|\!)}, P_{(\!|\phi|\!)})$. Hence $\mathcal{M}, P \models \phi$ implies $\mathcal{M}_{(\!|\phi|\!)}, P_{(\!|\phi|\!)} \models \phi$.

Definition 20. *The set of actions of a formula ϕ is defined in Table 3.*

Table 3. The set of actions of a formula

1. $act(0) = act(\top) \stackrel{def}{=} \emptyset$
2. $act(\langle\alpha\rangle\phi) \stackrel{def}{=} \{\alpha\} \cup act(\phi)$
3. $act(\neg\phi) = act(\phi)$
4. $act(\langle\alpha, \overline{\alpha}\rangle\phi) \stackrel{def}{=} \{\alpha, \overline{\alpha}\} \cup act(\phi)$
5. $act(\phi \wedge \psi) = act(\phi|\psi) \stackrel{def}{=} act(\phi) \cup act(\psi)$
6. $act(\langle A : \alpha\rangle\phi) = act(K_A\phi) \stackrel{def}{=} Act(I(A)) \cup act(\phi)$
7. $act(\langle A, A' : \alpha\rangle\phi) \stackrel{def}{=} Act(I(A)) \cup Act(I(A')) \cup act(\phi)$

The next result states that a formula ϕ does not reflect properties that involve more then the actions in its syntax. Thus if $\mathcal{M}, P \models \phi$ then any substitution σ having the elements of $act(\phi)$ as fix points preserves the satisfaction relation.

Lemma 3. *If $\mathcal{M}, P \models \phi$ and σ a substitution with $act(\sigma) \bigcap act(\phi) = \emptyset$ then $\mathcal{M}^\sigma, P^\sigma \models \phi$.*

Consider a lexicographical order $\ll$ on $\mathbb{A}$. For a finite set $B \subset \mathbb{A}$ there exists a unique maximal element. We denote by B_+ the set obtained by adding to B the successor, w.r.t. $\ll$, of its maximal element.

Lemma 4 (Finite model property). *If $\mathcal{M}, P \models \phi$ then $\exists \mathcal{N} \in \mathfrak{M}_{(\!|\phi|\!)}^{act(\phi)_+}$ and $Q \in \mathcal{N}$ such that $\mathcal{N}, Q \models \phi$.*

Because $act(\phi)$ is finite, Theorem 1 states that $\mathfrak{M}_{(\!|\phi|\!)}^{act(\phi)_+}$ is finite and any closed set $\mathcal{M} \in \mathfrak{M}_{(\!|\phi|\!)}^{act(\phi)_+}$ is finite as well. Thus we obtain the finite model property for our logic. A consequence of theorem 4 is the decidability for satisfiability, validity and model checking against the process semantics.

Theorem 3 (Decidability). *For $\mathcal{L}^{\mathfrak{A}}_{\mathbb{A}}$ validity, satisfiability and model checking are decidable against process semantics.*

4.4 Characteristic Formulas

In this subsection we use the peculiarities of the dynamic and epistemic operators to define characteristic formulas for processes and for finite closed sets of processes.

Definition 21. *Consider the class of logical formulas indexed by ($\equiv$-equivalence classes of) processes $\mathcal{F}_{\mathbb{P}} = \{(f_P) \mid P \in \mathbb{P}\}$ defined as follows*[5]*:*

1. $f_0 \stackrel{def}{=} 0$ 2. $f_{P|Q} \stackrel{def}{=} f_P|f_Q$ 3. $f_{\alpha.P} \stackrel{def}{=} \alpha.f_P$

Proposition 5. *f_P is a characteristic formula for P, i.e. $\mathcal{M}, P \models f_Q$ iff $P \equiv Q$.*

Definition 22. *Consider the class of logical formulas indexed by epistemic agents $\mathcal{F}_{\mathfrak{A}}$ defined as follows*[6]*: Similarly we introduce a class of logical formulas $(f_A)_{A\in\mathfrak{A}}$, on epistemic agents*

1. $f_A \stackrel{def}{=} 0$ *if A is atomic agent*   2. $f_{A_1|A_2} \stackrel{def}{=} f_{A_1}|f_{A_2}$ 3. $f_{\alpha.A} \stackrel{def}{=} \alpha.f_A$

Proposition 6. *$\mathcal{M}, P \models f_A$ iff $P \equiv I(A)$.*

Definition 23. *Let $\Phi \subset \mathcal{F}^{\mathfrak{A}}$ be a finite set of formulas and $A \in \mathfrak{A}$ an atomic agent. Let $\Delta\Phi \stackrel{def}{=} K_A(\bigvee_{\phi\in\Phi}\phi) \wedge (\bigwedge_{\phi\in\Phi}\widetilde{K}_A\phi)$.*

Observe that $\mathcal{M}, P \models \Delta\Phi$ iff for any $Q \in \mathcal{M}$ there exists $\phi \in \Phi$ such that $\mathcal{M}, Q \models \phi$ and for any $\phi \in \Phi$ there exists $Q \in \mathcal{M}$ such that $\mathcal{M}, Q \models \phi$. Observe also that it is irrelevant which atomic agent A we choose to define Δ, as the epistemic operators of any atomic agent can encode validity and satisfiability.

Further we exploit the semantics of this operator for defining characteristic formulas for finite closed sets of processes.

Definition 24. *For a finite closed set of processes $\mathcal{M}$ let $f_{\mathcal{M}} = \Delta\{f_P \mid P \in \mathcal{M}\}$.*

Proposition 7. *If $\mathcal{M}, \mathcal{N}$ are finite closed sets of processes and $P \in \mathcal{M}$ then $\mathcal{M}, P \models f_{\mathcal{N}}$ iff $\mathcal{N} = \mathcal{M}$.*

4.5 Axiomatic System

Consider the subset of logical formulas given by $f := \alpha.0 \mid \alpha.f \mid f|f$ for $\alpha \in \mathbb{A}$. We denote the class of these formulas by $\mathcal{F}$[7]. Hereafter we use f, g, h for denoting arbitrary formulas from $\mathcal{F}$, while ϕ, ψ, ρ will be used for formulas in $\mathcal{F}^{\mathfrak{A}}_{\mathbb{A}}$.

Proposition 8. *$\mathcal{F} \cup \{0\} = \mathcal{F}_{\mathbb{P}}$.*

In table 4 is proposed a Hilbert-style axiomatic system for $\mathcal{L}^{\mathfrak{A}}_{\mathbb{A}}$. We assume the axioms and the rules of propositional logic. In addition we have a set of spatial axioms

Table 4. The axiomatic system $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$

Spatial axioms	
S1: $\vdash \top \vert \bot \to \bot$	S4: $\vdash \phi \vert (\psi \vee \rho) \to (\phi \vert \psi) \vee (\phi \vert \rho)$
S2: $\vdash (\phi \vert \psi) \vert \rho \to \phi \vert (\psi \vert \rho)$	S5: $\vdash \phi \vert \psi \to \psi \vert \phi$
S3: $\vdash \phi \vert 0 \leftrightarrow \phi$	S6: $\vdash (f \wedge \phi \vert \psi) \to \bigvee_{f \leftrightarrow g \vert h} (g \wedge \phi) \vert (h \wedge \psi)$
Spatial rules	
SR1: If $\vdash \phi \to \psi$ then $\vdash \phi \vert \rho \to \psi \vert \rho$	
Dynamic axioms	
D1: $\vdash \langle a \rangle \phi \vert \psi \to \langle a \rangle (\phi \vert \psi)$	D3: $\vdash 0 \vee \langle ! \alpha \rangle \top \to [\beta] \bot$, for $\alpha \neq \beta$
D2: $\vdash [a](\phi \to \psi) \to ([a]\phi \to [a]\psi)$	D4: $\vdash \langle ! \alpha \rangle \phi \to [\alpha] \phi$
Dynamic rules	
DR1: If $\vdash \phi$ then $\vdash [a]\phi$	DR2: If $\vdash \phi_1 \to [a]\phi_1'$ and $\vdash \phi_2 \to [a]\phi_2'$ *then* $\vdash \phi_1 \vert \phi_2 \to [a](\phi_1' \vert \phi_2 \vee \phi_1 \vert \phi_2')$
Epistemic axioms	
E1: $\vdash K_A \phi \wedge K_A(\phi \to \psi) \to K_A \psi$	
E2: $\vdash K_A \phi \to \phi$	E4: $\vdash K_A \top \to (\neg K_A \phi \to K_A \neg K_A \phi)$
E3: $\vdash K_A \phi \to K_A K_A \phi$	E5: $\vdash K_A \top \leftrightarrow f_A \vert \top$
Axioms involving atomic agents $A_0 \in \mathcal{A}$	
E6: $\vdash K_A \phi \leftrightarrow (K_A \top \wedge K_{A_0}(K_A \top \to \phi))$	E8: $\vdash K_{A_0} \phi \to [a] K_{A_0} \phi$
E7: $\vdash K_{A_0} \phi \wedge \psi \vert \rho \to (K_{A_0} \phi \wedge \psi) \vert (K_{A_0} \phi \wedge \rho)$	E9: $\vdash K_{A_0} \phi \to (K_A \top \to K_A K_{A_0} \phi)$
Epistemic rules	
ER1: If $\vdash \phi$ then $\vdash K_A \top \to K_A \phi$	
Mixed axioms	
M1: $\vdash \langle A : \alpha \rangle \top \to K_A \top$	M3: $\vdash \langle A : \alpha \rangle \phi \wedge \langle A \vert A' : \alpha \rangle \top \to \langle A \vert A' : \alpha \rangle \phi$
M2: $\vdash f_A \to (\langle \alpha \rangle \phi \leftrightarrow \langle A : \alpha \rangle \phi)$	M4: $\vdash \langle A : \alpha \rangle \phi \vert \langle A' : \overline{\alpha} \rangle \psi \to \langle A, A' : \alpha \rangle (\phi \vert \psi)$
Mixed rules	
MR1: If $\vdash (\bigvee_{\mathcal{M} \in \mathfrak{M}^{act(\phi)+}_{(\!\vert \phi \vert\!)}} f_{\mathcal{M}}) \to \phi$ then $\vdash \phi$	

and rules, of dynamic axioms and rules and of epistemic axioms and rules. We will also have a class of mixed axioms and rules that combine different operators.

Observe that the disjunctions in axiom $S6$ and in the rule $MR1$ are finitary.

Definition 25. *We say that a formula* $\phi \in \mathcal{F}^{\mathfrak{A}}$ *is* provable in $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$ *if* ϕ *can be derived, as a theorem, using the axioms and the rules of* $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$*. We denote this by* $\vdash \phi$*. We say that a formula* $\phi \in \mathcal{F}_{\mathbb{A}}^{\mathfrak{A}}$ *is* consistent in $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$ *if* $\neg\phi$ *is not* $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$*-provable.*

We call a formula $\phi \in \mathcal{F}_{\mathbb{A}}^{\mathfrak{A}}$ satisfiable *if there exists a context* $\mathcal{M}$ *and a process* $P \in \mathcal{M}$ *such that* $\mathcal{M}, P \models \phi$*. We call a formula* $\phi \in \mathcal{F}_{\mathbb{A}}^{\mathfrak{A}}$ validity *if for any context* $\mathcal{M}$ *and any process* $P \in \mathcal{M}$ *we have* $\mathcal{M}, P \models \phi$*. In such a situation we write* $\models \phi$*.*

Theorem 4 (Soundness). *The system* $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$ *is sound w.r.t. process semantics, i.e. if* $\vdash \phi$ *then* $\models \phi$*.*

Theorem 5 (Completeness). *The axiomatic system of* $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$ *is complete w.r.t. process semantics, i.e. if* $\models \phi$ *then* $\vdash \phi$*.*

[5] Note that $\mathcal{F}_{\mathbb{P}} \subset \mathcal{F}_{\mathbb{A}}^{\mathfrak{A}}$.

[6] Note that $\mathcal{F}_{\mathfrak{A}} \subset \mathcal{F}_{\mathbb{A}}^{\mathfrak{A}}$.

[7] By construction, $\mathcal{F} \subset \mathcal{F}_{\mathbb{A}}^{\mathfrak{A}}$.

The proof of this theorem uses the characteristic formulas for processes and finite closed sets and consists in proving the equivalence equivalence between $\mathcal{M}, P \models \phi$ and $\vdash f_{\mathcal{M}} \wedge f_P \rightarrow \phi$.

5 Formalizing the Security Scenario

We return to the security scenario proposed in subsection 1.1 and we will use CCS to encode the process and the logic to analyze it. The entire process can be represented as the process $P \equiv k.\alpha.A \mid \overline{k}.(\overline{\alpha}.P'|\overline{\beta}.P'') \mid k.\beta.E$, where P' is interpreted as *"event p happened"* while P'' as *"event ¬p happened"*. $k.\alpha.A$ is the process of Alice, $\overline{k}(\overline{\alpha}.P'|\overline{\beta}.P'')$ is Bob's and $k.\beta.E$ is the process of Eve. We associate to Alice three epistemic agents, A_1, A_2, A_3 that represent the three successive states of Alice in our scenario. Similarly E_1, E_2, E_3 are the agents representing different instances of Eve, while B_1, B_2, B_3, B_4 represent instances of Bob. The model is given by $\mathcal{M} = \overline{\{P\}}$ and the interpretation I in Table 5.

Table 5. The interpretation of epistemic agents

Alice	*Bob*	*Eve*
$I(A_1) = k.\alpha.A$	$I(B_1) = \overline{k}.(\overline{\alpha}.P'|\overline{\beta}.P'').\beta.E$	$I(E_1) = k.\beta.E$
$I(A_2) = \alpha.A$	$I(B_2) = \overline{\alpha}.P'|\overline{\beta}.P''$	$I(E_2) = \beta.E$
$I(A_3) = A$	$I(B_3) = P'|\overline{\beta}.P''$	$I(E_3) = E$
	$I(B_4) = \overline{\alpha}.P'|P''$	

Now we can express that Alice and Bob can recognize each other and that Alice can inform Bob about p by $\mathcal{M}, P \models \langle A_1, B_1 : k\rangle\langle A_2, B_2 : \alpha\rangle(P'|\top)$.

But Bob can also communicate with Eve, as Eve has the encryption key: $\mathcal{M}, P \models \langle A_1, E_1 : k\rangle\langle A_2, E_2 : \alpha\rangle(P''|\top)$.

Alice knows that she can communicate with Bob, using the key k, and as a result Bob will be informed about the event p: $\mathcal{M}, P \models K_{A_1}\langle k\rangle\langle\alpha\rangle(P'|\top)$. But if Alice is aware of the possibility of an attack she cannot be sure that, after she sent the messages to Bob, Bob does know that p happened; it might be the case that Bob did not receive Alice's message and that he communicated instead with the impersonator:
$\mathcal{M}, P \models \neg[k][\alpha]K_{A_3}(P'|\top)$ or $\mathcal{M}, P \models \langle k\rangle\langle\alpha\rangle\neg K_{A_3}K_{B_3}(P'|\top)$.

Alice knows that Bob knows that p happened only if Bob did the two communications with her: $\mathcal{M}, P \models K_{A_1}\langle A_1, B_1 : k\rangle\langle A_2, B_2 : \alpha\rangle K_{B_3}(P'|\top)$.

Before communication Bob knows only that whatever Alice will say it will be true: $\mathcal{M}, P \models K_{B_1}[A_1, B_1 : k][A_2, B_2 : \alpha]\top$.

Eve knows that she can present herself as Alice (i.e. can send k) but she can be sure that will communicate with Bob:
$\mathcal{M}, P \models K_{E_1}\langle E_1 : k\rangle\top$ and $\mathcal{M}, P \models \neg K_{E_1}\langle E_1, B_1\rangle\top$

In the same way we can express many complex properties. Further we can use model checking or theorem proving to play with such properties.

6 Concluding Remarks and Future Works

In this paper we introduced a new dynamic-epistemic logic, $\mathcal{L}_{\mathbb{A}}^{\mathfrak{A}}$, with a process-algebraical semantics that combines well with process algebraical modelling of information flow, but that can also directly express agent-dependent partial information features and their dynamics. This logic is meant to be used for expressing properties of multiagent distributed systems. In this respect the society of agents $\mathfrak{A}$ came with an algebraical structure that depicts the distribution of the modules which are observed by the agents. In expressing this we used operators from spatial logics together with operators characteristic for dynamic-epistemic logics.

The logic is presented with a complete and decidable axiomatic system containing similar axioms with the logics it combines.

With respect to dynamic-epistemic logics, the novelty of our logic consists in assuming an algebraical structure on the class of agents. Thus, we can speak about the knowledge of agents A', A'' but also about the knowledge of the agent $A'|A''$ which subsumes the knowledge of A', of A'', and the knowledge derived from the fact that what A' and A'' see are modules running in parallel as parts of the same system.

With respect to logics for processes (spatial logics), our logic focuses on agents and their knowledge proposing a direct way of encoding epistemic properties that are relevant for many applications and which, using the logics of processes only can be encoded in a difficult or unnatural way. Thus we can trace the evolution of the agent's knowledge and we can express properties concerning their dynamics. Such properties are important e.g. in analyzing communication protocols where the success of a protocol depends on the knowledge of the agents involved.

References

1. Baltag, A., Moss, L.S.: Logics for Epistemic Programs. Synthese 139(2) (2004)
2. Baltag, A., Moss, L.S., Solecki, S.: The Logic of Public Announcements. Common Knowledge and Private Suspicions, CWI Tech. Rep. SEN-R9922 (1999)
3. van Benthem, J.F.A.K.: Logic for information update. In: Proc. of TARK'01 (2001)
4. Calcagno, C., Cardelli, L., Gordon, A.D.: Deciding validity in a spatial logic for trees. Journal of Functional Programming 15 (2005)
5. Caires, L., Lozes, E.: Elimination of Quantifiers and Decidability in Spatial Logics for Concurrency. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, Springer, Heidelberg (2004)
6. Caires, L., Cardelli, L.: A Spatial Logic for Concurrency (Part I). Information and Computation 186(2) (2003)
7. Dam, M.: Model checking mobile processes. Information and Computation 129(1) (1996)
8. van Ditmarsch, H.: Knowledge games. Bull. of Economic Research 53(4) (2001)
9. Fagin, R., et al.: Reasoning about Knowledge. MIT Press, Cambridge (1995)
10. Gabbay, M., Pitts, A.: A New Approach to Abstract Syntax Involving Binders. Formal Aspects of Computing 13(3-5), 341–363 (2002)
11. Gerbrandy, J., Groeneveld, W.: Reasoning about information change. Journal of Logic, Language and Information 6 (1997)

12. Harel, D., et al.: Dynamic Logic. MIT Press, Cambridge (2000)
13. Hennessy, M., Milner, R.: Algebraic laws for Nondeterminism and Concurrency. Journal of JACM 32(1) (1985)
14. Hintikka, J.: Knowledge and Belief, Ithaca, N.Y.: Cornell University Press (1962)
15. Mardare, R.: Logical analysis of Complex Systems. Dynamic Epistemic Spatial Logics, Ph.D. thesis, DIT, University of Trento (2006)
16. Mardare, R., Priami, C.: Decidable extensions of Hennessy-Milner Logic. In: Najm, E., Pradat-Peyre, J.F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, Springer, Heidelberg (2006)
17. Mardare, R., Priami, C.: Dynamic Epistemic Spatial Logics, Technical Report, 03/, Microsoft Research Center for Computational and Systems Biology, Trento, Italy (2006) available from http://www.cosbi.eu
18. Mardare, R.: Dynamic-Epistemic reasoning on distributed systems, Technical Report 2007, Microsoft Research Center for Computational and Systems Biology, Trento, Italy (2006) available from http://www.cosbi.eu
19. Milner, R.: A Calculus of Communicating Systems. Springer-Verlag, New York, Inc. (1982)
20. Milner, R., Parrow, J., Walker, D.: Modal logics for mobile processes. TCS 114 (1993)
21. Sangiorgi, D.: Extensionality and Intensionality of the Ambient Logics. In: Proc. of the 28th ACM Annual Symposium on Principles of Programming Languages (2001)
22. Stirling, C.: Modal and temporal properties of processes. Springer-Verlag, New York, Inc. (2001)

Nabla Algebras and Chu Spaces

Alessandra Palmigiano and Yde Venema*

Universiteit van Amsterdam, ILLC
Plantage Muidergracht 24
1018 TV Amsterdam, Netherlands

Abstract. This paper is a study into some properties and applications of Moss' coalgebraic or 'cover' modality ∇.

First we present two axiomatizations of this operator, and we prove these axiomatizations to be sound and complete with respect to basic modal and positive modal logic, respectively. More precisely, we introduce the notions of a modal ∇-algebra and of a positive modal ∇-algebra. We establish a categorical isomorphism between the category of modal ∇-algebra and that of modal algebras, and similarly for positive modal ∇-algebras and positive modal algebras.

We then turn to a presentation, in terms of relation lifting, of the Vietoris hyperspace in topology. The key ingredient is an F-lifting construction, for an arbitrary set functor F, on the category Chu of two-valued Chu spaces and Chu transforms, based on relation lifting.

As a case study, we show how to realize the Vietoris construction on Stone spaces as a special instance of this Chu construction for the (finite) power set functor. Finally, we establish a tight connection with the axiomatization of the modal ∇-algebras.

Keywords: coalgebra, relation lifting, modal algebra, Vietoris hyperspace, Chu space.

1 Introduction

This paper is a study into the algebraic properties of the coalgebraic modal operator ∇, and some of its applications. The connective ∇ takes a finite[1] *set* Φ of formulas and returns a single formula $\nabla\Phi$. Using the standard modal language, ∇ can be seen as a defined operator:

$$\nabla\Phi = \Box(\bigvee \Phi) \wedge \bigwedge \Diamond\Phi, \tag{1}$$

where $\Diamond\Phi$ denotes the set $\{\Diamond\varphi \mid \varphi \in \Phi\}$.

Readers familiar with classical first-order logic will recognize the quantification pattern in (1) from the theory of Ehrenfeucht-Fraïssé games, Scott sentences, and the like, see [9] for an overview. In modal logic, related ideas made

* The research of both authors has been made possible by VICI grant 639.073.501 of the Netherlands Organization for Scientific Research(NWO).

[1] In this paper we restrict to the finitary version of the operator.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 394–408, 2007.
© Springer-Verlag Berlin Heidelberg 2007

an early appearance in Fine's work on normal forms [8]. As far as we know, however, the first explicit occurrences of the nabla *connective* appeared roughly at the same time, in the work of Barwise & Moss on circularity [4], and that of Janin & Walukiewicz on automata-theoretic approaches towards the modal μ-calculus [10].

The semantics of the nabla modality can be explicitly formulated as follows, for an arbitrary Kripke structure $\mathbb{S}$ with accessibility relation R:

$$\mathbb{S}, s \Vdash \nabla\Phi \text{ if } \begin{array}{l} \text{for all } \varphi \in \Phi \text{ there is a } t \in R[s] \text{ with } \mathbb{S}, t \Vdash \varphi, \text{ and} \\ \text{for all } t \in R[s] \text{ there is a } \varphi \in \Phi \text{ with } \mathbb{S}, t \Vdash \varphi. \end{array} \tag{2}$$

In other words, the semantics of ∇ can be expressed in terms of the *relation lifting* of the satisfaction relation between states and formulas:

$$\mathbb{S}, s \Vdash \nabla\Phi \text{ iff } (R[s], \Phi) \in \overline{\mathsf{P}}(\Vdash). \tag{3}$$

This insight, which is nothing less than a coalgebraic reformulation of modal logic, led Moss [14] to the introduction of *coalgebraic logic*, in which (3) is generalized to an (almost) arbitrary set functor F by introducing a coalgebraic operator ∇_{F}, and interpreting it using the relation lifting $\overline{\mathsf{F}}(\Vdash)$ of the forcing relation.

In this paper we want to look at ∇ as an algebraic operator in its own right. Our motivation for undertaking such a study, besides a natural intellectual curiosity, was twofold: firstly, we hope that such a study might be a first step in the direction of a 'coalgebraic proof theory' (we'll come back to this towards the end of this paper). And second, we belief that a thorough algebraic understanding of the nabla operator might shed light on power lifting constructions, such as the Vietoris hyperspace construction in topology. Let us address these issues in some more detail, and on the way explain what we believe to be the contribution of this paper.

Concerning the algebraic properties, the main issue that we address concerns *axiomatizations*. We were interested in axiomatizing the properties of the nabla operator in terms that only refer to ∇ itself and its interaction with the Boolean connectives, but which does not involve the non-coalgebraic modalities $\Box$ and $\Diamond$. As we will see in the next section, such an 'intrinsic' axiomatization is indeed possible. A remarkable feature of our axiomatization is that it is largely independent of the Boolean negation, so that its natural algebraic setting is that of positive modal algebras [7]. On the other hand, the nabla operator for the power set functor interacts reasonably well with the complementation operator, so that in fact we obtain two sound and complete axiomatizations for ∇, one in the setting of positive modal logic, and one in the setting of classical (i.e., Boolean) modal logic. Both of these results are formulated in terms of an isomorphism between categories of algebras.

The connection between the nabla operator and powering constructions in topology [13] is less obvious — we confine our attention to the Vietoris hyperspace. Formulations of the Vietoris hyperspace construction involving modal logic are well-known [11,18], and the importance of the Vietoris construction on the interface of coalgebra and modal logic has already been the object of a

number of studies [1,12,15,6]. Indeed, one may argue that the coalgebras of the Vietoris endofunctor on Stone spaces provide an adequate semantics for all modal logics since there is an isomorphism between the category of these coalgebras, and the category of the descriptive general frames known in modal logic [12]. Here, however, we take a slightly different angle. Our goal was to somehow define the Vietoris hyperspace construction in a way that would be relevant and useful for coalgebraic applications and that would only refer to category-theoretic properties of the power set functor. Analogous to Moss' coalgebraic approach to modal logic, this might enable one to generalize the Vietoris construction to arbitrary set functors. The key idea in our approach is to formulate the Vietoris construction in terms of the relation lifting $\overline{\in} := \overline{\mathsf{P}}(\in)$ of the *membership* relation between points and (open/closed/clopen) sets.

As it turned out, *Chu spaces* provide a natural setting for this. A Chu space is a triple $S = \langle X, S, A \rangle$ consisting of two sets X and A, together with a binary relation[2] $S \subseteq X \times A$. In itself, the connection with Chu spaces should not come as a big surprise: as we will show in more detail further on, we may read (3) as saying that the semantics of ∇ itself can be seen as a Chu transform, that is, an arrow in the category Chu. In section 3, we give F-lifting constructions on Chu spaces for arbitrary endofunctors F on Set. The main desiderata of these constructions are functoriality and preservation of the full subcategory of *normal* Chu spaces (see Definition 8 below). Since the latter is not met in general, we also introduce a normalization functor on Chu spaces. We show that if F preserves weak pullbacks, then its associated lifting construction, and also the finite version of it, are functorial on Chu spaces. Then, as a case study in section 4, we show how to realize the Vietoris construction on Stone spaces as a special instance of this Chu construction for the (finite) power set functor (Theorem 4).

Finally, the two parts of the paper come together in Theorem 5, which establishes a tight connection between the Vietoris construction and the axiomatization of the modal ∇-algebras.

2 An Axiomatization of ∇

In the introduction we mentioned that the nabla operator enables a coalgebraic reformulation of standard modal logic. The aim of this section is to substantiate this claim.

First of all, while we introduced the nabla operator as an abbreviation in the language of standard modal logic, for a proper use of the word 'reformulation', we need of course *inter*definability of the nabla operator on the one hand, and the standard modal operators on the other. It is in fact an easy exercise to prove that with the semantics of ∇ as given by (2), we have the following semantic equivalences:

[2] We restrict attention to two-valued Chu spaces in this paper. In fact, these structures are known from the literature under various names, including *topological systems* [18] and *classifications* [5].

$$\Diamond\varphi \equiv \nabla\{\varphi, \top\}$$
$$\Box\varphi \equiv \nabla\varnothing \vee \nabla\{\varphi\} \tag{4}$$

In other words, the standard modalities $\Box$ and $\Diamond$ can be defined in terms of the nabla operator (together with $\vee$ and $\top$).

Taken together, (1) and (4) show that on the *semantic* level of Kripke structures, the language with the nabla operator is indeed a reformulation of standard modal logic. This naturally raises the question whether this equivalence can also be expressed *axiomatically*. That is, we are interested in the question whether we may impose natural conditions which characterize those nablas that behave like the 'real' ones defined using (1). From the semantic interdefinability of ∇ with respect to $\Box$ and $\Diamond$, it follows that a 'roundabout' axiomatization of the nabla operation is possible. However, it is of course much more interesting to try and find a more 'direct' axiomatization, in terms of the intrinsic properties of the nabla operator, and its interaction with the Boolean connectives.

A good starting point for this would be to look for *validities*, i.e., ∇-formulas that are true in every state of every Kripke structure, or, equivalently, for pairs of *equivalent* formulas. As an example of such an equivalence, we give an interesting distributive law; for a concise formulation we need the notion of relation lifting.

Definition 1. *Given a relation $Z \subseteq A \times A'$, define its* power lifting relation *$\overline{\mathsf{P}}Z \subseteq \mathsf{P}A \times \mathsf{P}A'$ as follows:*

$$\overline{\mathsf{P}}Z := \{(X, X') \mid \text{for all } x \in X \text{ there is an } x' \in X' \text{ with } (x, x') \in Z \;\&\; \text{for all } x' \in X' \text{ there is an } x \in X \text{ with } (x, x') \in Z\}.$$

We say that $Z \subseteq A \times A'$ is full *on A and A', notation: $Z \in A \bowtie A'$, if $(A, A') \in \overline{\mathsf{P}}Z$. Observe that as a special case, $\varnothing \bowtie A = \varnothing$ if $A \neq \varnothing$, while $\varnothing \bowtie \varnothing = \{\varnothing\}$ (i.e., the empty relation is full on $\varnothing$ and $\varnothing$).*

The distributive law that we mentioned concerns the following equivalence, which holds for arbitrary sets of formulas Φ, Φ':

$$\nabla\Phi \wedge \nabla\Phi' \;\equiv\; \bigvee_{Z \in \Phi \bowtie \Phi'} \nabla\{\varphi \wedge \varphi' \mid (\varphi, \varphi') \in Z\}. \tag{5}$$

For a proof of (5), first suppose that $\mathbb{S}, s \Vdash \nabla\Phi \wedge \nabla\Phi'$. Let $Z_s \subseteq \Phi \times \Phi'$ consist of those pairs (φ, φ') such that the conjunction $\varphi \wedge \varphi'$ is true at some successor $t \in R[s]$. It is then straightforward to derive from (2) that Z_s is full on Φ and Φ', and that $\mathbb{S}, s \Vdash \nabla\{\varphi \wedge \varphi' \mid (\varphi, \varphi') \in Z_s\}$. The converse direction follows fairly directly from the definitions.

We have now arrived at one of the key definitions of the paper, namely that of nabla algebras. Here we provide the desired direct axiomatization of the nabla operator.

Definition 2. *A structure $A = \langle A, \wedge, \vee, \top, \bot, \nabla\rangle$ is a* positive modal ∇-algebra *if its lattice reduct $A_\flat := \langle A, \wedge, \vee, \top, \bot\rangle$ is a distributive*[3] *lattice and $\nabla : \mathsf{P}_\omega(A) \to A$ satisfies the laws $\nabla 1$ – $\nabla 6$ below. Here Greek lower case letters refer to finite subsets of A.*

[3] In this paper with a 'lattice' we shall always mean a *bounded* lattice.

$\nabla 1$. *If* $\alpha \overline{\mathsf{P}}(\leq)\beta$, *then* $\nabla\alpha \leq \nabla\beta$,
$\nabla 2$. *If* $\bot \in \alpha$, *then* $\nabla\alpha = \bot$,
$\nabla 3$. $\nabla\alpha \wedge \nabla\beta \leq \bigvee\{\nabla\{a \wedge b \mid (a,b) \in Z\} \mid Z \in \alpha \bowtie \beta\}$,
$\nabla 4$. *If* $\top \in \alpha \cap \beta$, *then* $\nabla\{a \vee b \mid a \in \alpha, b \in \beta\} \leq \nabla\alpha \vee \nabla\beta$,
$\nabla 5$. $\nabla\varnothing \vee \nabla\{\top\} = \top$,
$\nabla 6$. $\nabla\alpha \cup \{a \vee b\} \leq \nabla(\alpha \cup \{a\}) \vee \nabla(\alpha \cup \{b\}) \vee \nabla(\alpha \cup \{a,b\})$.

A structure $A = \langle A, \wedge, \vee, \top, \bot, \neg, \nabla\rangle$ *is a* modal ∇-algebra *if* $\langle A, \wedge, \vee, \top, \bot, \neg\rangle$ *is a Boolean algebra and the structure satisfies, in addition to the axioms* $\nabla 1$ – $\nabla 6$ *above, the following*

$\nabla 7$. $\neg\nabla\alpha = \nabla\{\bigwedge\alpha, \top\} \vee \nabla\varnothing \vee \bigvee\{\nabla\{a\} \mid a \in \alpha\}$.

The category of (positive) modal ∇*-algebras with homomorphisms is denoted as* $(\mathsf{P})\mathsf{MA}_\nabla$.

Remark 1. It is not hard to see that the following formulas can be derived from the axioms $\nabla 1$ – $\nabla 6$:

$\nabla 3'$. If $\alpha \neq \varnothing$, then $\nabla\varnothing \wedge \nabla\alpha = \bot$,
$\nabla 3^n$. $\bigwedge_{i=1}^n \nabla\alpha_i \leq \bigvee\{\nabla Z^\wedge \mid Z \in \bigodot_i \alpha_i\}$,
where, for a finite collection α_i of finite subsets of A, $\bigodot_i \alpha_i := \{Z \subseteq \Pi_i\alpha_i \mid \pi_i[Z] = \alpha_i$ for every $i\}$, and, for $Z \in \bigodot_i \alpha_i$, $Z^\wedge := \{\bigwedge_i a_i : (a_i)_{i\in I} \in Z\}$,
$\nabla 6'$. $\nabla\alpha = \nabla\alpha \cup \{\bigvee\alpha\}$,

For instance, $\nabla 3'$ follows by instantiating β with the empty set in $\nabla 3$, and $\nabla 3^n$ is just the n-ary version of $\nabla 3$ and can be shown by induction on n. Recall from Definition 1 that $\varnothing \bowtie \alpha = \varnothing$ in this case, and $\bigvee\varnothing = \bot$.

Definition 3. *A structure* $A = \langle A, \wedge, \vee, \top, \bot, \Diamond, \Box\rangle$ *is a* positive modal algebra *if the* lattice reduct $A_\flat := \langle A, \wedge, \vee, \top, \bot\rangle$ *is a distributive lattice, and* $\Box, \Diamond$ *are unary operations on* A *that satisfy the following axioms:*

$\Diamond(a \vee b) = \Diamond a \vee \Diamond b$ $\quad\Diamond\bot = \bot$
$\Box(a \wedge b) = \Box a \wedge \Box b$ $\quad\Box\top = \top$
$\Box a \wedge \Diamond b \leq \Diamond(a \wedge b)$
$\Box(a \vee b) \leq \Box a \vee \Diamond b$

A modal algebra *is an algebra* $A = \langle A, \wedge, \vee, \top, \bot, \neg, \Diamond, \Box\rangle$ *such that* $A_\flat := \langle A, \wedge, \vee, \top, \bot, \neg\rangle$ *is a Boolean algebra and the operations* $\Box$ *and* $\Diamond$ *satisfy, in addition to the axioms above:*

$\neg\Diamond a = \Box\neg a$.

We let MA *and* PMA *denote the categories of modal algebras (positive modal algebras, respectively) as objects, and algebraic homomorphisms as arrows.*

Definition 4. *Let* A *be a positive modal algebra (modal algebra, respectively). Then we let* A^∇ *denote the structure* $\langle A_\flat, \nabla)\rangle$, *where* ∇ *is defined using (1).*

Conversely, if B *is a positive modal* ∇*-algebra (modal* ∇*-algebra, respectively), we let* $B^\Diamond$ *denote the structure* $\langle B_\flat, \Diamond, \Box\rangle$, *where* $\Diamond$ *and* $\Box$ *are defined using (4).*

We let both $(\cdot)^\nabla$ *and* $(\cdot)^\Diamond$ *operate as the identity on maps, i.e.,* $f^\nabla := f$ *and* $f^\Diamond := f$ *whenever applicable.*

Theorem 1. *The functors* $(\cdot)^\nabla$ *and* $(\cdot)^\Diamond$ *establish a categorical isomorphism between the categories* PMA *and* PMA_∇*, and between the categories* MA *and* MA_∇*.*

Proof. We restrict ourselves to a proof of the following two claims, for an arbitrary positive modal ∇-algebra A:

1. $A^\Diamond$ is a positive modal algebra;
2. $(A^\Diamond)^\nabla \cong A$.

1. $\nabla 2$ implies that $\Diamond\bot = \bot$.
$\nabla 3$ instantiated with $\alpha = \{a\}$ and $\beta = \{b\}$ yields $\Box a \wedge \Box b = \Box(a \wedge b)$.
$\nabla 4$ instantiated with $\alpha = \{a, \top\}$ and $\beta = \{b, \top\}$ yields $\Diamond a \vee \Diamond b = \Diamond(a \vee b)$.
$\nabla 5$ says that $\Box\top = \top$.
$\nabla 6$ instantiated with $\alpha = \emptyset$ yields that $\nabla\{a \vee b\} \leq \nabla\{a\} \vee \nabla\{b\} \vee \nabla\{a, b\}$. Since $\{b\}\overline{\mathsf{P}}(\leq)\{b, \top\}$ and $\{a, b\}\overline{\mathsf{P}}(\leq)\{b, \top\}$, then by $\nabla 1$, $\nabla\{b\} \vee \nabla\{a, b\} \leq \nabla\{b, \top\}$. Hence $\nabla\{a \vee b\} \leq \nabla\{a\} \vee \nabla\{b, \top\}$, from which we get $\Box(a \vee b) \leq \Box a \vee \Diamond b$. In order to show that $\Box a \wedge \Diamond b \leq \Diamond(a \wedge b)$ we need to show that $(\nabla\{a\} \vee \nabla\emptyset) \wedge \nabla\{b, \top\} \leq \nabla\{a \wedge b, \top\}$.

$$\begin{aligned}
(\nabla\{a\} \vee \nabla\emptyset) \wedge \nabla\{b, \top\} &= [\nabla\{a\} \wedge \nabla\{b, \top\}] \vee [\nabla\emptyset \wedge \nabla\{b, \top\}] \\
&= [\nabla\{a\} \wedge \nabla\{b, \top\}] \vee \bot && (\nabla 3') \\
&= \nabla\{a \wedge b, a \wedge \top\} && (\nabla 3) \\
&\leq \nabla\{a \wedge b, \top\} && (\nabla 1)
\end{aligned}$$

The last inequality holds since $\{a \wedge b, a\}\overline{\mathsf{P}}(\leq)\{a \wedge b, \top\}$. This completes the proof that $A^\Diamond$ is a positive modal algebra.

2. We need to show that ∇ coincides with the operator $\widetilde{\nabla}$ associated with the ∇-induced modal operators. For every finite subset α of A,

$$\begin{aligned}
\widetilde{\nabla}\alpha &= [\nabla\{\bigvee\alpha\} \vee \nabla\emptyset] \wedge \bigwedge\{\nabla\{a, \top\} : a \in \alpha\} \\
&= \bigwedge\{[\nabla\{\bigvee\alpha\} \vee \nabla\emptyset] \wedge \nabla\{a, \top\} : a \in \alpha\} \\
&= \bigwedge\{[\nabla\{\bigvee\alpha\} \wedge \nabla\{a, \top\}] \vee [\nabla\emptyset \wedge \nabla\{a, \top\}] : a \in \alpha\} \\
&= \bigwedge\{[\nabla\{\bigvee\alpha\} \wedge \nabla\{a, \top\}] \vee \bot : a \in \alpha\} && (\nabla 3') \\
&= \bigwedge\{[\nabla\{\bigvee\alpha, a\}] : a \in \alpha\} && (\nabla 3) \\
&= \bigvee\{\nabla Z^\wedge : Z \in \textstyle\bigodot_{a\in\alpha}\{\bigvee\alpha, a\}\} && (\nabla 3^n) \\
&= \nabla\alpha \cup \{\bigvee\alpha\} && (*) \\
&= \nabla\alpha && (\nabla 6')
\end{aligned}$$

Let us show the $(*)$-marked equality: let $\alpha = \{a_i, i = 1 \ldots, n\}$ for every i, let $\beta_i = \{\bigvee\alpha, a_i\}$. Then consider the following relation:

$$Z = \{(b_i)_i \in \Pi_i\beta_i : \text{for at most one } i, b_i \neq \bigvee\alpha\}.$$

Then $Z^\wedge = \{\bigvee\alpha, a_1, \ldots, a_n\}$, and moreover it is not difficult to see that for every $W \in \bigodot_i \beta_i$, the pair $(W^\wedge, Z^\wedge)$ belongs to the relation $\overline{\mathsf{P}}(\leq)$, so the statement follows by $\nabla 1$. □

Remark 2. As an easy corollary of Theorem 1, we can obtain a completeness result for modal logic formulated in terms of the nabla operator.

Finally, in section 4 we will need the construction which can be seen as a kind of *power set lifting* of a Boolean algebra A. Following terminology and notation of [18], in the definition below we present a Boolean algebra by generators and relations.

Definition 5. *Let A be a Boolean algebra. Then*

$$\mathsf{BA}\langle\ \{\nabla\alpha \mid \alpha \in \mathsf{P}_\omega A\} : \nabla 1 - \nabla 7\ \rangle$$

presents a Boolean algebra that we shall denote as A^{P}.

In words, A^{P} is the Boolean algebra we obtain as follows: first freely generate a Boolean algebra by taking the set $\{\nabla\alpha \mid \alpha \in \mathsf{P}_\omega A\}$ as *generators*, and then take a quotient of this algebra, by identify those elements that can be proven equal on the basis of the *relations* (axioms) $\nabla 1$–$\nabla 7$. In section 4 we will see a different characterization of this algebra: Theorem 5 states that A^{P} is in fact isomorphic to the algebra of clopens of the Vietoris hyperspace of the Stone space which is dual to A.

3 Chu Spaces and Their Liftings

Chu spaces [16] unify a wide range of mathematical structures, including relational, algebraic and topological ones. Surprisingly this degree of generality can be achieved with a remarkably simple form of structure. As we mentioned already, in this paper we will only enter a small part of Chu territory since we restrict attention to two-valued Chu spaces. These can be defined as follows.

Definition 6. *A (two-valued) Chu space is a triple $\mathbb{S} = \langle X, S, A\rangle$ consisting of two sets X and A, together with a binary relation $S \subseteq X \times A$. Elements of X are called* objects *or* points*, and elements of A,* attributes*; the relation S is the* matrix *of the space. Given two Chu spaces $\mathbb{S}' = \langle X', S', A'\rangle$ and $\mathbb{S} = \langle X, S, A\rangle$, a* Chu transform *from $\mathbb{S}'$ to $\mathbb{S}$ is a pair (f, f') of functions $f : X' \to X$, $f' : A \to A'$ that satisfy the* (generalized) adjointness condition

$$f(x')Sa \iff xS'f'(a). \tag{6}$$

for all $x' \in X'$ and $a \in A$. We let Chu denote the category with Chu spaces as objects and Chu transforms as arrows.

As a motivating example of a Chu transform, consider once more the semantics of ∇. One may read (3) above as saying that the pair $(R[\cdot] : S \to \mathsf{P}(S), \nabla : \mathsf{P}_\omega(Fma) \to Fma)$ is a *Chu transform* from the Chu space $(S, \Vdash, Fma)$ to its power set lifting $(\mathsf{P}S, \overline{\mathsf{P}}(\Vdash), \mathsf{P}_\omega(Fma))$. In a slogan: the semantics of ∇ is an arrow in the category Chu of Chu spaces and Chu transforms.

Clearly, the generalized adjointness condition specializes to adjointness in the right context, for example if partial orders $\langle P, \leq, \rangle$ are represented as the Chu spaces $\langle P, \leq, P\rangle$, then the Chu transforms between two such structures are exactly tuples of residuated maps.

Definition 7. *Any Chu space $S = \langle X, S, A\rangle$ gives rise to an order on X, the* specialization order $\sqsubseteq_S$, *defined as follows: $x \sqsubseteq_S y$ iff for every $a \in A$ $(xSa \Rightarrow ySa)$. The specialization order then induces the following equivalence relation $\equiv_S$ on X: $x \equiv_S y$ iff $x \sqsubseteq_S y$ and $y \sqsubseteq_S x$, i.e. iff for every $a \in A$ $(xSa \Leftrightarrow ySa)$.*

Normal Chu Spaces. A prominent role within Chu, from the point of view of logic, is played by the so-called *normal* Chu spaces. Normal Chu spaces provide a general and uniform setting for algebraic, set-based and topological semantics of propositional logics.

Definition 8. *A Chu space $S = \langle X, S, A\rangle$ is* normal *if $A \subseteq \mathsf{P}(X)$ and S is the membership relation restricted to A, that is, xSa iff $x \in a$. NChu denotes the full subcategory of Chu based on these normal spaces.*

To mention an important example, any Stone space $X = \langle X, \tau\rangle$ can be represented as $S_X = \langle X, \in, C\rangle$, C being the Boolean algebra of the clopen subsets in τ. Then a map f between Stone spaces is continuous exactly when (f, f^{-1}) is a Chu transform between their associated Chu spaces. In fact, any Chu transform from one normal Chu space to another is of the form (f, f^{-1}).

Since not all our constructions on Chu spaces preserve normality, we shall need a *normalization* operation on Chu spaces.

Definition 9. *Given a Chu space $S = \langle X, S, A\rangle$ define the map $E_S : A \to \mathsf{P}(X)$ by putting $E_S(a) := S^{-1}[a]$, that is:*

$$E_S(a) := \{x \in X \mid xSa\}.$$

Then the normalization *of S is given as the structure $\mathsf{N}(S) = \langle X, \in, E_S[A]\rangle$. Extending this definition to transforms, we define the normalization $\mathsf{N}(f, f')$ of a Chu transform $(f, f') : S' \to S$ as the pair (f, f^{-1}).*

Proposition 1. *The normalization construction N is a functor from Chu to NChu.*

Proof. We confine ourselves to checking that the normalization of a Chu transform is again a Chu transform. Suppose that $(f, f') : S' \to S$ is a Chu transform from $S' = \langle X', S', A'\rangle$ to $S = \langle X, S, A\rangle$. It is obvious that any elements $x' \in X'$ and $Y \in E_S[A]$ satisfy the adjointness condition (6) with respect to f and f^{-1}. The point is to prove that f^{-1} is a well-defined map from $E_S[A]$ to $E_{S'}[A']$. For this purpose, take an arbitrary element $E_S(a) = S^{-1}[a] \in E_S[A]$. Then

$$\begin{aligned} f^{-1}(S^{-1}[a]) &= \{x' \in X' \mid f(x')Sa\} \\ &= \{x' \in X' \mid x'S'f'(a)\} \\ &= (S')^{-1}[f'(a)] \\ &= E_{S'}(a), \end{aligned}$$

which shows that, indeed, $f^{-1}(S^{-1}[a])$ belongs to $E_{S'}[A']$. □

Strongly Normal Chu Spaces. In the next section we will be interested in Chu spaces that satisfy a strong form of normality that we will describe now. Normal Chu spaces are extensional in that every attribute is completely determined by the set of objects that it is related to, but they do not necessarily satisfy the dual property of *separation*.

Definition 10. *A Chu space* $S = \langle X, S, A\rangle$ *is* separated *if for every distinct pair of points* x *and* y *in* X *there is an attribute* $a \in A$ *separating* x *from* y*, in the sense that it is either related to* x *and not to* y*, or related to* y *and not to* x*.*

The following, slightly technical definition will be of use in section 4, when we will understand the Vietoris construction as a special power lifting construction:

Definition 11. *Let* $S = \langle X, S, A\rangle$ *be a Chu space. A subset* $Y \subseteq X$ *is called a* representative subset of X*, if* Y *contains exactly one representant of every* $\equiv_S$*-cell of* S *(where* $\equiv_S$ *is as defined in Definition 7). For any such* Y*, the* strong normalization $\mathsf{N}^Y(S)$ *of* S *based on* Y *is the Chu space* $\langle Y, \in, E^Y_S[A]\rangle$*, where* $E^Y_S : A \to \mathsf{P}(Y)$ *is the map given by* $E^Y_S(a) := \{y \in Y \mid ySa\}$*.*

It is not difficult to prove that for any representative subset Y of X the Chu space $\mathsf{N}^Y(S)$ is strongly normal, and that the pair (ι_{YX}, E^Y_S) (with ι_{XY} the inclusion) is a Chu transform from $\mathsf{N}^Y(S)$ to S.

Remark 3. One of the referees pointed out that Proposition 1 can be expanded to state that NChu is a coreflective subcategory of Chu, and that separated Chu spaces form a reflective subcategory of Chu, cf. [2].

Lifting Chu Spaces. Many category-theoretic operations can be defined on Chu spaces, for instance orthogonality, tensor product, transposition (see [16] for an overview). Here our focus will be on lifting constructions; our aim is to define, for an arbitrary set functor F and for an arbitrary Chu space $S = \langle X, \in, A\rangle$, a Chu space $\widetilde{\mathsf{F}}(S)$ which is based on the set $\mathsf{F}(X)$. Although we are mainly interested in a lifting construction for normal Chu spaces, we take a little detour to first define a functorial power lifting construction on the full category Chu. For that purpose we need the notion of relation lifting for an arbitrary set functor. Recall that in Definition 1 we gave the power set lifting of a binary relation.

For the definition of relation lifting with respect to a general set functor F, consider a binary relation $Z \subseteq S \times S'$, with associated projections π, π':

$$S \stackrel{\pi}{\longleftarrow} Z \stackrel{\pi'}{\longrightarrow} S'$$

Applying F to this diagram we obtain

$$\mathsf{F}S \stackrel{\mathsf{F}\pi}{\longleftarrow} \mathsf{F}Z \stackrel{\mathsf{F}\pi'}{\longrightarrow} \mathsf{F}S'$$

so that by the properties of the product $\mathsf{F}S \times \mathsf{F}S'$, we may consider the product map $(\mathsf{F}\pi, \mathsf{F}\pi') : \mathsf{F}Z \to \mathsf{F}S \times \mathsf{F}S'$. This map need not be an inclusion (or even an injection), and $\mathsf{F}Z$ need not be a binary relation between $\mathsf{F}S$ and $\mathsf{F}S'$. However, we may consider the *range* $\overline{\mathsf{F}}(Z)$ of the map $(\mathsf{F}\pi, \mathsf{F}\pi')$ which is of the right shape.

Definition 12. *Let* F *be a set functor. Given two sets* S *and* S'*, and a binary relation* Z *between* $S \times S'$*, we define the* lifted relation $\overline{\mathsf{F}}(Z) \subseteq \mathsf{F}S \times \mathsf{F}S'$ *as follows:*

$$\overline{\mathsf{F}}(Z) := \{((\mathsf{F}\pi)(\varphi), (\mathsf{F}\pi')(\varphi)) \mid \varphi \in \mathsf{F}Z\},$$

where $\pi : Z \to S$ *and* $\pi' : Z \to S'$ *are the projection functions given by* $\pi(s, s') = s$ *and* $\pi'(s, s') = s'$*.*

Definition 13. *Let* F *be a set functor, and let* $S = \langle X, S, A\rangle$ *be a Chu space. Then we define the* F-lifting of S *to be the Chu space*

$$\widetilde{\mathsf{F}}S := \langle \mathsf{F}(X), \overline{\mathsf{F}}(S), \mathsf{F}(A)\rangle.$$

Given a Chu transform (f, f') *from* $S' = \langle X', S', A'\rangle$ *to* $S = \langle X, S, A\rangle$*, we define* $\widetilde{\mathsf{F}}(f, f')$ *as the pair* $(\mathsf{F}f, \mathsf{F}f')$ *of maps.*

We need some of the properties of relation lifting. Given a function $f : A \to B$, we let $Gr(f)$ denote the graph of f, i.e., $Gr(f) := \{(a, b) \in A \times B \mid b = f(a)\}$.

Fact 2. *Let* F *be a set functor. Then the relation lifting* $\overline{\mathsf{F}}$ *satisfies the following properties, for all functions* $f : S \to S'$*, all relations* $R, Q \subseteq S \times S'$*, and all subsets* $T \subseteq S$*,* $T' \subseteq S'$*:*
(1) $\overline{\mathsf{F}}$ *extends* F*:* $\overline{\mathsf{F}}(Gr(f)) = Gr(\mathsf{F}f)$*;*
(2) $\overline{\mathsf{F}}$ *preserves the diagonal:* $\overline{\mathsf{F}}(\mathsf{Id}_S) = \mathsf{Id}_{\mathsf{F}S}$*;*
(3) $\overline{\mathsf{F}}$ *commutes with relation converse:* $\overline{\mathsf{F}}(R^{\smile}) = (\overline{\mathsf{F}}R)^{\smile}$*;*
(4) $\overline{\mathsf{F}}$ *is monotone: if* $R \subseteq Q$ *then* $\overline{\mathsf{F}}(R) \subseteq \overline{\mathsf{F}}(Q)$*;*
(5) $\overline{\mathsf{F}}$ *distributes over composition:* $\overline{\mathsf{F}}(R \circ Q) = \overline{\mathsf{F}}(R) \circ \overline{\mathsf{F}}(Q)$*, if* F *preserves weak pullbacks.*

For proofs we refer to [14,3], and references therein. The proof that Fact 2(5) depends on the property of weak pullback preservation goes back to Trnková [17].

Theorem 3. *If* F *preserves weak pullbacks, then* $\widetilde{\mathsf{F}}$ *is an endofunctor on* Chu*.*

Proof. We restrict our proof to showing that $\widetilde{\mathsf{F}}$ turns Chu transforms into Chu transforms. Let $S' = \langle X', S', A'\rangle$ and $S = \langle X, S, A\rangle$ be two Chu spaces, and let $f : X' \to X$ and $f' : A \to A'$ be two maps. It is easily verified that (f, f') is a Chu transform iff

$$Gr(f) \circ S' = (Gr(f') \circ S)^{\smile}.$$

But then it follows from the properties of relation lifting for weak pullback preserving functors that

$$\begin{aligned} Gr(\mathsf{F}f)) \circ \overline{\mathsf{F}}(S') &= \overline{\mathsf{F}}(Gr(f) \circ S') \\ &= \overline{\mathsf{F}}((Gr(f') \circ S)^{\smile}) \\ &= (Gr(\mathsf{F}f') \circ \overline{\mathsf{F}}(S))^{\smile}. \end{aligned}$$

In other words, $(\mathsf{F}f, \mathsf{F}f')$ is a Chu transform as well. □

Unfortunately, normality of a Chu space is not preserved under taking liftings of Chu spaces. But clearly, we can combine lifting with normalization.

Definition 14. *Assume that* F *preserves weak pullbacks. Then* $\widehat{\mathsf{F}}$ *denotes the endofunctor on* NChu *defined by* $\widehat{\mathsf{F}} := \mathsf{N} \circ \widetilde{\mathsf{F}}$.

Remark 4. It will be useful in the next section to have a more concrete definition of the normalization operation for Chu spaces of the form $\widetilde{\mathsf{F}}S$, where S is normal. Suppose that $S = \langle X, \in, A \rangle$, then $\widetilde{\mathsf{F}}S = \langle \mathsf{F}(X), \overline{\in}, \mathsf{F}(A) \rangle$, where we write $\overline{\in}$ for the lifted membership relation $\overline{\mathsf{F}}(\in)$. Now the normalization map $E_{\overline{\in}}$ is given by

$$E_{\overline{\in}}[\alpha] = \{\varphi \in \mathsf{F}(X) \mid \varphi \overline{\in} \alpha\}.$$

4 Stone Spaces

As a case study, let us show how the Vietoris construction on Stone spaces naturally arises in the framework that we have developed in the previous two sections.

As we mentioned earlier, any Stone space $\langle X, \tau \rangle$ can be represented as a Chu space $S = \langle X, \in, A \rangle$. The Boolean algebra A of the clopen subsets of X is a base for the topology τ, so for instance, for every $Y \subseteq X$, the τ-closure of Y is $Y^{\bullet} = \bigcap\{a \in A \mid Y \subseteq a\}$.

Definition 15. *Given a Stone space* $S = \langle X, \in, A \rangle$*, we let* $K(S)$ *denote the collection of closed sets of* S*. We define the operations* $\langle \ni \rangle, [\ni] : \mathsf{P}(X) \to \mathsf{P}(K(S))$ *by*

$$[\ni]a := \{F \in K(S) \mid F \subseteq a\}\,,$$
$$\langle \ni \rangle a := \{F \in K(S) \mid F \cap a \neq \varnothing\}\,.$$

We let $V(A)$ *denote the Boolean subalgebra of* $\mathsf{P}(K(S))$ *generated by the set* $\{\langle \ni \rangle a, [\ni]a \mid a \in A\}$. $V(A)$ *is the Boolean algebra of clopen subsets of the* Vietoris topology *on* $K(S)$.

Modal logicians will recognize the above notation as indicating that $[\ni]$ and $\langle \ni \rangle$ are the 'box' and the 'diamond' associated with the converse membership relation $\ni \subseteq K(S) \times X$.

It is well-known that the Vietoris hyperspace of a Stone space $S = \langle X, \in, A \rangle$ is a Stone space, so $V(A)$ is a base for the Vietoris topology. Then $V(S) = \langle K(S), \in, V(A) \rangle$ is the Chu-representation of the Vietoris hyperspace of S.

For the remainder of this section fix a Stone space $S = \langle X, \in, A \rangle$. Here is a summary of our approach:

1. First, as a minor variation on Chu power set lifting, consider the Chu space $\widetilde{\mathsf{P}}_{\omega}(S) := \langle \mathsf{P}(X), \overline{\in}, \mathsf{P}_{\omega}(A) \rangle$ where $\overline{\in}$ denotes the relation lifting $\overline{\mathsf{P}}(\in)$, restricted to $\mathsf{P}(X) \times \mathsf{P}_{\omega}(A)$. Thus the variation consists in taking the *finite* power set $\mathsf{P}_{\omega}(A)$ rather than the full power set $\mathsf{P}(A)$.

2. We then show that every equivalence class of the relation $\equiv_{\overline{\in}}$ contains exactly one closed element, so that we may take the collection $K(S)$ as the 'canonical representants' in order to define a strong normalization $\widehat{\mathsf{P}}_\omega := \langle K(S), \in, Q\rangle$ of $\langle \mathsf{P}(X), \overline{\in}, \mathsf{P}_\omega(A)\rangle$ (see Definition 11).
3. We then prove that the Boolean algebra generated by Q is *identical* to the Vietoris algebra $V(A)$.
4. Finally we prove that the Vietoris algebra is isomorphic to the algebra A^{P} (defined in section 2 as the Boolean algebra generated by the set $\{\nabla\alpha \mid \alpha \in \mathsf{P}_\omega(A)\}$ modulo the ∇ axioms).

Definition 16. *Given a Stone space $S = \langle X, \in, A\rangle$, let $\overline{\in}$ denote the relation lifting $\overline{\mathsf{P}}(\in)$, restricted to $\mathsf{P}(X) \times \mathsf{P}_\omega(A)$. Define the Chu space $\widetilde{\mathsf{P}}_\omega(S)$ as the structure $\langle \mathsf{P}(X), \overline{\in}, \mathsf{P}_\omega(A)\rangle$.*

For the following proposition, recall that the *closure* of a set $Y \subseteq X$ is denoted by $Y^\bullet$, and that for any Chu space $T = \langle P, T, B\rangle$, the specialization order $\sqsubseteq_T$ on X induced by T is given by $p \sqsubseteq_T q$ iff $(pTb \Rightarrow qTb)$ for all $b \in B$.

Proposition 2. *Let $\sqsubseteq$ and $\equiv$ be the specialization order and the equivalence relation associated with the Chu space $\widetilde{\mathsf{P}}_\omega(S)$, respectively. Then, for every set $Y \in \mathsf{P}(X)$, its closure $Y^\bullet$ is the maximum element of the $\equiv$-cell $Y/{\equiv}$. In particular, $Y^\bullet$ is the unique closed set in $Y/{\equiv}$.*

Proof. Clearly it suffices to prove that

$$Y \sqsubseteq Z \Rightarrow Y^\bullet \subseteq Z^\bullet \tag{7}$$

and

$$Y \equiv Y^\bullet. \tag{8}$$

For (7), suppose that $Y^\bullet \not\subseteq Z^\bullet = \bigcap\{a \in A \mid Z \subseteq a\}$. Then, since every clopen is closed and $Y^\bullet$ is the smallest closed set that contains Y, there must be some $a \in A$ such that $Z \subseteq a$ and $Y \not\subseteq a$. Let $\alpha = \{\neg a, X\} \in \mathsf{P}_\omega(A)$: it holds that $Y \overline{\in} \alpha$ but $Z \overline{\notin} \alpha$, hence $Y \not\sqsubseteq Z$.

For (8), if $\alpha \in \mathsf{P}_\omega(A)$ and $Y \overline{\in} \alpha$, then $a \cap Y \neq \varnothing$ for every $a \in \alpha$ and $Y \subseteq \bigcup \alpha$. Then, as $Y \subseteq Y^\bullet$, we get $a \cap Y^\bullet \neq \varnothing$ for every $a \in \alpha$. Also, as $\bigcup \alpha \in A$ is in particular closed, from $Y \subseteq \bigcup\alpha$ we get $Y^\bullet \subseteq \bigcup\alpha$, which proves that $Y^\bullet \overline{\in} \alpha$. This shows that $Y \sqsubseteq Y^\bullet$. Conversely, if $Y^\bullet \overline{\in} \alpha$ then $Y \subseteq Y^\bullet \subseteq \bigcup\alpha$. In addition, for every $a \in A$, $Y^\bullet \cap a \neq \varnothing$ implies $Y \cap a \neq \varnothing$, for if not, then $Y \subseteq \neg a \in K(S)$, which would imply $Y^\bullet \subseteq \neg a$, contradiction. □

The proposition above says that $K(S)$ is a representative subset of $\mathsf{P}(X)$ (see Definition 11). So we can consider the strong normalization of $\widetilde{\mathsf{P}}_\omega(S)$:

Definition 17. *Given a Stone space $S = \langle X, \in, A\rangle$, define $\widehat{\mathsf{P}}_\omega(S)$ as the strong normalization of $\widetilde{\mathsf{P}}_\omega(S)$ w.r.t. $K(S)$, i.e. $\widehat{\mathsf{P}}_\omega(S)$ is the normal and separated Chu space*

$$\langle K(S), \in, Q\rangle,$$

where $Q = E[\mathsf{P}_\omega(A)]$ and $E : \mathsf{P}_\omega(A) \to \mathsf{P}(K(S))$ is the map given by $E(\alpha) := \{F \in K(S) \mid F \mathrel{\overline{\in}} \alpha\}$.

The following theorem states that the Vietoris construction of a Boolean space can indeed be seen as an instance of power lifting.

Theorem 4. *Let $S = \langle X, \in, A\rangle$ be a Stone space. Then $V(A)$ is the Boolean algebra generated by the set $Q \in \mathsf{P}(K(S))$, where Q is the set of attributes of $\widehat{\mathsf{P}}_\omega(S)$.*

Proof. For every $F \in K(S)$ and every $\alpha \in \mathsf{P}_\omega(A)$,

$$F \in E(\alpha) \text{ iff } F \mathrel{\overline{\in}} \alpha \text{ iff } F \in [\ni](\bigcup \alpha) \cap \bigcap_{a\in\alpha} \langle \ni \rangle a,$$

which means that E is the nabla operator defined from $[\ni]$ and $\langle \ni \rangle$. Hence $Q \subseteq V(A)$, and moreover for every $a \in A$, $[\ni]a = E(\{a\}) \cup E(\varnothing)$ and $\langle \ni \rangle a = E(\{a, \top\})$, which makes Q a set of generators for $V(A)$. □

To see where does the material of the second section come in: The ∇-axioms are an important ingredient for the following representation theorem for the lifted Boolean algebra A^{P}:

Theorem 5. *Let $S = \langle X, \in, A\rangle$ be a Stone space. Then $V(A)$ is isomorphic to the power lifting A^{P} of A.*

Proof. It is not difficult to see, given the axioms $\nabla 1$–$\nabla 7$, that an arbitrary element of A^{P} can be represented as a finite *join* of generators. Now define the map $\rho : A^{\mathsf{P}} \to \mathsf{P}(K(S))$ by putting

$$\rho(\nabla\alpha_1 \vee \ldots \vee \nabla\alpha_n) := E(\alpha_1) \cup \ldots \cup E(\alpha_n),$$

where E is the strong normalization map of $\widehat{\mathsf{P}}_\omega(S)$, given by

$$E(\alpha) = \{F \in K(S) \mid F \mathrel{\overline{\in}} \alpha\}.$$

In order to establish the theorem, it suffices to prove our claim that

$$\rho \text{ is an isomorphism.}$$

We omit the argument why ρ is a homomorphism and only sketch the proof that it is an injection. Given a homomorphism between two Boolean algebras, in order to prove injectivity, it suffices to show that the homomorphism maps nonzero elements to nonzero elements. So let $\nabla\alpha_1 \vee \ldots \vee \nabla\alpha_n$ be an arbitrary nonzero element of A^{P}, then at least one of the $\nabla\alpha_i$, say $\nabla\alpha$, is nonzero. Then it follows from axiom $\nabla 2$ that $\bot \notin \alpha$.

Now consider the set $Y = \bigcup \alpha$. Y is a finite union of clopens and hence, certainly closed. Since $\bot \notin \alpha$, it is also straightforward to verify that $Y \mathrel{\overline{\in}} \alpha$. In other words, we have found that $Y \in E(\alpha) = \rho(\nabla\alpha) \subseteq \rho(\nabla\alpha_1 \vee \ldots \vee \nabla\alpha_n)$. Hence we have proved indeed that ρ maps an arbitrary nonzero element of A^{P} to a nonempty set of closed elements, i.e., a nonzero element of the algebra $V(A)$. □

The point of carrying out the Vietoris construction in terms of the nabla operator rather than the standard modalities $[\ni]$ and $\langle \ni \rangle$ is that the former is coalgebraic in nature, and the latter are not. This will be advantageous when it comes to generalizing the Vietoris construction to other functors (and other categories).

5 Conclusions

We presented an algebraic study of the coalgebraic modal operator ∇, and we related this to a presentation of the Vietoris power construction on Stone spaces. We believe the main contribution of the paper to be threefold. First, on the algebraic side, we gave an axiomatization for ∇ that characterizes the class of ∇-algebras that is category-theoretically isomorphic (see Theorem 1) to the (positive) modal algebras. Second, using the concept of relation lifting, we showed how an arbitrary set functor F naturally gives rise to various lifting constructions on the category Chu of two-valued Chu spaces. These constructions are functorial in case F preserves weak pullbacks (Theorem 3). And finally, we showed how to realize the Vietoris construction on Stone spaces as a special instance of this Chu construction for the (finite) power set functor (Theorem 4), and linked this approach to the axiomatization of the modal ∇-algebras (Theorem 5).

In the future we hope to expand the work presented here in various directions. Because of space limitations we have to be brief.

1. First of all, there is no strong reason to confine ourselves to a finitary setting. The first natural generalization of this work is to move to a 'localic' setting and study the case of logical languages with infinitary conjunctions and/or disjunctions, and an infinitary version of the nabla operator.
2. In such a generalized setting, it would make sense to look at power constructions for other topologies than just Stone spaces, and to formalize these constructions not in terms of clopens but in terms of closed or open sets.
3. We think it is very interesting to try and generalize the results in this paper to other functors than P. This is the reason why we have taken care to formulate all our results as generally as possible, see for instance our remark following the proof of Theorem 5.
4. We already mentioned in the introduction that our first motivation was to pave the way for a 'coalgebraic proof theory', by which we mean to try and give an algebraic and syntactic account of nabla operators associated with weak pullback-preserving endofunctors. As a first step in this direction, we are currently working on a Gentzen-style derivation system for the modal nabla operator.

References

1. Abramsky, S.: A Cooks Tour of the Finitary Non-Well-Founded Sets. Invited Lecture at BCTCS (1988)
2. Barr, M.: The separated extensional Chu category. Theory and Applications of Categories 4, 137–147 (1998)
3. Baltag, A.: A Logic for Coalgebraic Simulation. Electronic Notes in Theoretical Computer Science 33, 41–60 (2000)
4. Barwise, J., Moss, L.: Vicious circles. CSLI Publications, Stanford (1997)
5. Barwise, J., Seligman, J.: Information Flow: the Logic of Distributed Systems. Cambridge University Press, Cambridge (1997)

6. Bonsangue, M., Kurz, A.: Dualities for Logics of Transition Systems. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 455–469. Springer, Heidelberg (2005)
7. Celani, S., Jansana, R.: Priestley duality, a Sahlqvist theorem and a Goldblatt-Thomason theorem for positive modal logic. Logic Journal of the IGPL 7, 683–715 (1999)
8. Fine, K.: Normal forms in modal logic. Notre Dame Journal of Formal Logic 16, 229–234 (1975)
9. Hodges, W.: Model Theory. Cambridge University Press, Cambridge (1993)
10. Janin, D., Walukiewicz, I.: Automata for the modal μ-calculus and related results. In: Hájek, P., Wiedermann, J. (eds.) MFCS 1995. LNCS, vol. 969, pp. 552–562. Springer, Heidelberg (1995)
11. Johnstone, P.: Stone Spaces. Cambridge University Press, Cambridge (1982)
12. Kupke, C., Kurz, A., Venema, Y.: Stone Coalgebras. Theoretical Computer Science 327, 109–134 (2004)
13. Michael, E.: Topologies on spaces of subsets. Transactions of the American Mathematical Society 71, 152–182 (1951)
14. Moss, L.: Coalgebraic logic. Annals of Pure and Applied Logic 96, 277–317 (1999) Erratum published APAL 99, 241–259 (1999)
15. Palmigiano, A.: A coalgebraic view on Positive Modal Logic. Theoretical Computer Science 327, 175–195 (2004)
16. Pratt, V.R.: Chu Spaces, Notes for School on Category Theory and Applications. University of Coimbra, Portugal (July 13-17, 1999)
17. Trnková, V.: Relational automata in a category and theory of languages. In: Karpinski, M. (ed.) Fundamentals of Computation Theory. LNCS, vol. 56, pp. 340–355. Springer, Heidelberg (1977)
18. Vickers, S.: Topology Via Logic, vol. 5. Cambridge University Press, Cambridge (1989)

An Institutional Version of Gödel's Completeness Theorem

Marius Petria

University of Edinburgh*

Abstract. Gödel's famous result about the completeness of first order deduction can be cast in the general framework of institutions. For this we use Henkin's method of proving completeness which is very generic and has been exploited over time by producing similar proofs of completeness for various logical systems. This paper sets out a general framework with the purpose to incorporates many of these proofs as examples. As a consequence of this abstraction, the completeness theorem becomes available for many "first order" logical systems that appear in the area of logic or computer science.

1 Summary

The goal of this paper is to express and prove Gödel's completeness theorem in the institutional framework and then apply the general results to a couple of different institutions. The proof needs a small and natural set of assumptions. There is however a condition that seems overly restrictive: that the signatures do not allow void sorts. We plan to investigate further the elimination of this condition such that the theorem can be applied to first order logics with void sorts.

This approach has two main motivations. Firstly, it allows us to obtain in an uniform way completeness results for many specific logics. Because of space restrictions we treat only the case of first order logic and that of partial algebras, but in an extended version we want to include more cases.

Secondly, the set of conditions that are sufficient for a completeness result helps in understanding abstractly what it means for a logic to be "first order". Also, the way the result is obtained by separating the deduction rules into two sets, rules that deal with the specific syntax of the atomic sentences and generic rules that deal with logical connectives and quantification, supports the view that "first order logics" freely extend "atomic logics" in all regards: syntactically, semantically and as proof systems, and completeness is preserved by this free extension. A way to formalise this by means of institution morphisms is left for further investigation.

The paper is organised as follows. Section 2 introduces the abstract definitions used to represent a logical system: institutions and entailment systems. Section 3

* On leave from the Institute of Mathematics of the Romanian Academy, Bucharest.

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 409–424, 2007.
© Springer-Verlag Berlin Heidelberg 2007

enumerates some general techniques enhancing the internal logic of an institution. Section 4 sets the framework and expresses more specifically what is the goal of this paper. We aim to prove that there is a canonical way to obtain a complete entailment system for a "first order" institution building upon a complete entailment system for atomic sentences. The following sections solve this by redoing Henkin's proof [8] of completeness for first order logic in the institutional framework. Section 5 shows that any consistent theory can be extended to a maximal Henkin theory, i.e. one that has a constant witness for every existentially quantified sentence that it satisfies. The construction of a Henkin theory follows a suggestion made by Henkin at the end of [9]. It is worth pointing that the extension to a signature that is "big enough" can be done in only one step when it is done for the concrete case of first order logic, but in the abstract framework we prefer to do the construction in an iterative fashion. Section 5 deals with satisfiability of Henkin theories. It is shown that every maximal Henkin theory has a model. This section uses the property of *basic coverage* (Definition 13) as a feature of first order logic that distinguishes it from second order logic. This property is the only condition of the completeness theorem that second order logic fails to satisfy. Section 7 just states the completeness theorem and lists all the sufficient conditions for it to hold. Section 8 shows how the theorem can be applied to produce particular completeness theorems for institutions of first order logic and partial algebras.

2 Model Theory in Institutions

2.1 Categories

We assume the reader is familiar with basic notions and standard notations from category theory; e.g., see [11] for an introduction to this subject. By way of notation, $|\mathbb{C}|$ denotes the class of objects of a category $\mathbb{C}$, $\mathbb{C}(A, B)$ the set of arrows with domain A and codomain B, and composition is denoted by ";" and in diagrammatic order. The category of sets (as objects) and functions (as arrows) is denoted by $\mathbb{S}et$, and $\mathbb{CAT}$ is the category of all categories.[1] The opposite of a category $\mathbb{C}$ (obtained by reversing the arrows of $\mathbb{C}$) is denoted $\mathbb{C}^{\mathrm{op}}$.

Given a functor $\mathcal{U}{:}\mathbb{C}' \to \mathbb{C}$, for any object $A \in |\mathbb{C}|$, the *comma category* $A/\mathcal{U}$ has arrows $f{:}A \to \mathcal{U}(B)$ as objects (sometimes denoted as (f, B)) and $h \in \mathbb{C}'(B, B')$ with $f;\mathcal{U}(h) = f'$ as arrows $(f, B) \to (f', B')$.

A *J-(co)limit* in a category $\mathbb{C}$ is a (co)limit of a functor $J \to \mathbb{C}$. When J are directed partial orders, respectively total orders, the J-colimits are called *directed* colimits, respectively *inductive* colimits.

A standard categorical approach to finiteness is provided by the concept of finitely presented object. An object A in a category $\mathbb{C}$ is *finitely presented* [1] if and only if the hom-functor $\mathbb{C}(A, _) : \mathbb{C} \to \mathbb{S}et$ preserves directed colimits. For example a set is finitely presented (as an object of $\mathbb{S}et$) if and only if it is finite.

[1] Strictly speaking, this is only a hyper-category living in a higher set-theoretic universe.

2.2 General Concepts

The theory of "institutions" [6] is a categorical abstract model theory which formalises the intuitive notion of logical system, including syntax, semantics, and the satisfaction between them.

The concept of institution arose within computing science (algebraic specification) in response to the population explosion among logics in use there, with the ambition of doing as much as possible at a level of abstraction independent of commitment to any particular logic [6]. Besides its extensive use in specification theory (it has become the most fundamental mathematical structure in algebraic specification theory), there have been several substantial developments towards an "institution-independent" (abstract) model theory [15,4,7,14]. A textbook dedicated to this topic is under preparation [3]. Apart from reformulation of standard concepts and results in a very general setting, thus applicable to many logical systems, institution-independent model theory has already produced a number of new significant results in classical model theory [4,7].

Definition 1. *An* institution $\mathcal{I} = (\mathbb{S}ign^{\mathcal{I}}, \mathsf{Sen}^{\mathcal{I}}, \mathsf{Mod}^{\mathcal{I}}, \models^{\mathcal{I}})$ *consists of*

1. *a category* $\mathbb{S}ign^{\mathcal{I}}$*, whose objects are called* signatures*,*
2. *a functor* $\mathsf{Sen}^{\mathcal{I}} : \mathbb{S}ign^{\mathcal{I}} \to \mathbb{S}et$*, giving for each signature a set whose elements are called* sentences *over that signature,*
3. *a functor* $\mathsf{Mod}^{\mathcal{I}} : (\mathbb{S}ign^{\mathcal{I}})^{\mathrm{op}} \to \mathbb{CAT}$ *giving for each signature* Σ *a category whose objects are called* Σ-models*, and whose arrows are called* Σ-*(*model*)* morphisms*, and*
4. *a relation* $\models^{\mathcal{I}}_{\Sigma} \subseteq |\mathsf{Mod}^{\mathcal{I}}(\Sigma)| \times \mathsf{Sen}^{\mathcal{I}}(\Sigma)$ *for each* $\Sigma \in |\mathbb{S}ign^{\mathcal{I}}|$*, called* Σ-satisfaction*,*

such that for each morphism $\varphi : \Sigma \to \Sigma'$ *in* $\mathbb{S}ign^{\mathcal{I}}$*, the* satisfaction condition

$$M' \models^{\mathcal{I}}_{\Sigma'} \mathsf{Sen}^{\mathcal{I}}(\varphi)(\rho) \quad \textit{iff} \quad \mathsf{Mod}^{\mathcal{I}}(\varphi)(M') \models^{\mathcal{I}}_{\Sigma} \rho$$

holds for each $M' \in |\mathsf{Mod}^{\mathcal{I}}(\Sigma')|$ *and* $\rho \in \mathsf{Sen}^{\mathcal{I}}(\Sigma)$*. We denote the* reduct *functor* $\mathsf{Mod}^{\mathcal{I}}(\varphi)$ *by* $_\restriction_{\varphi}$ *and the sentence translation* $\mathsf{Sen}^{\mathcal{I}}(\varphi)$ *by* $\varphi(_)$*. When* $M = M'\restriction_{\varphi}$ *we say that* M *is a* φ-reduct of M'*, and that* M' *is a* φ-expansion of M*.*

Some of the proof theoretic aspects of a logic can be captured by the definition of an entailment system.

Definition 2. *A sentence system* $(\mathbb{S}ign, \mathsf{Sen})$ *consists of a category of signatures and a sentence functor* $\mathsf{Sen} : \mathbb{S}ign \to \mathbb{S}et$*.*

Definition 3. *[12] An entailment system* $\vdash$ *for a sentence system* $(\mathbb{S}ign, \mathsf{Sen})$ *is a family of relations* $\{\vdash_{\Sigma}\}_{\Sigma \in \mathbb{S}ign}$ *between sets of sentences* $\vdash_{\Sigma} \subseteq \mathcal{P}(\mathsf{Sen}(\Sigma)) \times \mathcal{P}(\mathsf{Sen}(\Sigma))$ *for all* $\Sigma \in \mathbb{S}ign$ *such that:*

anti-monotonicity. $E_0 \subseteq E_1$ *implies* $E_1 \vdash E_0$
transitivity. $E_0 \vdash E_1$ *and* $E_1 \vdash E_2$ *implies* $E_0 \vdash E_2$*.*

unions. *$E_0 \vdash E_1$ and $E_0 \vdash E_2$ implies $E_0 \vdash E_1 \cup E_2$.*
translation. *$E \vdash E' \Rightarrow \phi(E) \vdash \phi(E')$ for all $\phi : \Sigma \to \Sigma'$*

Definition 4. *An entailment system $\vdash$ is* compact *if whenever $E \vdash \Gamma$ for a finite set of sentences Γ there exists a finite subset E_0 of E such that $E_0 \vdash \Gamma$.*

Example 1. Let **FOL** be the institution of *many sorted first order logic with equality.* Its signatures (S, F, P) consist of a set of sort symbols S, a set F of function symbols, and a set P of relation symbols. Each function or relation symbol comes with a string of argument sorts, called its *arity*, and for functions symbols, a result sort. We assume that each sort has at least one term (i.e. the signatures are sensible [10]).

Simple signature morphisms map the three components in a compatible way. In order to treat substitutions as signature morphisms we will work in this paper with a more powerful version of signature morphisms. *A generalised* **FOL**-*morphism* between (S, F, P) and (S', F', P') is a simple signature morphism between (S, F, P) and $(S', F' + T_{F'}, P')$, i.e. constants can be mapped to terms.

Models M are first order structures interpreting each sort symbol s as a set M_s, each function symbol σ as a total function M_σ from the product of the interpretations of the argument sorts to the interpretation of the result sort, and each relation symbol π as a subset M_π of the product of the interpretations of the argument sorts. A model morphism $h : M \to N$ is a family of functions $\{h_s : M_s \to N_s\}_{s \in S}$ indexed by the sets of sorts of the signature such that: $h_s(M_\sigma(m)) = N_\sigma(h_w(m))$ for each $\sigma : w \to s$ and each $m \in M_w$ and $h_w(M_\pi) \subseteq N_\pi$ for each $\pi : w$.

Note that each sort interpretation M_s is non-empty since it contains the interpretation of at least one term.

Sentences are the usual first order sentences built from equational and relational atoms by iterative application of logical connectives (conjunction, negation and false), and existential quantifiers over a finite number of variables[2].

Sentence translations rename the sort, function, and relation symbols. For each signature morphism φ, the reduct $M'{\restriction_\varphi}$ of a model M' is defined by $(M'{\restriction_\varphi})_x = M'_{\varphi(x)}$ for each sort, function, or relation symbol x from the domain signature of φ. The satisfaction of sentences by models is defined inductively on the structure of the sentences.

Remark 1. Notice that when we refer to "atomic sentences" we mean "ground atomic sentences". This distinction is not necessary in an institutional presentation as we do not have atomic sentences with free variables. Sentences with variables are quantified and hence not atomic.

3 Institution Independent Techniques

The concept of institution tries to capture the essence of "being a logic" and reasoning at institutional level is an attempt to reason generically about properties

[2] We will identify sentences modulo renaming of the variables. A more explicit description of the identifications required will be given in Example 3.2.

of logics. Of course, there cannot be many results that are derivable only from the definition of an institution, therefore, to obtain nontrivial theorems about a class of logics we must define abstractly the properties of these logics. An enumeration of these abstract properties used in this paper together with explanations regarding their counterparts in concrete examples is given below.

The main result is obtained under the assumption that signatures do not have "void sorts". We express this requirement abstractly by the following definition:

Definition 5. *A signature morphism* $\phi : \Sigma \to \Sigma'$ *is* non-void *if there exists* $\phi' : \Sigma' \to \Sigma$ *such that* $\phi; \phi' = 1_\Sigma$*.*

Example 1 (continued). In **FOL** with generalised signature morphisms the non-void quantification translates into accepting only signatures which are sensible [10].

As implied by the choice of the signatures morphisms in the example we plan to treat substitutions as morphisms in a comma category of signature morphisms.

Definition 6. *Consider two signature morphisms* $\phi_0 : \Sigma \to \Sigma_0$ *and* $\phi_1 : \Sigma \to \Sigma_1$*. A signature morphism* $\phi : \Sigma_0 \to \Sigma_1$ *such that* $\phi_0; \phi = \phi_1$ *is called a* substitution morphism *between* ϕ_0 *and* ϕ_1*.*

3.1 Sentences

Definition 7. *A set of sentence* $E \subseteq \mathsf{Sen}(\Sigma)$ *is called* basic *if there exists a model* $M_E \in \mathsf{Mod}(\Sigma)$*, called a* basic model *of* E*, such that for all* $M \in \mathsf{Mod}(\Sigma)$*:*

$$M \models E \Leftrightarrow \exists h : M_E \to M$$

Example 1 (continued). In typical institutions, the simplest sentences are basic sentences, i.e. are preserved by model homomorphisms. They constitute the bricks from which complex sentences are constructed using Boolean connectives and quantification.

In **FOL** all sets of atomic sentences are basic. The basic model M_E for a set E of atomic sentences, is the initial model for E. This is constructed as the quotient of the initial algebra by the congruence defined by the equational atoms from E, and the interpretation of predicates contains only the congruence classes of the terms appearing in the relational atoms of E.

Definition 8. *In any institution a* Σ*-sentence* ρ *is* finitary *if and only if it can be written as* $\phi(\rho_f)$ *where* $\phi : \Sigma_f \to \Sigma$ *is a signature morphism such that* Σ_f *is a finitely presented signature and* ρ_f *is a* Σ_f *sentence.*

An institution has finitary sentences *when all its sentences are finitary.*

Example 1 (continued). This condition usually means in typical institutions that the sentences contain only a finite number of symbols.

Proposition 1. *Let* Σ_ω *be the inductive colimit of a chain of signature morphisms* $\{\phi_{n,n+1} : \Sigma_n \to \Sigma_{n+1}\}_{n<\omega}$*. For each* $\rho_\omega \in \mathsf{Sen}(\Sigma_\omega)$ *there exists* n *and* $\rho_n \in \mathsf{Sen}(\Sigma_n)$ *such that* $\phi_n(\rho_n) = \rho_\omega$*.*

3.2 Internal Logic

The logical connectives and quantification can be defined generically in any institution.

Definition 9. *A sentence $\rho \in \mathsf{Sen}(\Sigma)$ is called a* semantic conjunction *of two sentences $\rho_0, \rho_1 \in \mathsf{Sen}(\Sigma)$ if $M \models \rho$ iff $M \models \rho_0$ and $M \models \rho_1$. A sentence $\rho \in \mathsf{Sen}(\Sigma)$ is called a* semantic negation *of a sentence $\rho_0 \in \mathsf{Sen}(\Sigma)$ if $M \models \rho$ iff $M \not\models \rho_0$. A sentence $\rho \in \mathsf{Sen}(\Sigma)$ is called a* semantic false *if there is no model that satisfies it. A sentence $\rho \in \mathsf{Sen}(\Sigma)$ is called a* semantic existential quantification *of a sentence $\rho' \in \mathsf{Sen}(\Sigma')$ over the signature morphism $\chi : \Sigma \to \Sigma'$ if $M \models \rho$ iff there exists a χ-expansion M' of M, i.e. $M'\restriction_\chi = M$, that satisfies ρ'.*

Definition 10. *[13,5] A sentence system $(\mathbb{S}ign, \mathsf{Sen})$ is* equipped with conjunctions, negation and false *if it has three natural transformations $\wedge : \mathsf{Sen} \times \mathsf{Sen} \to \mathsf{Sen}$, $\neg : \mathsf{Sen} \to \mathsf{Sen}$, $\bot : 1 \to \mathsf{Sen}$.*

A sentence system $(\mathbb{S}ign, \mathsf{Sen})$ is equipped with pre-quantifiers *if there is a wide subcategory of signature morphisms $\mathcal{D}$ and a functor $\mathcal{Q} : \mathcal{D} \to \mathbb{S}et^{\mathrm{op}}$, that works the same as Sen on objects, such that for each pushout of signature morphisms:*

$$\begin{array}{ccc} \Sigma & \xrightarrow{\phi} & \Sigma_0 \\ {\scriptstyle\chi}\downarrow & & \downarrow{\scriptstyle\chi_0} \\ \Sigma' & \xrightarrow[\phi']{} & \Sigma'_0 \end{array}$$

with $\chi \in \mathcal{D}$ the following square commutes and is a weak pullback:

$$\begin{array}{ccc} \mathsf{Sen}(\Sigma) & \xrightarrow{\phi} & \mathsf{Sen}(\Sigma_0) \\ {\scriptstyle\mathcal{Q}(\chi)}\uparrow & & \uparrow{\scriptstyle\mathcal{Q}(\chi_0)} \\ \mathsf{Sen}(\Sigma') & \xrightarrow[\phi']{} & \mathsf{Sen}(\Sigma'_0) \end{array}$$

An institution $(\mathbb{S}ign, \mathsf{Sen}.\mathsf{Mod}, \models)$ has explicit conjunctions, negations, false and existential quantification *if the underlying sentence system is equipped with the enumerated constructors and the sentences produced by these have the semantical meaning defined above. For example $\mathcal{Q}(\chi)(\rho')$ must be a semantic existential quantification of ρ' over χ for every $\chi : \Sigma \to \Sigma'$ from $\mathcal{D}$ and $\rho' \in \mathsf{Sen}(\Sigma')$. We will write $\exists\chi.\rho'$ for $\mathcal{Q}(\chi)(\rho')$ in an institution with explicit existential quantifiers. We also denote by $Quant_\Sigma$ the set of existential quantified sentences in $\mathsf{Sen}(\Sigma)$, i.e. $\rho \in Quant_\Sigma$ if there exists $\chi : \Sigma \to \Sigma'$ and $\rho' \in \mathsf{Sen}(\Sigma')$ such that $\rho = \mathcal{Q}(\chi)(\rho')$.*

Based on the available sentence constructors we can introduce also disjunction, implication and universal quantification with the usual definitions.

Remark 2. Some conditions on $\mathcal{D}$ will be summarised below in order to assure that quantification is done in a homogenous way. First of all we will require that

colimits done with morphisms from $\mathcal{D}$ exist. Secondly, we want that the class of quantifying morphisms is essentially the same when we change the signature; this property will be stated in the form of pushout (co-)completeness. Finally, notice that if two signatures morphisms from D are isomorphic in the comma category $i : \chi_0 \to \chi_1$ then $\mathcal{Q}(\chi_0)(\rho_0) = \mathcal{Q}(\chi_1)(i(\rho_0))$. We would also like the converse of this property to hold, i.e. when we have two representations of the same sentence then these should be isomorphic.

Definition 11. *Let $\mathcal{D}$ be a wide subcategory of $\mathbb{S}ign$. We say that $\mathcal{D}$ is a* proper quantification class of signature morphisms *if it has the following properties:*

- *there exists a wide subcategory $\overline{\mathbb{S}ign}$ of $\mathbb{S}ign$, that includes $\mathcal{D}$ such that $\overline{\mathbb{S}ign}$ has colimits.*
- *$\mathcal{D}$ is pushout complete, i.e. for all $\chi : \Sigma \to \Sigma_0$ in $\mathcal{D}$ and all $\phi : \Sigma \to \Sigma'$ in $\overline{\mathbb{S}ign}$ there exists $\chi' : \Sigma' \to \Sigma'_0$ in $\mathcal{D}$ and $\phi_0 : \Sigma_0 \to \Sigma'_0$ such that the following diagram is a pushout:*

$$\begin{array}{ccc} \Sigma & \xrightarrow{\phi} & \Sigma' \\ \chi \downarrow & & \vdots\, \chi' \\ \Sigma_0 & \xrightarrow[\phi_0]{\cdots} & \Sigma'_0 \end{array}$$

- *$\mathcal{D}$ is pushout co-complete, i.e. for all $\chi' : \Sigma' \to \Sigma'_0$ in $\mathcal{D}$ and all $\phi : \Sigma \to \Sigma'$ in $\overline{\mathbb{S}ign}$ there exists $\chi : \Sigma \to \Sigma_0$ in $\mathcal{D}$ and $\phi_0 : \Sigma_0 \to \Sigma'_0$ such that the following diagram is a pushout:*

$$\begin{array}{ccc} \Sigma & \xrightarrow{\phi} & \Sigma' \\ \chi\, \vdots & & \downarrow \chi' \\ \Sigma_0 & \xrightarrow[\phi_0]{\cdots} & \Sigma'_0 \end{array}$$

- *if $\mathcal{Q}(\chi_0)(\rho_0) = \mathcal{Q}(\chi_1)(\rho_1)$ then there exists an isomorphism $i : \Sigma_0 \to \Sigma_1$ such that $\chi_0; i = \chi_1$ and $i(\rho_0) = \rho_1$*

Example 1 (continued). Notice that **FOL** with generalised signature morphisms does not have all colimits. Therefore, we take $\overline{\mathbb{S}ign}$ to be the class of simple signature morphisms; this class is closed under the construction of colimits.

The class $\mathcal{D}$ will contain the signature morphisms that add a finite number of constants. In order to obtain all the properties for a proper class we will consider that existential quantified sentences are identified modulo renaming of variables thus enforcing the last condition from Definition 11. We also require that $\exists\chi_0; \chi_1.\rho_1 = \exists\chi_0.\exists\chi_1.\rho_1$ in order for Q, defined as $Q(\chi)(\rho') = \exists\chi.\rho'$, to be a functor.

Remark 3. Consider a sentence system with pre-quantifiers ($\mathbb{S}ign, \mathsf{Sen}, \mathcal{D}$, $\mathcal{Q}$), such that $\mathcal{D}$ is a proper quantification class of signature morphisms. Then for all $\phi : \Sigma \to \Sigma_0$, $\rho \in \mathsf{Sen}(\Sigma)$ and $\rho_0 \in \mathsf{Sen}(\Sigma_0)$ such that $\phi(\rho) = \rho_0$ we have that $\rho \in \mathrm{Quant}_\Sigma$ iff $\rho_0 \in \mathrm{Quant}_{\Sigma_0}$.

Proof: The "only if" implication follows from completeness of $\mathcal{D}$ and from the commutativity condition for $\mathcal{Q}$ while the "if" implication follows from co-completeness of $\mathcal{D}$ and the pullback condition for $\mathcal{Q}$. ⊣

Notice that the general method of talking about quantification introduced above is powerful enough to express second order quantification. To restrict some of this power, and to specialise the main result to "first order" logics it is necessary to characterise abstractly the notion of "first order" quantification.

Definition 12. *A signature morphism* $\chi : \Sigma \to \Sigma'$ covers $M \in \mathsf{Mod}(\Sigma)$ *if for each* $M' \in \mathsf{Mod}(\Sigma')$ *such that* $M'\restriction_\chi = M$ *there exists a signature morphism* $\chi' : \Sigma' \to \Sigma$ *such that* $\chi;\chi' = 1_\Sigma$ *and* $M\restriction_{\chi'} = M'$.

Definition 13. *Consider a* $\mathbb{S}ign$*-indexed family of sets of sentences, i.e.* $\mathsf{Sen}^{\mathbf{base}}(\Sigma) \subseteq \mathsf{Sen}(\Sigma)$*, such that any set of sentences* E *from* $Sen^{\mathbf{base}}(\Sigma)$ *is basic. We say that* $\mathcal{D}$ has $\mathsf{Sen}^{\mathbf{base}}$ basic coverage *if for all* $\chi : \Sigma \to \Sigma'$ *in* $\mathcal{D}$ *and* $E \subseteq \mathsf{Sen}^{\mathbf{base}}(\Sigma)$*,* χ *covers the basic model* M_E*.*

Remark 4. This condition will be the only one in the list of assumptions for the completeness theorem that fails for second order logic. It is useful to have a closer look at its meaning.

Having in mind that basic models are typically reachable, i.e. each element from the model is the denotation of a ground term, the basic coverage property can be restated as: $\mathcal{D}$-extensions of term models have a syntactical flavour, i.e. correspond to signature morphisms.

This is true for extensions that interpret constants, as there is a generalised signature morphism for each interpretation in a term model. However, for extensions that interpret predicates or function symbols this correspondence fails.

4 Generated Institutions and Entailment Systems

Definition 14. *Let* $\mathcal{I} = (\mathbb{S}ign, \mathsf{Sen}.\mathsf{Mod}, \models, \mathcal{D}, \mathcal{Q}, \wedge, \neg)$ *an institution with explicit existential quantification, conjunctions, negations and false and* $\mathsf{Sen}^{\mathbf{base}}$ *a* $\mathbb{S}ign$*-indexed family of sets of sentences.*

$\mathcal{I}$ *is* $\mathsf{Sen}^{\mathbf{base}}$-generated *if any sentence in* $\mathsf{Sen}(\Sigma)$ *can be obtained from sentences in* $\mathsf{Sen}^{\mathbf{base}}(\Sigma)$ *by applying conjuction, negation, false and existential quantification.*

Definition 15. *Let* $\mathcal{I} = (\mathbb{S}ign, \mathsf{Sen}, \mathsf{Mod}, \models, \vdash)$ *be a* $\mathsf{Sen}^{\mathbf{base}}$ *generated institution with entailment. We say that* $\vdash$ *is* good w.r.t. $\mathsf{Sen}^{\mathbf{base}}$ *if it obeys the following conditions:*

Rule 1 $\models^{\mathbf{base}} \subseteq \vdash$
Rule 2 $\{\rho, \rho'\} \dashv\vdash \rho \wedge \rho'$
Rule 3 $E \cup \{\rho\} \vdash \bot \Rightarrow E \vdash \neg\rho$
Rule 4 $\bot \vdash \rho$ *for all* ρ
Rule 5 $\rho \wedge \neg\rho \vdash \bot$ *for all* ρ

Rule 6 $\neg\neg\rho \vdash \rho$ *for all* ρ
Rule 7 $\chi(E) \vdash \neg\rho' \Rightarrow E \vdash \neg\exists\chi.\rho'$
Rule 8 $E \vdash \psi(\rho') \Rightarrow E \vdash \exists\chi.\rho'$ *for all substitutions* $\psi : \chi \to 1_\Sigma$

Proposition 2. *The following properties hold for a good entailment system* $\vdash$*:*

- $\chi(E) \vdash \rho' \Rightarrow E \vdash \forall\chi.\rho'$
- $E \vdash \forall\chi.\rho' \Rightarrow E \vdash \psi(\rho')$ *for all* $\psi : \chi \to 1_\Sigma$
- $\phi(E) \vdash \perp \Rightarrow E \vdash \perp$ *for all non-void* $\phi : \Sigma \to \Sigma'$
- $\forall\chi.(\rho'_1 \wedge \rho'_2) \vdash \forall\chi.\rho'_1 \wedge \forall\chi.\rho'_2$
- $\forall\chi.\chi(\rho) \vdash \rho$ *for all non-void* $\chi \in \mathcal{D}$
- $\emptyset \vdash \exists\chi.(\chi(\exists\chi.\rho') \to \rho')$ *for all non-void* $\chi \in \mathcal{D}$

The conditions enumerated above give a sufficient framework in which to prove the inclusion $\models \subseteq \vdash$, and this will be the subject of the next sections.

5 Henkin Theory

This section illustrates Henkin's construction that takes any consistent theory and extends it to a maximal theory in a language big enough to contain a witness constant for every existential truth entailed by the original theory.

Let us fix the framework for this section:

Framework 1 $\mathcal{I}$ *is a* $\mathsf{Sen}^{\mathbf{base}}$*-generated institution with finitary sentences,* $\vdash$ *is a compact entailment system that is good w.r.t.* $\mathsf{Sen}^{\mathbf{base}}$*, and* $\mathcal{D}$ *is a proper quantification class of non-void signature morphisms.*

Definition 16. *A set of sentences* E *is* consistent *if* $E \nvdash \perp$*. A set of sentences* E *is* maximal *if* $E \nvdash \rho \Leftrightarrow E \vdash \neg\rho$*.*

A set of sentences E *is a* Henkin theory *if for all* $\chi : \Sigma \to \Sigma'$ *and* $\rho' \in \mathsf{Sen}(\Sigma')$

$$E \vdash \exists\chi.\rho' \Leftrightarrow \text{ there exists } \chi' : \Sigma' \to \Sigma \text{ s.t. } \chi;\chi' = 1_\Sigma \text{ and } E \vdash \chi'(\rho')$$

Remark 5. Note that the implication from right to left in the definition of a Henkin theory is equivalent to Rule 8.

Definition 17. *For each signature* Σ *we define the* one-step extension signature Σ° *and the morphism* $\phi_\Sigma : \Sigma \to \Sigma^\circ$ *as follows:*

Assume $\{\langle\chi_\alpha, \rho_\alpha\rangle\}_{\alpha < card(Quant_\Sigma)}$ *is an ordered choice of representants for the quantified sentences of* $Quant_\Sigma$ *and let* Σ° *be the colimit of the diagram below:*

$$\begin{array}{ccccc} & & \Sigma_\alpha & & \\ & \nearrow^{\chi_\alpha} & \vdots & \searrow^{c_\alpha} & \\ \Sigma & & \xrightarrow{\phi_\Sigma} & & \Sigma^\circ \\ & \searrow_{\chi_\beta} & \vdots & \nearrow_{c_\beta} & \\ & & \Sigma_\beta & & \end{array}$$

We will denote by Γ_Σ *the set of* Σ°*-sentences* $\{c_\alpha(\rho_\alpha) \mid \emptyset \vdash \phi_\Sigma(\exists\chi_\alpha.\rho_\alpha)$ *and* $\alpha < card(Quant_\Sigma)\}$*.*

Remark 6. The colimit present in the above definition exists because we can make colimits using morphisms of a proper class of quantification.

Lemma 1. *The extension signature morphism $\phi_\Sigma : \Sigma \to \Sigma^\circ$ is non-void.*

Lemma 2. *$\phi_\Sigma(E) \cup \Gamma_\Sigma$ is consistent whenever E is consistent.*

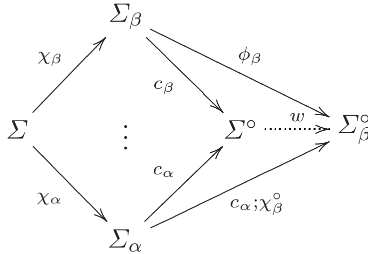
PROOF: First of all let us point that because ϕ_Σ is non-void, using Proposition 2 we obtain that $\phi_\Sigma(E)$ is consistent.

Now, assume that $\phi_\Sigma(E)\cup\Gamma_\Sigma$ is inconsistent, i.e. $\phi_\Sigma(E)\cup\Gamma_\Sigma \vdash\perp$. Because the entailment system is compact there exists a finite set $E_0 \subseteq \phi_\Sigma(E)\cup\Gamma_\Sigma$ such that $E_0 \vdash\perp$. If we consider $\Gamma_0 := E_0\cap\Gamma_\Sigma$ we have that $\phi_\Sigma(E)\cup\Gamma_0 \vdash\perp$. Because $\phi_\Sigma(E)$ is consistent we get that there exists β such that $E' := \phi_\Sigma(E)\cup\{c_\alpha(\rho_\alpha) \mid \alpha < \beta\}$, $E' \nvdash\perp$ and $E' \cup \{c_\beta(\rho_\beta)\} \vdash\perp$.

Let $\langle\phi_\beta,\, \chi^\circ_\beta\rangle$ be the pushout of $\langle\chi_\beta,\, \phi_\Sigma\rangle$ such that $\chi^\circ_\beta \in \mathcal{D}_{\Sigma^\circ}$:

$$\begin{array}{ccc} \Sigma & \xrightarrow{\phi_\Sigma} & \Sigma^\circ \\ {\scriptstyle\chi_\beta}\downarrow & & \downarrow{\scriptstyle\chi^\circ_\beta} \\ \Sigma_\beta & \xrightarrow[\phi_\beta]{} & \Sigma^\circ_\beta \end{array}$$

And also remark that $\langle\phi_\beta,\, \{c_\alpha;\chi^\circ_\beta|\alpha \neq \beta\}\rangle$ is a cocone for $\langle\chi_\beta,\, \{\chi_\alpha|\alpha \neq \beta\}\rangle$. (Because $\chi_\alpha; c_\alpha; \chi^\circ_\beta = \phi_\Sigma; \chi^\circ_\beta = \chi_\beta; \phi_\beta$ for all $\alpha \neq \beta$)



From the universality property of the colimit we get that there exists $w : \Sigma^\circ \to \Sigma^\circ_\beta$ such that $c_\beta; w = \phi_\beta$ and $c_\alpha; w = c_\alpha; \chi^\circ_\beta$ for $\alpha \neq \beta$.

Now remember that $E' \cup \{c_\beta(\rho_\beta)\} \vdash\perp$. Using rule 3 we can deduce that $E' \vdash \neg c_\beta(\rho_\beta)$. Using the translation rule we obtain $w(E') \vdash w(\neg c_\beta(\rho_\beta))$. Because $E' = \phi_\Sigma(E) \cup \{c_\alpha(\rho_\alpha) \mid \alpha < \beta\}$ we can easily see that $w(E') = \chi^\circ_\beta(E')$ (using the fact that $c_\alpha; w = c_\alpha; \chi^\circ_\beta$). This leads us to $\chi^\circ_\beta(E') \vdash \neg\phi_\beta(\rho_\beta)$ which by the use of rule 7 gives us $E' \vdash \neg\exists\chi^\circ_\beta.\phi_\beta(\rho_\beta)$. But $\phi_\Sigma(\exists\chi_\beta.\rho_\beta) = \exists\chi^\circ_\beta.\phi_\beta(\rho_\beta)$ and we obtain $E' \vdash \neg\phi_\Sigma(\exists\chi_\beta.\rho_\beta)$ and hence a contradiction, because $\phi_\Sigma(\exists\chi_\beta.\rho_\beta)$ is entailed by the empty set. $\dashv$

Definition 18. *Iterating this process we can define the* full extension *of Σ:*

base $\Sigma_0 := \Sigma$ *and* $\Gamma_0 = \emptyset$

step *we make the construction from Definition 17 for the signature* Σ_n *obtaining* $\phi_{\Sigma_n} : \Sigma_n \to \Sigma_n^\circ$ *and we define* $\phi_{n,n+1} := \phi_{\Sigma_n}$, $\Sigma_{n+1} := \Sigma_n^\circ$, $\Gamma_{n+1} := \Gamma_{\Sigma_n}$.

The extension theory $(\Sigma_\omega, \Gamma_\omega)$ *will be the colimit of the chain* $\{\phi_{n,n+1}\}_{n<\omega}$.

Lemma 3. *Consider a chain of consistent theories such that each morphism* $\phi_{n,n+1}$ *is non-void. Then its inductive colimit is consistent.*

Proposition 3. $\phi(E) \cup \Gamma_\omega$ *is consistent whenever* E *is consistent.*

Lemma 4. *Let* $\rho_\omega \in Quant_{\Sigma_\omega}$ *such that* $\emptyset \vdash \rho_\omega$. *Then for every pair* $\langle \chi, \rho'_\omega \rangle$ *with* $\rho_\omega = \exists\chi.\rho'_\omega$ *there exists* $\chi' : \Sigma'_\omega \to \Sigma_\omega$ *such that* $\chi;\chi' = 1_{\Sigma_\omega}$ *and* $\chi'(\rho'_\omega) \in \Gamma_\omega$.

Proof: First let us point that it is sufficient to prove the conclusion for one pair because two signature morphisms that are used to obtain the same quantified sentence must be isomorphic.

Assume $\emptyset \vdash \rho_\omega$. Because $\mathcal{I}$ has finitary sentences we get that there exists n and $\rho_n \in \mathsf{Sen}(\Sigma_n)$ such that $\phi_n(\rho_n) = \rho_\omega$.

From Remark 3 we can reason that ρ_n is a quantified sentence in $\mathsf{Sen}(\Sigma_n)$. Let α be the index of ρ_n in the enumeration of $\mathrm{Quant}_{\Sigma_n}$ and $\langle \chi_\alpha, \rho'_\alpha \rangle$ be the chosen representant pair for this sentence with $\chi_\alpha : \Sigma_n \to \Sigma'_n$ and $\rho'_\alpha \in \mathsf{Sen}(\Sigma'_n)$.

Consider the following pushout:

$$\begin{array}{ccc} \Sigma_n & \xrightarrow{\phi_n} & \Sigma_\omega \\ {\scriptstyle \chi_\alpha}\downarrow & & \downarrow{\scriptstyle \chi} \\ \Sigma'_n & \xrightarrow[\phi'_n]{} & \Sigma'_\omega \end{array}$$

We will prove the conclusion for the pair $\langle \chi, \rho'_\omega \rangle$ where $\rho'_\omega = \phi'_n(\rho'_\alpha)$.

From the universality property of the pushout we get that there exists $\chi' : \Sigma'_\omega \to \Sigma_\omega$ such that $\phi'_n;\chi' = c_\alpha;\phi_{n+1}$ and $\chi;\chi' = 1_{\Sigma_\omega}$:

$$\begin{array}{ccccc} \Sigma_n & \xrightarrow{\phi_n} & \Sigma_\omega & & \\ {\scriptstyle \chi_\alpha}\downarrow & & \downarrow{\scriptstyle \chi} & \searrow{\scriptstyle 1_{\Sigma_\omega}} & \\ \Sigma'_n & \xrightarrow[\phi'_n]{} & \Sigma'_\omega & & \\ & \searrow{\scriptstyle c_\alpha;\phi_{n+1}} & & \searrow{\scriptstyle \chi'} & \\ & & & & \Sigma_\omega \end{array}$$

Because $\emptyset \vdash \rho_\omega$ and $\rho_\omega = \phi_n(\rho_n) = \phi_n(\exists\chi_\alpha.\rho'_\alpha)$ we get that $\emptyset \vdash \phi_n(\exists\chi_\alpha.\rho'_\alpha)$. Furthermore using the fact that ϕ_{n+1} is non-void we get that $\emptyset \vdash \phi_{n,n+1}(\exists\chi_\alpha.\rho'_\alpha)$. This means that $c_\alpha(\rho'_\alpha) \in \Gamma_{n+1}$ which using $\phi'_n;\chi' = c_\alpha;\phi_{n+1}$ leads us to $\chi'(\phi'_n(\rho'_\alpha)) \in \Gamma_\omega$ or equivalently $\chi'(\rho'_\omega) \in \Gamma_\omega$. $\dashv$

Proposition 4. *Every extension* E *of* Γ_ω *is a Henkin theory.*

PROOF: Assume $\Gamma_\omega \subseteq E$ and $E \vdash \exists\chi.\rho'$.

Note that $\emptyset \vdash \exists\chi.(\chi(\exists\chi.\rho') \to \rho')$ (Proposition 2). Using lemma 4 we get that there exists $\chi' : \Sigma_0^\circ \to \Sigma^\circ$ such that $\chi;\chi' = 1_\Sigma$ and $\chi'(\chi(\exists\chi.\rho') \to \rho') \in \Gamma_\Sigma$. This leads us to $E \vdash \chi'(\chi(\exists\chi.\rho') \to \rho')$ and furthermore to $E \vdash (\exists\chi.\rho') \to \chi'(\rho')$. In conclusion, $E \vdash \chi'(\rho')$. ⊣

Proposition 5. *Every consistent theory can be extended to a maximal consistent theory.*

Theorem 1. *Every consistent theory can be extended to a maximal Henkin theory.*

Proof. To obtain a maximal Henkin theory just consider the maximal theory that contains $\phi(E) \cup \Gamma_\omega$. ⊣

6 Every Henkin Theory Has a Model

Framework 2 *I is a $\mathsf{Sen}^{\mathbf{base}}$ generated institution, all the sets of sentences in $\mathsf{Sen}^{\mathbf{base}}(\Sigma)$ are basic. $\mathcal{D}$ is a class of signature morphisms that has $\mathsf{Sen}^{\mathbf{base}}$-basic coverage.*

Theorem 2. *Every maximal Henkin theory has a model.*

PROOF: Let E be a Henkin theory in $\mathsf{Sen}(\Sigma)$. We define the set $E_B := \{\rho \mid E \vdash \rho, \rho \in \mathsf{Sen}(\Sigma)\}$ and let M_{E_B} be the basic model of E_B. We show that:

$$M_{E_B} \models \rho \Leftrightarrow E \vdash \rho$$

for every $\rho \in \mathsf{Sen}(\Sigma)$.

We will prove this using induction over the structure of sentences:

ρ is a base sentence.

"$M_{E_B} \models \rho \Rightarrow E \vdash \rho$" Because ρ is basic and $M_{E_B} \models \rho$ there exists a morphism $h : M_\rho \to M_{E_B}$. For every $M \models E_B$ there exists a morphism $h_M : M_{E_B} \to M$. In conclusion, the morphism $h;h_M$ assures us that $M \models \rho$ whenever $M \models E_B$. Furthermore, $E_B \models \rho$ implies $E_B \vdash \rho$ and finally $E \vdash \rho$.

"$E \vdash \rho \Rightarrow M_{E_B} \models \rho$". $E \vdash \rho \Rightarrow \rho \in E_B \Rightarrow M_{E_B} \models \rho$

ρ is a negation $\rho := \neg\rho_0$

"$M_{E_B} \models \rho \Leftrightarrow E \vdash \rho$". $M_{E_B} \models \neg\rho_0 \Leftrightarrow M_{E_B} \not\models \rho_0 \Leftrightarrow E \nvdash \rho_0 \Leftrightarrow E \vdash \neg\rho_0$

ρ is a conjuction, $\rho := \rho_1 \wedge \rho_2$.

"$M_{E_B} \models \rho \Leftrightarrow E \vdash \rho$". $M_{E_B} \models \rho_1 \wedge \rho_2 \Leftrightarrow M_{E_B} \models \rho_1$ and $M_{E_B} \models \rho_2 \Leftrightarrow E \vdash \rho_1$ and $E \vdash \rho_2 \Leftrightarrow E \vdash \rho_1 \wedge \rho_2$

ρ is an existential question $\rho := \exists\chi.\rho'$. Let us first write the two equivalences that will help us in this proof:

$$M_{E_B} \models \exists\chi.\rho' \Leftrightarrow \big(\text{ there exists } M' \text{ such that } M'\restriction_\chi = M \text{ and } M' \models \rho'\big)$$

$$E \vdash \exists\chi.\rho' \Leftrightarrow \big(\text{ there exists } \chi' \text{ such that } \chi;\chi' = 1_\Sigma \text{ and } E \vdash \chi'(\rho')\big)$$

"$M_{E_B} \models \rho \Rightarrow E \vdash \rho$". Let $M'\restriction_\chi = M_{E_B}$ be such that $M' \models \rho'$. Because $\mathcal{D}$ has $\mathsf{Sen}^{\mathbf{base}}$-basic coverage we get that there exists a $\chi' : \Sigma' \to \Sigma$ such that

$\chi;\chi' = 1_\Sigma$ and $M_{E_B}\restriction_{\chi'} = M'$. From the satisfaction condition we get that $M_{E_B} \models \chi'(\rho')$ and finally using the induction hypothesis we can conclude that $E \vdash \chi'(\rho')$.

"$E \vdash \rho \Rightarrow M_{E_B} \models \rho$". Let $\chi' : \Sigma' \to \Sigma$ be a signature morphism such that $\chi;\chi' = 1_\Sigma$ and $E \vdash \chi'(\rho')$. Using the induction hypothesis we get that $M_{E_B} \models \chi'(\rho')$. We can define M' to be $M_{E_B}\restriction_{\chi'}$. From the satisfaction condition we get that $M' \models \rho'$ but we can easily see that M' is a χ-expansion of M(i.e. $M'\restriction_\chi = M$) therefore $M \models \exists\chi.\rho'$. ⊣

7 Completeness

For the main theorem's hypothesis we will enumerate all the conditions from the two frameworks defined above: Framework 1 and Framework 2:

Theorem 3. *Let $I = (\mathbb{S}ign, \mathsf{Sen}, \mathsf{Mod}, \models, \vdash)$ be a $\mathsf{Sen}^{\mathbf{base}}$-generated institution with entailment such that*

- *$\mathcal{I}$ has finitary sentences*
- *every set of sentences in $\mathsf{Sen}^{\mathbf{base}}(\Sigma)$ is basic*
- *$\vdash$ is compact and good w.r.t. $\mathsf{Sen}^{\mathbf{base}}$*

and $\mathcal{D}$ be a proper quantification class of non-void signature morphisms that has $\mathsf{Sen}^{\mathbf{base}}$-basic coverage. Then the entailment system $\vdash$ is complete.

Proof. Using Theorem 1 and Theorem 2 we get that every consistent theory is satisfiable, and hence the completeness. ⊣

8 Working Examples

Before starting to apply Theorem 3 to typical institutions it is necessary to comment slightly on the nature of the result that has just been proved.

In the following examples we will describe some complete rules also for the base sentences and will consider the minimal generated entailment systems that include these rules and those enumerated in Definition 15. Because the definition is inductive we assure ourselves that we are dealing with compact and effective computable entailment relations.

The main points in checking the conditions of Theorem 3 are: choosing the set of base sentences such that each subset is basic; enumerating the rules for base sentences; choosing the quantification class of morphisms $\mathcal{D}$; and checking that $\mathcal{D}$ covers the basic models.

8.1 First Order Logic

In **FOL** we take $\mathsf{Sen}^{\mathbf{base}}(S, F, P)$ to be the set of atomic sentences. As explained in section 3.1 every set of atomic sentences is basic. We take $\vdash^{\mathbf{base}}$ to be the entailment system generated by the following rules:

Reflexivity $\vdash^{\mathbf{base}} t = t$
Symmetry $t = t' \vdash^{\mathbf{base}} t' = t$
Transitivity $\{t = t', t' = t''\} \vdash^{\mathbf{base}} t = t''$
Compatibility with F $\{t_i = t'_i \mid i = \overline{1,n}\} \vdash^{\mathbf{base}} \sigma(t_1, \ldots, t_n) = \sigma(t'_1, \ldots, t'_n)$
Compatibility with P $\{t_i = t'_i \mid i = \overline{1,n}\} \cup \{\pi(t_1, \ldots, t_n)\} \vdash^{\mathbf{base}} \pi(t'_1, \ldots, t'_n)$

and $\vdash$ to be the minimum good entailment system that includes $\vdash^{\mathbf{base}}$. The proper quantification class of signature morphisms will contain the signature morphisms that add a finite number of constants.

The property of basic coverage is easy to establish cosidering that the basic models are reachable. Consider an extension M' of a basic model M_E. Each constant added by the signature morphism from $\mathcal{D}$ is interpreted by a reachable element; this map between constants and terms builds a generalised signature morphism as needed.

8.2 Partial Algebras

Example 2. The institution **PA** of partial algebra [2] is defined as follows. A *partial algebraic signature* (S, F) consists of a set S of sorts and a set F of *partial* operations. We assume that there is a distinguished constant on each sort $\perp_s : s$. Signature morphisms map the sorts and operations in a compatible way, preserving $\perp_s$; we also allow that constants can be mapped to terms.

A partial algebra is just like an ordinary algebra but interpreting the operations of F as partial rather than total functions; $\perp_s$ is always interpreted as undefined. A *partial algebra homomorphism* $h : A \to B$ is a family of (total) functions $\{h_s : A_s \to B_s\}_{s \in S}$ indexed by the set of sorts S of the signature such that $h_s(A_\sigma(a)) = B_\sigma(h_w(a))$ for each operation $\sigma : w \to s$ and each string of arguments $a \in A_w$ for which $A_\sigma(a)$ is defined.

We consider one kind of atoms: *existence equality* $t \stackrel{e}{=} t'$. The existence equality $t \stackrel{e}{=} t'$ holds when both terms are defined and are equal[3]. Also, we identify all atomic sentences that contain $\perp_s$ with a special sentence $\perp_{(S,F)}$ having the semantic value of false.

The sentences are formed from these atoms by logical connectives and quantification over variables.

We consider the base sentences in $\mathsf{Sen}^{\mathbf{base}}(S, F)$ to be the atomic existential equalities that do not contain $\perp_s$; and $\mathcal{D}$ to be the class of signature morphisms that add a finite number of constants.

Proposition 6. *Every set of sentences from* $\mathsf{Sen}^{\mathbf{base}}$ *is basic.*

PROOF: For a set of atomic sentences E we define S_E to be the set of subterms appearing in E. We define $T_F(S_E)$ to be the partial algebra that has the carrier S_E. The basic model M_E will be the quotient of this algebra by the partial congruence induced by the equalities from E. $\dashv$

[3] The definedness predicate and strong equality can be introduced as notations: $def(t)$ stands for $t \stackrel{e}{=} t$ and $t \stackrel{s}{=} t'$ stands for $(t \stackrel{e}{=} t') \vee (\neg def(t) \wedge \neg def(t'))$.

Proposition 7. *$\mathcal{D}$ has* $\mathsf{Sen}^{\mathbf{base}}$*-basic coverage.*

Proof: Let $M'\restriction_\chi = M_E$. For any added constant x we must find a mapping into terms. If M'_x is defined then the value of its interpretation is an isomorphism class (modulo the equations from E) of terms from S_E. We can map x to any of the terms from this isomorphism class. Otherwise, if M'_x is undefined we map x to $\bot_s$. $\dashv$

Now we present a set of rules capable of enumerating all semantic entailments between base sentences. We take $\vdash^{\mathbf{base}}$ to be the entailment system generated by the following rules:

Reflexivity $\vdash^{\mathbf{base}} t \stackrel{e}{=} t$
Symmetry $t \stackrel{e}{=} t' \vdash^{\mathbf{base}} t' \stackrel{e}{=} t$
Transitivity $\{t \stackrel{e}{=} t', t' \stackrel{e}{=} t''\} \vdash^{\mathbf{base}} t \stackrel{e}{=} t''$
Congruence $\{t_i \stackrel{e}{=} t'_i | i \in 1 \ldots n\} \cup \{def(\sigma(t_1, \ldots, t_n)), def(\sigma(t'_1, \ldots, t'_n))\} \vdash^{\mathbf{base}}$
$\sigma(t_1, \ldots, t_n) \stackrel{e}{=} \sigma(t'_1, \ldots, t'_n)$ for $\sigma \in F$
Subterm $def(\sigma(t_1, \ldots, t_n)) \vdash^{\mathbf{base}} \{def(t_i) | i \in 1 \ldots n\}$

The minimal good entailment system w.r.t. $\vdash^{\mathbf{base}}$ will be compact and complete by Theorem 3.

9 Conclusions and Future Work

The results that have been presented can be divided into two parts. The first part, the construction of a Henkin theory in Section 5, is a very general proof, done with few assumptions that don't require the "first order" flavour of a logic. The second part, in Section 6, proves the existence of a model for a Henkin theory using the property of basic coverage (Definition 13) which excludes the application of the completeness theorem to second order logics. Future investigations should look for other meaningful examples that fit the intuition of first-order quantification, like completeness w.r.t. Kripke semantics. The most important restriction of this presentation is the assumption of sensible signatures. This deficiency should be corrected in future work allowing the derivation of a completeness theorem for first order logics with void sorts.

References

1. Adámek, J., Rosický, J.: Locally Presentable and Accessible Categories. London Mathematical Society Lecture Notes, vol. 189. Cambridge University Press, Cambridge (1994)
2. Burmeister, P.: A Model Theoretic Oriented Approach to Partial Algebras. Akademie-Verlag (1986)
3. Diaconescu, R.: Institution-independent Model Theory (to appear)
4. Diaconescu, R.: An institution-independent proof of Craig Interpolation Theorem. Studia Logica 77(1), 59–79 (2004)
5. Diaconescu, R.: Institutions with proofs. Journal of Logic and Computation (2006)

6. Goguen, J., Burstall, R.: Institutions: Abstract model theory for specification and programming. Journal of the Association for Computing Machinery 39(1), 95–146 (1992)
7. Găină, D., Popescu, A.: An institution-independent proof of Robinson consistency theorem. Studia Logica (to appear)
8. Henkin, L.: The completeness of the first-order functional calculus. Journal of Symbolic Logic 14(3), 159–166 (1949)
9. Henkin, L.: The discovery of my completeness proofs. Bulletin of Symbolic Logic 2(2), 127–158 (1996)
10. Huet, G., Oppen, D.C.: Equations and rewrite rules: a survey. Formal Language Theory: Perspectives and Open Problems, 349–405 (1980)
11. MacLane, S.: Categories for the Working Mathematician. Springer, New York (1998)
12. Meseguer, J.: General logics. In: Logic Colloquium 87, pp. 275–329. North Holland, Amsterdam (1989)
13. Mossakowski, T., Goguen, J., Diaconescu, R., Tarlecki, A.: What is a logic? Logica Universalis (2005)
14. Petria, M., Diaconescu, R.: Abstract Beth definability in institutions. Journal of Symbolic Logic 71(3), 1002–1028 (2006)
15. Tarlecki, A.: Bits and pieces of the theory of institutions. In: Poigné, A., Pitt, D.H., Rydeheard, D.E., Abramsky, S. (eds.) Category Theory and Computer Programming. LNCS, vol. 240, Springer, Heidelberg (1986)

Coalgebraic Foundations of Linear Systems
(An Exercise in Stream Calculus)

J.J.M.M. Rutten

CWI and Vrije Universiteit Amsterdam

Abstract. Viewing discrete-time causal linear systems as (Mealy) coalgebras, we describe their semantics, minimization and realisation as universal constructions, based on the final coalgebras of streams and causal stream functions.

1 Introduction

Linear systems are a fundamental mathematical structure with applications in control theory, signal processing, and telecommunications. In computer science, they are given but little attention. However, linear systems provide a mathematical model for various types of networks, including signal flow graphs and linear sequential Boolean circuits (see, for instance, [Koh78, Lah98]). Such networks are highly relevant for the foundations of computing, being elementary and beautiful examples of the combined occurrence of memory and feedback.

In this paper, we give a coalgebraic account of the semantics of the following elementary type of linear system: a (state-based) discrete-time (strongly) causal linear system consists of a vector space V of states; vector spaces I and O of inputs and outputs; and linear maps $F : V \to V$, describing the system's dynamics, and $G : I \to V$ and $H : V \to O$, describing the system's input and output. We shall model such a system as a Mealy automaton (V, Φ), defined by

$$\Phi : V \to (O \times V)^I \qquad \Phi(v)(i) = \langle H(v),\, F(v) + G(i)\rangle$$

Such Mealy automata, or (I, O)-*systems* as we shall call them here, are coalgebras of the functor $\mathcal{F} : \mathit{Set} \to \mathit{Set}$ defined by $\mathcal{F}(S) = (O \times S)^I$. The choice to model linear systems as Mealy automata or, in other words, as coalgebras of this particular choice of functor $\mathcal{F}$, is motivated by the following observation: In [Rut06], it is shown that the final coalgebra of $\mathcal{F}$, which is to serve as our semantic universe, is (isomorphic to) the set Γ of all *causal functions* from the set of input streams I^ω to the set of output streams O^ω. In system theory, the input-output behavior of a linear discrete-time causal system is often described in terms of precisely such a causal stream function (traditionally called the *transfer* function of the system).

Note that we work in the category of sets and functions rather than vector spaces and linear maps. Although the functor $\mathcal{F}$ can also be defined on vector spaces, the function Φ defined above will in general not be linear, even if F, G, and H are. However, linearity of these maps *does* play a role in the various

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 425–446, 2007.
© Springer-Verlag Berlin Heidelberg 2007

characterisations of the semantics of linear systems, as we shall see later. (And, of course, vector addition in V is used in the definition of Φ.)

Once the functor (that is, the type of our systems) has been fixed and its final coalgebra identified, a coalgebraic treatment of linear systems follows from general insights of universal coalgebra: the *behaviour* (or semantics) of a system is given by the unique homomorphism into the final coalgebra; the image of this homomorphism constitutes the system's *minimisation*; and systems can be *specified* by elements of the final coalgebra and then *realised* (synthesised) by the corresponding generated subsystems of the final coalgebra.

The exercise mentioned in the title then consists of working out the details of all this. We view the formulation and the carrying out of this exercise as the main contribution of the present paper. Technically, we had to extend our earlier work [Rut03, Rut05] a bit in order to deal with streams of linear transformations, in Section 3. After recalling the coalgebraic treatment of (I, O)-systems, in Section 4, the main technical contribution lies in Section 5. It will be based on the elementary but crucial observation that the function Φ above factors through three maps of the following type (see (21)):

$$\Phi : \; V \longrightarrow V \times V \longrightarrow O \times V \longrightarrow (O \times V)^I$$

This is the basis for Theorem 8, which presents the final behaviour of (V, Φ) as the composition of three corresponding final homomorphisms. This final semantics f assigns to each (initial) state $v \in V$ a causal function $f(v) : I^\omega \to O^\omega$ (called the *transfer function* in system theory). This leads then to characterisations of system minimization and realisation, in Sections 6 and 7. Surprisingly, the final semantics f turns out to be the composition of a (linear) final mapping $H \times \tilde{F} : V \to O^\omega$ followed by a (non-linear) *injection* $g : O^\omega \to \Gamma$. As a consequence, minimization and realisation can be simply described in terms of just output streams, ignoring the presence of input streams altogether.

From the perspective of the theory of coalgebra, the relevance of our contribution consists of the following points. (i) It adds one more basic but important example to the family of mathematical structures that can be treated naturally and fairly completely by coalgebraic means. Other well-known examples are streams, automata, formal power series, infinite data types etc. (ii) Technically, the interaction between algebra and coalgebra is interesting. In general, (I, O)-systems (Mealy automata) live in the category of *sets*. As we shall see, *linear* (I, O)-systems are completely determined by their underlying linear O-systems (in which input plays no role), and these do live in the category of vector spaces and linear maps. As a consequence, the final behaviour of linear (I, O)-systems, which itself is obtained in Set, can be pleasantly characterised in terms of the basic operations (of sum and convolution product) of stream calculus. (iii) It also follows that streams – which constitute the prototypical example of a final coalgebra – are essentially all that is needed for the modelling of linear systems, since O-systems can be completely described in terms of O-streams. (iv) The final behaviour of *finite dimensional* linear (I, O)-systems will be characterised in terms of *rational* streams, in essentially the same way as finite deterministic

automata, which can be viewed as elementary non-linear (I, O)-systems, correspond to rational (regular) languages. (v) More generally, the present model shows that from a coalgebraic perspective, there is no essential difference between the treatment of linear and non-linear systems. This opens the way for future applications of coalgebraic techniques to non-linear phenomena in system theory.

Some of these points may also be of interest for system theory, where the semantics of the linear systems that we are considering is since long well understood (see, for instance, [Kai80]). In particular, our emphasis on the central role of (the final coalgebra of) streams leads to a very elementary treatment of system realisation, which – depending on taste and background – might be considered as a simpler alternative to Kalman's [Kal63, KFA69] classical construction using Hankel matrices. See the appendix for a further discussion of this.

We mention a few directions for further research. Since the semantics of both linear and non-linear systems is given by finality, it would be interesting to try and fit instances of non-linear systems from system theory (cf. [Son79]) into the coalgebraic framework. Also generalisations to *continuous* systems could be considered. Finally, one of the hallmarks of coalgebra is the notion of bisimulation, or observational equivalence, which comes along with every (functor) type of system. It should therefore be possible to study notions of equivalence for linear systems, including recently introduced ones such as in [Pap03] and [vdS04], from a coalgebraic perspective.

2 Preliminaries

We define the set of *streams* over a given set A by

$$A^\omega = \{\sigma \mid \sigma : \{0, 1, 2, \ldots\} \to A\}$$

We will denote elements $\sigma \in A^\omega$ by $\sigma = (\sigma(0), \sigma(1), \sigma(2), \ldots)$. We define the *stream derivative* of a stream σ by

$$\sigma' = (\sigma(1), \sigma(2), \sigma(3), \ldots)$$

and we call $\sigma(0)$ the *initial value* of σ. For $a \in A$ and $\sigma \in A^\omega$ we use the following notation:

$$a : \sigma = (a, \sigma(0), \sigma(1), \sigma(2), \ldots)$$

For instance, $\sigma = \sigma(0) : \sigma'$, for any $\sigma \in A^\omega$. Any function $f : A \to B$ induces a function

$$f^\omega : A^\omega \to B^\omega \qquad f^\omega(\sigma) = (f(\sigma(0)),\ f(\sigma(1)),\ f(\sigma(2)),\ \ldots) \tag{1}$$

Any function $f : A \to A$ induces a function

$$\tilde{f} : A \to A^\omega \qquad \tilde{f}(a) = (a,\ f(a),\ f^2(a),\ \ldots) \tag{2}$$

where $f^0 = 1$, the identity on A and $f^{n+1} = f \circ f^n$. If V is a set and W is a *vector space* (over some field k) then the set W^V of all functions

$$W^V = \{f \mid f : V \to W\}$$

is a vector space, with addition and scalar multiplication given, for $v \in V$ and $c \in k$, by

$$(f + g)(v) = f(v) + g(v) \qquad (c \cdot f)(v) = c \cdot f(v)$$

In particular, if V is a vector space over k then so is the set V^ω of all streams over V. Both the operations of initial value and derivative are linear transformations: for all $c, d \in k$, $\sigma, \tau \in V^\omega$,

$$(c \cdot \sigma + d \cdot \tau)(0) = c \cdot \sigma(0) + d \cdot \tau(0) \qquad (c \cdot \sigma + d \cdot \tau)' = c \cdot \sigma' + d \cdot \tau'$$

For any set A and $n \geq 1$, we denote the elements $v \in A^n$ by $v = (v_1, \ldots, v_n)$. It will sometimes be convenient to switch between streams of tuples and tuples of streams. We define the *transpose* as follows:

$$(-)^T : (A^n)^\omega \to (A^\omega)^n \qquad (\sigma^T)_i(j) = (\sigma(j))_i \tag{3}$$

This function is an isomorphism and has an inverse which we denote again by

$$(-)^T : (A^\omega)^n \to (A^n)^\omega$$

A *semi-ring* is a set R with a commutative operation of addition $c + d$; a (generally non-commutative) operation of multiplication $c \cdot d$ with $c \cdot (d + e) = (c \cdot d) + (c \cdot e)$ and $(d + e) \cdot c = (d \cdot c) + (e \cdot c)$; and with neutral elements 0 and 1 such that $c + 0 = c$, $1 \cdot c = c \cdot 1 = c$ and $c \cdot 0 = 0 \cdot c = 0$. If every $c \in R$ moreover has an additive inverse $-c$ (with $c + (-c) = 0$) then R is a *ring*.

Any field is a ring. The following example of a ring will be used later. Let V be a vector space (over some field k). The set $V \to_L V$ of linear maps $F : V \to V$ is a ring with addition and multiplication defined by

$$(F + G)(v) = F(v) + G(v) \qquad (F \times G)(v) = F(G(v))$$

and with the everywhere zero map and the identity map as neutral elements 0 and 1.

3 Stream Calculus

Let R be a ring. We define the following operators on the set R^ω of streams over R, for all $c \in R$, $\sigma, \tau \in R^\omega$, $n \geq 0$:

$$\begin{aligned}
[c] &= (c, 0, 0, 0, \ldots) \quad \text{(often simply denoted again by } c\text{)} \\
X &= (0, 1, 0, 0, 0, \ldots) \\
(\sigma + \tau)(n) &= \sigma(n) + \tau(n) \qquad \text{[sum]} \\
(\sigma \times \tau)(n) &= \sum_{i=0}^{n} \sigma(i) \cdot \tau(n - i) \qquad \text{[convolution product]}
\end{aligned}$$

(where $\cdot$ denotes multiplication in the ring R). A stream σ has a (unique) multiplicative inverse σ^{-1} in R^ω:

$$\sigma^{-1} \times \sigma = [1]$$

whenever its initial value $\sigma(0)$ has a multiplicative inverse $\sigma(0)^{-1}$ in R. As usual, we shall often write $1/\sigma$ for σ^{-1} and σ/τ for $\sigma \times \tau^{-1}$. Since $X^2 = (0,0,1,0,0,0,\ldots)$, $X^3 = (0,0,0,1,0,0,0,\ldots)$ and so on, the following infinite sum is well defined, for all $\sigma \in R^\omega$:

$$\sigma = \sigma(0) + (\sigma(1) \times X) + (\sigma(2) \times X^2) + \cdots$$

(Note that we write $\sigma(i)$ for $[\sigma(i)]$; similarly below.) It shows that σ can be viewed as a formal power series in the indeterminate X (which here in fact is a constant stream). What distinguishes our approach from formal power series is a systematic use of the operation of stream derivative and the universal property of finality it induces (see Section 4). This leads to a somewhat non-standard algebraic calculus, which we call *stream calculus*. We mention a few identities which are helpful for the computation of stream derivatives. (Computing stream derivatives is crucial in our approach to system *realisation*, in Section 7).

Lemma 1 ([Rut03]). *Let R be a ring. For all $\sigma, \tau \in R^\omega$,*

$$\begin{aligned} (\sigma + \tau)' &= \sigma' + \tau' \\ (\sigma \times \tau)' &= (\sigma' \times \tau) \; + \; (\sigma(0) \times \tau') \\ (\sigma^{-1})' &= -\sigma(0)^{-1} \times \sigma' \times \sigma^{-1} \end{aligned}$$

and $(\sigma + \tau)(0) = \sigma(0) + \tau(0)$, $(\sigma \times \tau)(0) = \sigma(0) \cdot \tau(0)$, and $\sigma^{-1}(0) = \sigma(0)^{-1}$ (if the latter exists). Moreover, $\sigma = \sigma(0) + (X \times \sigma')$ and $X \times \sigma = \sigma \times X$. □

We call a stream *polynomial* if it is of the form

$$c_0 + (c_1 \times X) + (c_2 \times X^2) + \cdots + (c_k \times X^k)$$

A stream is *rational* if it is the quotient $\sigma/\tau = \sigma \times \tau^{-1}$ of two polynomial streams σ and τ for which $\tau(0)^{-1}$ exists. We denote the set of all rational streams over R by

$$Rat(R^\omega) = \{\sigma \in R^\omega \mid \sigma \text{ is rational}\}$$

A prototypical example of a rational stream in R^ω, for $c \in R$, is

$$\frac{1}{1 - (c \times X)} = (1,\, c,\, c^2,\, \ldots)$$

If we consider the ring $V \to_L V$, for a vector space V, then streams $\phi \in (V \to_L V)^\omega$ are infinite sequences $\phi = (\phi(0), \phi(1), \phi(2), \ldots)$ of linear transformations $\phi(i) : V \to V$. For a linear transformation $F \in (V \to_L V)$, the example above becomes

$$\frac{1}{1 - (F \times X)} = (1, F, F^2, \cdots) \tag{4}$$

which, under the isomorphism $(V \to V)^\omega \cong V \to V^\omega$, is equal to $\tilde{F}$ defined in (2) above.

We shall also use the following type of convolution product. Let V and W be vector spaces. For streams $\phi \in (V \to_L W)^\omega$ and $\sigma \in V^\omega$, we define $\phi \times \sigma \in W^\omega$ by

$$(\phi \times \sigma)(n) = \sum_{i=0}^{n} \phi(i) \times \sigma(n-i) \tag{5}$$

where on the right we write $\phi(i) \times \sigma(n-i)$ for $\phi(i)(\sigma(n-i))$. For a linear map $H : V \to W$, we have as a special case

$$[H] \times \sigma = (H, 0, 0, 0, \ldots) \times \sigma = (H(\sigma(0)),\ H(\sigma(1)),\ H(\sigma(2)), \ldots)$$

which equals $H^\omega(\sigma)$ defined in (1) above. Note that if $W = V$, the set of streams $(V \to_L V)^\omega$ has itself also an operation of convolution product, which interacts nicely with the product defined in (5). For example, for $\phi, \psi \in (V \to_L V)^\omega$ and $\sigma \in V^\omega$,

$$(\phi \times \psi) \times \sigma = \phi \times (\psi \times \sigma) \tag{6}$$

Let k be a field. A linear transformation $F : k^n \to k^m$ between *finite dimensional* vector spaces corresponds to an $m \times n$ matrix M_F with values F_{ij} in k:

$$F : k^n \to k^m \qquad\qquad M_F = \begin{pmatrix} F_{11} & F_{12} & \cdots & F_{1n} \\ F_{21} & F_{22} & \cdots & F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ F_{m1} & F_{m2} & \cdots & F_{mn} \end{pmatrix}$$

Here and in what follows, the matrix is with respect to the standard basis

$$(1, 0, \ldots, 0),\ \ldots,\ (0, \ldots, 0, 1)$$

of k^n and k^m. Any stream $\phi = (\phi(0),\ \phi(1),\ \phi(2),\ \ldots)$ of linear transformations $\phi(i) : k^n \to k^m$ corresponds to a stream of matrices

$$(M_{\phi(0)},\ M_{\phi(1)},\ M_{\phi(2)},\ \ldots) = M_{\phi(0)} \ +\ (M_{\phi(1)} \times X) \ +\ (M_{\phi(2)} \times X^2) + \cdots$$

If we consider $M_{\phi(i)} \times X^i$ as an $m \times n$ matrix obtained from $M_{\phi(i)}$ by multiplying each of its entries by X^i, then the infinite sum on the right can itself be viewed as an $m \times n$ matrix M_ϕ with entries in k^ω:

$$(M_\phi)_{ij} = \ (M_{\phi(0)})_{ij} \ +\ ((M_{\phi(1)})_{ij} \times X) \ +\ ((M_{\phi(2)})_{ij} \times X^2) + \cdots \tag{7}$$

For the special case of $[H] = (H, 0, 0, 0, \ldots)$, for a linear transformation $H : k^n \to k^m$, we have

$$(M_{[H]})_{ij} = ((M_H)_{ij}, 0, 0, 0, \ldots) \tag{8}$$

We will let the context determine whether entries in k or k^ω are intended, and we shall simply write

$$M_{[H]} = M_H \tag{9}$$

The correspondence between ϕ and M_ϕ is given by the following commutative diagram:

$$\begin{array}{ccc} (k^n)^\omega & \xrightarrow{\phi\times(-)} & (k^m)^\omega \\ {\scriptstyle (-)^T}\downarrow & & \downarrow{\scriptstyle (-)^T} \\ (k^\omega)^n & \xrightarrow[M_\phi\times(-)]{} & (k^\omega)^m \end{array} \qquad (\phi \times \sigma)^T = M_\phi \times \sigma^T \tag{10}$$

(Recall the definition of $(-)^T$ from (3).) Here $\phi\times(-)$ denotes convolution product and $M_\phi \times (-)$ denotes matrix multiplication. Note that $M_1 = 1$, where 1 on the left denotes the stream $(1, 0, 0, 0, \ldots)$ (consisting of the identity map followed by zero maps), and 1 on the right denotes the identity matrix (having 1's on the diagonal and 0's everywhere else). Also note that

$$M_{\phi\times\psi} = M_\phi \times M_\psi \tag{11}$$

We have the following proposition.

Proposition 2. *Let $\rho \in (k^n \to_L k^n)^\omega$ be a stream of linear transformations $\rho(i) : k^n \to k^n$. If ρ is rational then M_ρ defined in (7) has entries in $Rat(k^\omega)$.*

Proof: Consider two polynomial streams $\phi, \psi \in (k^n \to_L k^n)^\omega$. The entries of the matrices M_ϕ and M_ψ are polynomial streams in k^ω. If ψ moreover has an inverse ψ^{-1} then $M_1 = 1$ and (11) imply $M_{\psi^{-1}} = (M_\psi)^{-1}$, which has values in $Rat(k^\omega)$. It follows that $M_{\phi\times\psi^{-1}} = M_\phi \times (M_\psi)^{-1}$ has values in $Rat(k^\omega)$. □

Example 3. Let $k = \mathbb{R}$ and let $F, G : \mathbb{R}^2 \to \mathbb{R}^2$ be linear transformations defined by

$$M_F = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \qquad M_G = \begin{pmatrix} 0 & -1 \\ 1 & 2 \end{pmatrix}$$

We compute the matrices of the rational streams $\tilde{F} = (\, 1 - (F \times X)\,)^{-1}$ and $\tilde{G} = (\, 1 - (G \times X)\,)^{-1}$:

$$M_{\tilde{F}} = \; (M_{1-(F\times X)})^{-1} = \begin{pmatrix} 1-X & -X \\ 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \frac{1}{1-X} & \frac{X}{1-X} \\ 0 & 1 \end{pmatrix}$$

$$M_{\tilde{G}} = \; (M_{1-(G\times X)})^{-1} = \begin{pmatrix} 1 & X \\ -X & 1-2X \end{pmatrix}^{-1} = \frac{1}{(1-X)^2} \cdot \begin{pmatrix} 1-2X & -X \\ X & 1 \end{pmatrix}$$

□

4 Systems Coalgebraically

We recall the coalgebraic semantics of systems with input and output. States, inputs and outputs will be represented by plain sets, and homomorphisms will

be simply functions between sets. In Section 5, we will look at the coalgebraic modelling of linear systems, involving vector spaces and linear maps.

A *system* (S, n) consists of a set S and a function $n : S \to S$, assigning to a state $s \in S$ its next state $n(s)$. We call the function n the *dynamics* of the system (S, n). A system $(S, \langle o, n\rangle)$ *with output in* a given set O (or simply O-system) consists of a set S of states, a function $n : S \to S$ and an *output* function $o : S \to O$. (Categorically speaking, an O-system is a *coalgebra* of the functor $O \times (-) : Set \to Set$.) A *homomorphism of O-systems* $(S, \langle o_S, n_S\rangle)$ and $(T, \langle o_T, n_T\rangle)$ is a function $h : S \to T$ such that $n_T \circ h = h \circ n_S$ and $o_T \circ h = o_S$; that is, such that the diagram below commutes:

$$\begin{array}{ccc} S & \xrightarrow{h} & T \\ {\scriptstyle \langle o_S, n_S\rangle}\downarrow & & \downarrow{\scriptstyle \langle o_T, n_T\rangle} \\ O \times S & \xrightarrow{1 \times h} & O \times T \end{array}$$

Here and throughout the paper, we use 1 to denote the identity function. The set of all streams O^ω is an O-system $(O^\omega, \langle h, t\rangle)$ where

$$h : O^\omega \to O, \quad h(\sigma) = \sigma(0) \quad \text{and} \quad t : O^\omega \to O^\omega, \quad t(\sigma) = \sigma'$$

(Recall that $\sigma' = (\sigma(1), \sigma(2), \sigma(3), \ldots)$.) Initial value and derivative are often called *head* and *tail*, hence our choice of symbols. The O-system $(O^\omega, \langle h, t\rangle)$ has the following universal property, called *finality*: For every O-system $(S, \langle o, n\rangle)$ there exists a unique homomorphism $f : (S, \langle o, n\rangle) \to (O^\omega, \langle h, t\rangle)$, called the *final behaviour* of $(S, \langle o, n\rangle)$. It is given by

$$\begin{array}{ccc} S & \overset{f}{\dashrightarrow} & O^\omega \\ {\scriptstyle \langle o, n\rangle}\downarrow & & \downarrow{\scriptstyle \langle h, t\rangle} \\ O \times S & \overset{1 \times f}{\dashrightarrow} & O \times O^\omega \end{array} \qquad f(s) = (o(s),\, o \circ n(s),\, o \circ n^2(s),\, \ldots)$$

where $n^0(s) = s$ and $n^{l+1}(s) = n(n^l(s))$.

Any system (S, n) (without output) is an S-system $(S, \langle 1, n\rangle)$ with output $1 : S \to S$ in S. We denote the corresponding final homomorphism by $\tilde{n}$:

$$\begin{array}{ccc} S & \overset{\tilde{n}}{\dashrightarrow} & S^\omega \\ {\scriptstyle \langle 1, n\rangle}\downarrow & & \downarrow{\scriptstyle \langle h, t\rangle} \\ S \times S & \overset{1 \times \tilde{n}}{\dashrightarrow} & S \times S^\omega \end{array} \qquad \tilde{n}(s) = (s,\, n(s),\, n^2(s),\, \ldots) \tag{12}$$

We call $\tilde{n}$ the *fully observable* behaviour of (S, n). The final behaviour f of an O-system $(S, \langle o, n\rangle)$ factors through its fully observable behaviour $\tilde{n}$ as follows:

$$
\begin{array}{ccccc}
S & \xrightarrow{\tilde{n}} & S^\omega & \xrightarrow{o^\omega} & O^\omega \\
\downarrow{\scriptstyle \langle 1,n\rangle} & & \downarrow{\scriptstyle \langle h,t\rangle} & & \\
S \times S & \xrightarrow{1\times\tilde{n}} & S \times S^\omega & & \downarrow{\scriptstyle \langle h,t\rangle} \\
\downarrow{\scriptstyle o\times 1} & & \downarrow{\scriptstyle o\times 1} & & \\
O \times S & \xrightarrow{1\times\tilde{n}} & O \times S^\omega & \xrightarrow{1\times o^\omega} & O \times O^\omega
\end{array}
\qquad f = o^\omega \circ \tilde{n} \tag{13}
$$

(with $f : S \to O^\omega$ along the top, $\langle o,n\rangle : S \to O \times S$ on the left, and $1\times f : O\times S \to O \times O^\omega$ along the bottom)

Next we consider systems with output *and* input. As before let O be a set of outputs. In addition, let I be an arbitrary set, the elements of which we call *inputs*. A system (S, ϕ) *with input in* I *and output in* O (or simply (I, O)-system) consists of a set S of states together with a function $\phi : S \to (O \times S)^I$. The function ϕ maps a state $s \in S$ to a function $\phi(s) : I \to O \times S$ that sends an input i to a pair $\phi(s)(i) \in O \times S$. We shall sometimes use the following notation:

$$s_1 \xrightarrow{i|o} s_2 \iff \phi(s_1)(i) = \langle o, s_2\rangle$$

which can be read as: in state s_1 and with input i the system changes to state s_2 while producing output o. Note that in general both the next state and the output depends on *both* the starting state and the input. Systems with input in I and output in O are also known in the literature as *Mealy machines* [Eil74]. Categorically, an (I, O)-system is a coalgebra of the functor $\mathcal{F} : \mathit{Set} \to \mathit{Set}$ defined by $\mathcal{F}(S) = (O \times S)^I$.

Let (S, ϕ_S) and (T, ϕ_T) be two (I, O)-systems. For $s_1 \in S$ and $i \in I$ let $\phi(s_1)(i) = \langle o, s_2\rangle$. A *homomorphism of* (I, O)*-systems* is a function $h : S \to T$ such that $\phi_T(h(s))(i) = \langle o, h(s_2)\rangle$, for all $s_1 \in S$ and $i \in I$. Equivalently, h should make the diagram below commute:

$$
\begin{array}{ccc}
S & \xrightarrow{h} & T \\
\downarrow{\scriptstyle \phi_S} & & \downarrow{\scriptstyle \phi_T} \\
(O \times S)^I & \xrightarrow{(1\times h)^1} & (O \times T)^I
\end{array}
$$

A *final* (I, O)-system can be constructed as follows. We call a function $g : I^\omega \to O^\omega$ *causal* (aka synchronous or letter-to-letter) if for any $\sigma \in I^\omega$ the n-th element of $g(\sigma)$ depends on only the first n elements of the input σ; that is,

$$\sigma(0) = \tau(0), \ldots, \sigma(n) = \tau(n) \;\Rightarrow\; g(\sigma)(n) = g(\tau)(n)$$

for all $\sigma, \tau \in I^\omega$ and $n \geq 0$. We denote the set of all causal functions by

$$\Gamma = \{\, g : I^\omega \to O^\omega \mid g \text{ is causal} \,\} \tag{14}$$

Let $g : I^\omega \to O^\omega$ be causal and let $i \in I$. We define the *initial output* of g on input i by

$$g[i] = g(i : \sigma)(0) \tag{15}$$

where $\sigma \in I^\omega$ is arbitrary. Note that the value $g[i] \in O$ does not depend on σ, since g is causal. We define the *stream function derivative* of g on input i by

$$g_i : I^\omega \to O^\omega, \quad g_i(\sigma) = g(i : \sigma)' \tag{16}$$

We obtain an (I, O)-system $(\Gamma, \gamma : \Gamma \to (O \times \Gamma)^I)$ by defining:

$$\gamma(g)\,(i) = \langle g[i],\, g_i \rangle$$

Proposition 4 ([Rut06, HCR06]). *The (I, O)-system (Γ, γ) of causal functions is* final*: for every (I, O)-system (S, ϕ) there exists a unique homomorphism*

$$\begin{array}{ccc} S & \overset{f}{\dashrightarrow} & \Gamma \\ {\scriptstyle\phi}\downarrow & & \downarrow{\scriptstyle\gamma} \\ (O \times S)^I & \overset{(1\times f)^1}{\dashrightarrow} & (O \times \Gamma)^I \end{array} \qquad \text{final behaviour of } (S, \phi)$$

Proof: Let $s_0 \in S$, $\sigma \in I^\omega$ and $n \geq 0$, and define

$$f(s_0)\,(\sigma)\,(n) = o_n \qquad \text{where} \quad s_0 \xrightarrow{\sigma(0)|o_0} s_1 \xrightarrow{\sigma(1)|o_1} \cdots \xrightarrow{\sigma(n)|o_n} s_{n+1}$$

Then $f(s_0)$ is causal and f is the unique function making the diagram above commute. □

5 Linear Systems Coalgebraically

We will now model linear systems coalgebraically, by simply applying the results from Section 4, and taking into account the fact that linear systems are defined in terms of vector spaces and linear maps. As before, we shall first treat systems with only output. Next we deal with systems that have both input and output.

We call a system (V, F) *linear* if the state space V is a vector space (over a given field k) and the dynamics $F : V \to V$ is a linear transformation. A system $(V, \langle H, F\rangle)$ with output in O is linear if in addition O is a vector space (over the same field k) and $H : V \to O$ is a linear transformation. A *homomorphism* of *linear* O-systems $(V, \langle H_V, F_V\rangle)$ and $(W, \langle H_W, F_W\rangle)$ is a homomorphism of O-systems which is linear:

$$\begin{array}{ccc} V & \xrightarrow{h} & W \\ {\scriptstyle\langle H_V, F_V\rangle}\downarrow & & \downarrow{\scriptstyle\langle H_W, F_W\rangle} \\ O \times V & \xrightarrow{1\times h} & O \times W \end{array} \qquad h \text{ is a linear transformation}$$

Recall from Section 4 that the O-system $(O^\omega, \langle h, t\rangle)$ is final among *all* (not necessarily linear) systems. We saw (in Section 2) that if O is a vector space then O^ω is also a vector space. Since initial value and derivative are linear transformations, $(O^\omega, \langle h, t\rangle)$ is a linear O-system. The final behaviour $f : V \to O^\omega$ of an O-system $(V, \langle H, F\rangle)$ is given, according to (13), by

$$V \xrightarrow{\tilde{F}} V^\omega \xrightarrow{H^\omega} O^\omega \quad (f = H^\omega \circ \tilde{F}) \qquad f(v) = H^\omega \circ \tilde{F}(v)$$

This is equivalent, for all $v \in V$, to

$$\begin{aligned} f(v) &= H^\omega \circ \tilde{F}(v) \\ &= (H(v),\, H \circ F(v),\, H \circ F^2(v), \ldots) \\ &= (H, 0, 0, 0, \ldots) \times (1, F, F^2, \ldots) \times (v, 0, 0, 0, \ldots) \quad \text{[using (5) and (6)]} \\ &= (H, 0, 0, 0, \ldots) \times \tilde{F} \times (v, 0, 0, 0, \ldots) \quad \text{[using } (1, F, F^2, \ldots) = \tilde{F}\text{, as in (4)]} \\ &= [H] \times \tilde{F} \times [v] \end{aligned}$$

Thus:

$$V \xrightarrow{\tilde{F} \times [-]} V^\omega \xrightarrow{[H] \times (-)} O^\omega \quad (f) \qquad f(v) = [H] \times \tilde{F} \times [v]$$

It follows that f is a linear transformation and that $(O^\omega, \langle h, t\rangle)$ is final in the family of all linear O-systems and linear homomorphisms between them.

The final behaviour of *finite dimensional* linear O-systems can be further characterised as follows. Let $n, m \geq 1$ and consider a system $(k^n, \langle H, F\rangle)$ with linear transformations $F : k^n \to k^n$ and $H : k^n \to k^m$. By (10), the following diagram commutes:

$$\begin{array}{ccccc} (k^n)^\omega & \xrightarrow{\tilde{F} \times (-)} & (k^n)^\omega & \xrightarrow{[H] \times (-)} & (k^m)^\omega \\ \downarrow (-)^T & & \downarrow (-)^T & & \downarrow (-)^T \\ (k^\omega)^n & \xrightarrow{M_{\tilde{F}} \times (-)} & (k^\omega)^n & \xrightarrow{M_H \times (-)} & (k^\omega)^m \end{array} \qquad ([H] \times \tilde{F} \times (-))^T = M_H \times M_{\tilde{F}} \times (-)^T \tag{17}$$

(where we use the convention (9) of writing $M_{[H]} = M_H$). It follows that the final behaviour f satisfies

$$f(v)^T \;=\; ([H] \times \tilde{F} \times [v])^T \;=\; M_H \times M_{\tilde{F}} \times [v]^T \tag{18}$$

We saw in (4) that $\tilde{F} = (1 - (F \times X))^{-1}$ is a rational stream. By Proposition 2, the matrix $M_{\tilde{F}}$ has values in $Rat(k^\omega)$. And so we have proved the following.

Proposition 5. *For a finite dimensional system $(k^n, \langle H, F\rangle)$ with dynamics $F : k^n \to k^n$ and output $H : k^n \to k^m$, the final behaviour $f : k^n \to k^m$ satisfies, for all $v \in k^n$,*

$$f(v)^T \;=\; M_H \times M_{\tilde{F}} \times [v]^T$$

and thus is obtained from $[v]^T$ by multiplication with an $m \times n$ matrix with values in $Rat(k^\omega)$. □

Example 6. Let $k = \mathbb{R}$ and consider the linear system $(\mathbb{R}^2, \langle H, F\rangle)$ with output $H : \mathbb{R}^2 \to \mathbb{R}$ and dynamics $F : \mathbb{R}^2 \to \mathbb{R}^2$ given by

$$H = \begin{pmatrix} 1 \; 1 \end{pmatrix} \qquad F = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$$

The matrix $M_{\tilde{F}}$ corresponding to $\tilde{F}$ has been computed in Example 3:

$$M_{\tilde{F}} = \begin{pmatrix} \frac{1}{1-X} & \frac{X}{1-X} \\ 0 & 1 \end{pmatrix}$$

The final behaviour $f_{\langle H,F\rangle} : \mathbb{R}^2 \to \mathbb{R}^\omega$ of this system is given, for any $(a, b) \in \mathbb{R}^2$, by

$$\begin{aligned} f_{\langle H,F\rangle}(a, b) &= \begin{pmatrix} 1 \; 1 \end{pmatrix} \times \begin{pmatrix} \frac{1}{1-X} & \frac{X}{1-X} \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} a \\ b \end{pmatrix} \\ &= \frac{a+b}{1-X} \end{aligned}$$

(omitting square brackets around a and b as usual). Repeating the example with a different output function $\bar{H}$ and the same dynamics F:

$$\bar{H} = \begin{pmatrix} 1 \; 2 \end{pmatrix} \qquad F = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$$

leads to the following final behaviour:

$$f_{\langle \bar{H},F\rangle}(a, b) \;=\; \begin{pmatrix} \frac{1}{1-X} & \frac{2-X}{1-X} \end{pmatrix} \times \begin{pmatrix} a \\ b \end{pmatrix} \;=\; \frac{a + 2b - bX}{1-X}$$

□

Next we discuss linear systems with *input and* output. We shall model them as (I, O)-systems, as defined in Section 4, and then study their final behaviour.

Let I, O and V be vector spaces over k, and let $F : V \to V$, $G : I \to V$ and $H : V \to O$ be linear transformations. We define the (I, O)-system $(V, \Phi_{\langle H,F,G\rangle})$ by

$$\Phi_{\langle H,F,G\rangle} : V \to (O \times V)^I \qquad \Phi_{\langle H,F,G\rangle}(v)(i) = \langle H(v), F(v) + G(i)\rangle \tag{19}$$

or equivalently, expressed in terms of transitions,

$$v \xrightarrow{\;i \,|\, H(v)\;} F(v) + G(i)$$

We call $(V, \Phi_{\langle H,F,G\rangle})$ a *linear* (I,O)-system because of the linearity of F, G, and H. However, note that Φ itself is not linear and likewise, homomorphisms of linear (I,O)-systems will generally not be linear. This is in contrast with the family of linear O-systems, where everything *is* linear.

For a linear(I,O)-system $(V, \Phi_{\langle H,F,G\rangle})$ we call $(V, \langle H, F\rangle)$ its *underlying* O-system. The key to the coalgebraic understanding of a linear (I,O)-system is the observation that its behaviour is in essence determined by that of its underlying O-system.

The following lemma will be helpful. Consider the final O-system $(O^\omega, \langle h,t\rangle)$ and an arbitrary linear transformation $\psi : I \to O^\omega$. This gives rise to a linear (I,O)-system $(O^\omega, \Phi_{\langle h,t,\psi\rangle})$, with $\Phi_{\langle h,t,\psi\rangle}$ defined as in (19). The lemma below describes its final behaviour $g : O^\omega \to \Gamma$, introduced in Proposition 4.

Lemma 7. *For all $\alpha \in O^\omega$ and $\sigma \in I^\omega$,*

$$\begin{array}{ccc} O^\omega & \xrightarrow{g} & \Gamma \\ \downarrow{\scriptstyle \Phi_{\langle h,t,\psi\rangle}} & & \downarrow{\scriptstyle \gamma} \\ (O \times O^\omega)^I & \xrightarrow[(1\times g)^1]{} & (O \times \Gamma)^I \end{array} \qquad g(\alpha)(\sigma) = \alpha + (\psi \times X \times \sigma)$$

(On the right, we read ψ as a stream of linear transformations $\psi \in (I \to_L O)^\omega \cong I \to_L O^\omega$.)

Proof: By finality of (Γ, γ), it is sufficient to show that the function g defined as above is a homomorphism of (I,O)-systems. By definition of γ, we have $\gamma(g(\alpha))(i) = \langle\, g(\alpha)[i],\, g(\alpha)_i \,\rangle$, for all $i \in I$. Now

$$g(\alpha)[i] = \; (\, g(\alpha)(i : \sigma)\,)\,(0) = \; \alpha(0)$$

and, for all $\sigma \in I^\omega$,

$$\begin{aligned} g(\alpha)(i : \sigma) &= g(\alpha)(i + (X \times \sigma)) \quad \text{[by Lemma 1, with } i = (i, 0, 0, 0, \ldots)] \\ &= \alpha + (\, X \times \psi \times (i + (X \times \sigma))\,) \\ &= \alpha + (X \times \psi \times i\,) + (X \times \psi \times X \times \sigma) \end{aligned} \tag{20}$$

This implies

$$\begin{aligned} g(\alpha)_i(\sigma) &= (\, g(\alpha)(i : \sigma)\,)' \quad \text{[definition stream function derivative (16)]} \\ &= (\, \alpha' + (\psi \times i)\,) + (\psi \times X \times \sigma) \quad \text{[using (20) and Lemma 1]} \\ &= g(\alpha' + \psi(i))(\sigma) \quad [\text{using } \psi \times i = \psi(i)] \end{aligned}$$

It follows that

$$\begin{aligned} \gamma(g(\alpha))(i) &= \langle\, \alpha(0),\, g(\alpha' + \psi(i))\,\rangle \\ &= (1 \times g)\,(\,\langle\, \alpha(0), \alpha' + \psi(i)\,\rangle\,) \\ &= \left((1 \times g)^1 \circ \Phi_{\langle h,t,\psi\rangle}\,(\alpha) \right)(i) \quad \text{[definition } \Phi_{\langle h,t,\psi\rangle} \text{ (19)]} \end{aligned}$$

This shows that the diagram above commutes. Thus g is a homomorphism. □

Next we observe that for a linear (I, O)-system $(V, \Phi_{\langle H,F,G\rangle})$, with $\Phi_{\langle H,F,G\rangle}$ as in (19), the function $\Phi_{\langle H,F,G\rangle}$ can be decomposed as follows:

$$V \xrightarrow{\langle 1,F\rangle} V \times V \xrightarrow{H\times 1} O \times V \xrightarrow{G_+} (O \times V)^I \qquad (\text{with } \Phi_{\langle H,F,G\rangle} : V \to (O\times V)^I \text{ as the composite}) \tag{21}$$

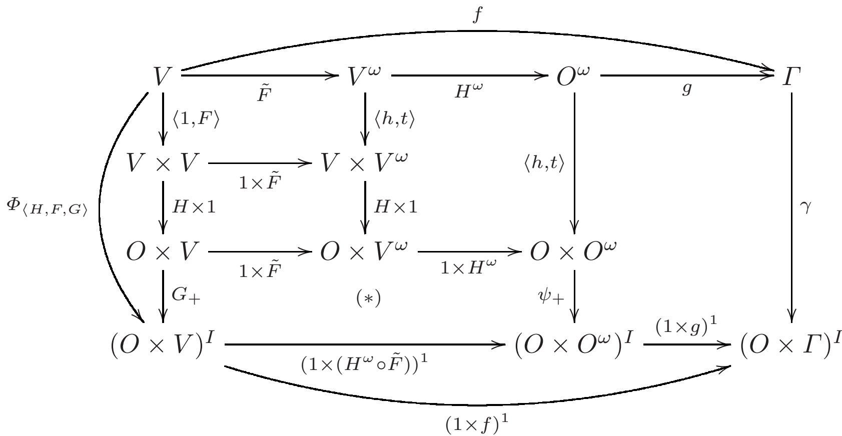
where the function G_+ is defined, for all $o \in O$, $v \in V$, and $i \in I$, by

$$G_+(\langle o, v\rangle)(i) = \langle o,\ v + G(i)\rangle$$

Theorem 8.
The final behaviour[1] $f : V \to \Gamma$ *of a linear* (I, O)*-system* $(V, \Phi_{\langle H,F,G\rangle})$ *(as defined in (19)) satisfies, for all* $v \in V$ *and* $\sigma \in I^\omega$*,*

$$f(v)(\sigma) = \left([H] \times \tilde{F} \times [v]^T\right) + \left([H] \times \tilde{F} \times [G] \times X \times \sigma\right)$$

Proof: Let $\psi : I \to O^\omega$ be defined by $\psi = [H] \times \tilde{F} \times [G]$ and consider the following diagram:



Recall that $H^\omega = [H] \times (-)$, using the convolution product introduced in (5) and, consequently, $H^\omega \circ \tilde{F} = [H] \times \tilde{F}$. The function ψ has been defined precisely such that the rectangle $(*)$ above commutes. (Note that a proof of $(*)$ will use the linearity of $[H] \times \tilde{F}$.) The right hand pentagon commutes by Lemma 7. Everything else commutes by finality. □

The final behaviour of *finite dimensional* linear (I, O)-systems can be further characterised, similarly to the case of linear O-systems. First we define for any causal function $g : (k^l)^\omega \to (k^m)^\omega$ a function.

[1] We observe that the final behaviour $f(0)$ of the initial state 0 corresponds to what is known in system theory as the *transfer function* of the system, where $\tilde{F}$ is often expressed as $(zI - F)^{-1}$.

$$\langle g \rangle : (k^\omega)^l \to (k^\omega)^m \qquad \langle g \rangle(\sigma) = g(\sigma^T)^T$$

for all $\sigma \in (k^\omega)^l$, and denote the image of Γ under this operation by $\langle \Gamma \rangle$. (Note that $\Gamma \cong \langle \Gamma \rangle$.)

Proposition 9.
For a finite dimensional linear (I, O)-system $(k^n, \Phi_{\langle H,F,G \rangle})$ with dynamics $F : k^n \to k^n$, input $G : k^l \to k^n$, and output $H : k^n \to k^m$, the final behaviour $\langle f \rangle : k^n \to \langle \Gamma \rangle$ satisfies, for all $v \in k^n$ and $\sigma \in (k^\omega)^l$,

$$\langle f(v) \rangle(\sigma) = \big(M_H \times M_{\tilde{F}} \times [v]^T \big) + (M_H \times M_{\tilde{F}} \times M_G \times X \times \sigma)$$

where all these matrices have values in $Rat(k^\omega)$.

Proof: By (10), all squares below commute:

$$\begin{array}{ccccccc}
(k^\omega)^l & \xrightarrow{M_G \times (-)} & (k^\omega)^n & \xrightarrow{M_{\tilde{F}} \times (-)} & (k^\omega)^n & \xrightarrow{M_H \times (-)} & (k^\omega)^m \\
\downarrow (-)^T & & \uparrow (-)^T & & \uparrow (-)^T & & \uparrow (-)^T \\
(k^l)^\omega & \xrightarrow[G \times (-)]{} & (k^n)^\omega & \xrightarrow[\tilde{F} \times (-)]{} & (k^n)^\omega & \xrightarrow[H \times (-)]{} & (k^m)^\omega
\end{array}$$

The proposition follows from this diagram and Theorem 8. As in Proposition 5, all matrices have values in $Rat(k^\omega)$. □

Example 10. (This is Example 6, continued.) Let $k = \mathbb{R}$ and consider the linear system (I, O)-system $(\mathbb{R}^2, \Phi_{\langle H,F,G \rangle})$ with $H : \mathbb{R}^2 \to \mathbb{R}$, $F : \mathbb{R}^2 \to \mathbb{R}^2$ and $G : \mathbb{R}^2 \to \mathbb{R}^2$ given by

$$H = \begin{pmatrix} 1 & 1 \end{pmatrix} \qquad F = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \qquad G = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}$$

The final behaviour $\langle f(a, b) \rangle : (\mathbb{R}^\omega)^2 \to \mathbb{R}^\omega$ of a state $(a, b) \in \mathbb{R}^2$ is given, for all pairs of input streams $(\sigma_1, \sigma_2) \in (\mathbb{R}^\omega)^2$, by

$$\begin{aligned}
\langle f(a, b) \rangle(\sigma) &= \left(M_H \times M_{\tilde{F}} \times \begin{pmatrix} a \\ b \end{pmatrix} \right) + \left(M_H \times M_{\tilde{F}} \times M_G \times X \times \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix} \right) \\
&= \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{1-X} & \frac{X}{1-X} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \quad + \\
&\quad \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{1-X} & \frac{X}{1-X} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} X \times \sigma_1 \\ X \times \sigma_2 \end{pmatrix} \\
&= \frac{a + (2X \times \sigma_1)}{1 - X} \quad + \quad \frac{b + (3X \times \sigma_2)}{1 - X}
\end{aligned}$$

□

6 Minimization and Equivalence

Because O- and (I, O)-systems are coalgebras, the general definition of coalgebraic equivalence applies. Here we spell out these definitions together with the observation that the corresponding minimization of a system is given by the (image under) the final behaviour mapping. For *linear* (I, O)-systems, we shall see that minimization and equivalence are particularly simple, as they are entirely determined by their underlying O-systems.

Equivalence of (not necessarily linear) O-systems is defined as follows. A relation $R \subseteq S \times T$ is called an *O-bisimulation* between O-systems $(S, \langle o_S, n_S \rangle)$ and $(T, \langle o_T, n_T \rangle)$ if for all $s \in S$ and $t \in T$:

$$\langle s, t \rangle \in R \Rightarrow \begin{cases} o_S(s) = o_T(t) \text{ and} \\ \langle n_S(s), n_T(t) \rangle \in R \end{cases}$$

We say that s and t are *O-equivalent* and write $s \sim_O t$ if there exists an O-bisimulation R with $\langle s, t \rangle \in R$. The final behaviour $f : S \to O^\omega$ of an O-system $(S, \langle o, n \rangle)$ identifies precisely all O-equivalent states: $s_1 \sim_O s_2$ iff $f(s_1) = f(s_2)$, for all $s_1, s_2 \in S$. (For the elementary proof, see [Rut03].) As a consequence, the minimization of an O-system with respect to O-equivalence is given by the image of S under f, which is a subsystem $f(S) \subseteq O^\omega$ because f is a homomorphism. It follows that if the system is linear, then the greatest O-equivalence on S is given by the kernel $ker(f)$.

For (I, O)-systems there exists a corresponding notion of (I, O)-equivalence and, again, the final behaviour identifies precisely all (I, O)-equivalent states: see [Rut06] for details. For *linear* (I, O)-systems, things are much simpler since their behaviour is determined by their underlying O-system.

Proposition 11. *The minimization of a linear (I, O)-system $(V, \Phi_{\langle H,F,G \rangle})$ is isomorphic to the minimization of its underlying O-system $(V, \langle H, F \rangle)$.*

Proof: By the proof of Theorem 8, the final behaviour $f : V \to \Gamma$ satisfies $f(v) = g(H \times \tilde{F} \times v)$, for all $v \in V$. Here the function $g : O^\omega \to \Gamma$ is given, according to Lemma 7, by $g(\alpha)(\sigma) = \alpha + (H \times \tilde{F} \times G \times X \times \sigma)$, for $\alpha \in O^\omega$ and $\sigma \in I^\omega$. Taking $\sigma = 0$, we see that g is injective. Thus the image of $(V, \Phi_{\langle H,F,G \rangle})$ under the final behaviour map f is isomorphic to its image under $H \times \tilde{F}$. The underlying O-system of this image is the minimization of $(V, \langle H, F \rangle)$. □

Example 12. Recall the (I, O)-system $(\mathbb{R}^2, \Phi_{\langle H,F,G \rangle})$ from Example 10. Computing its image W under $H \times \tilde{F}$ yields

$$W = \left(H \times \tilde{F} \right) (\mathbb{R}^2) = \{ \frac{a+b}{1-X} \mid (a, b) \in \mathbb{R}^2 \} \subseteq \mathbb{R}^\omega$$

Output and dynamics on W are induced by $\langle h, t \rangle : \mathbb{R}^\omega \to (\mathbb{R} \times \mathbb{R}^\omega)$. The input map on W is given by (the corestriction of) $\psi = H \times \tilde{F} \times G : \mathbb{R}^2 \to \mathbb{R}^\omega$ and satisfies

$$H \times \tilde{F} \times G = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{1-X} & \frac{X}{1-X} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{1-X} & \frac{3}{1-X} \end{pmatrix}$$

Choosing $1/1-X$ as a basis for W, we find that the resulting minimization is isomorphic to $\mathbb{R}$, with output and dynamics both given by $1 : \mathbb{R} \to \mathbb{R}$, and with input $(2\ 3) : \mathbb{R}^2 \to \mathbb{R}$. □

7 Realisation

We discuss the realisation of linear and non-linear systems, first with only output and then with input and output.

A state $s \in S$ in a (not necessarily linear) O-system $(S, \langle o, n \rangle)$ *realises* a stream $\sigma \in O^\omega$ if the final behaviour of s satisfies $f(s) = \sigma$. If O is a *set* (and not necessarily a vector space), a minimal realisation for a stream $\sigma \in O^\omega$ is obtained by taking as state space the set

$$S_\sigma = \{\sigma^{(0)}, \sigma^{(1)}, \sigma^{(2)}, \ldots\} \tag{22}$$

with $\sigma^{(0)} = \sigma$ and $\sigma^{(n+1)} = t(\sigma^{(n)}) = (\sigma^{(n)})'$. As output function and dynamics, one simply takes the restrictions of $h : O^\omega \to O$ and $t : O^\omega \to O^\omega$ to S_σ. The set inclusion $S_\sigma \subseteq O^\omega$ is a homomorphism of O-systems. By finality of $(O^\omega, \langle h, t \rangle)$, this homomorphism is unique. It follows that $f(\sigma) = \sigma$ and hence that $(S_\sigma, \langle h, t \rangle)$ with initial state σ is a minimal realisation of σ.

If O is a vector space then O^ω is also a vector space and we will be interested in realisations that themselves are vector spaces again. Thus a minimal realisation for a stream $\sigma \in O^\omega$ will consist of the smallest *subspace* of O^ω that contains σ and is closed under the linear transformation $t : O^\omega \to O^\omega$. This (so-called t-cyclic) vector space $Z_\sigma \subseteq O^\omega$ is the subspace of O^ω that is spanned by the set S_σ of vectors in (22).

Of special interest are those $\sigma \in O^\omega$ such that, for some $n \geq 1$, all of $\sigma = \sigma^{(0)}$ through $\sigma^{(n-1)}$ are linearly independent and

$$\sigma^{(n)} + \left(c_{n-1} \times \sigma^{(n-1)}\right) + \cdots + \left(c_1 \times \sigma^{(1)}\right) + \left(c_0 \times \sigma^{(0)}\right) = 0$$

for some coefficients $c_0, \ldots, c_{n-1}$ in the base field k of O and O^ω. Then Z_σ is a vector space of dimension n. The linear transformation $F : Z_\sigma \to Z_\sigma$ induced by $t : O^\omega \to O^\omega$ is given, with respect to the (ordered) basis $\sigma^{(0)}, \ldots, \sigma^{(n-1)}$, by the $n \times n$ matrix

$$M_F = \begin{pmatrix} 0 & 0 & \cdots & 0 & -c_0 \\ 1 & 0 & \cdots & 0 & -c_1 \\ 0 & 1 & \cdots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -c_{n-1} \end{pmatrix}$$

(This matrix is in fact (a variation of) the *companion* matrix of the so-called t-order polynomial of σ; cf. [BM77, Thm.15, p.339].) The linear transformation

$H : Z_\sigma \to O$ induced by $h : O^\omega \to O$ is given, again with respect to the basis $\sigma^{(0)}, \ldots, \sigma^{(n-1)}$, by the matrix (of size $dim(O) \times n$)

$$M_H = \begin{pmatrix} \sigma^{(0)}(0) & \sigma^{(1)}(0) & \sigma^{(2)}(0) & \cdots & \sigma^{(n-1)}(0) \end{pmatrix}$$

Thus we have obtained a linear O-system $(Z_\sigma, \langle H, F \rangle)$ of dimension n. As before, the inclusion $Z_\sigma \subseteq O^\omega$ is a homomorphism. Thus $f(\tau) = \tau$, for all $\tau \in Z_\sigma$ and $(Z_\sigma, \langle H, F \rangle)$ with σ as initial state is a minimal realisation of σ.

Example 13. Let $O = \mathbb{R}$ and consider the stream $\sigma = 1/(1-X)^2 \in O^\omega$. Computing the successive stream derivatives of $\sigma = \sigma^{(0)}$, using Lemma 1, gives

$$\sigma^{(1)} = \frac{2-X}{(1-X)^2} \qquad \sigma^{(2)} = \frac{3-2X}{(1-X)^2} = -\sigma^{(0)} + (2 \times \sigma^{(1)})$$

Thus $\sigma^{(0)}$ and $\sigma^{(1)}$ form a basis for Z_σ. Because $\sigma^{(0)}(0) = 1$ and $\sigma^{(1)}(0) = 2$, we have

$$M_H = \begin{pmatrix} 1 & 2 \end{pmatrix} \qquad M_F = \begin{pmatrix} 0 & -1 \\ 1 & 2 \end{pmatrix}$$

Now σ is realised by $(Z_\sigma, \langle H, F \rangle)$, with σ as the initial state. Clearly, $\mathbb{R}^2 \cong Z_\sigma$. Note that the isomorphism can also be obtained by computing the final behaviour $f : \mathbb{R}^2 \to \mathbb{R}^\omega$ of the O-system $(\mathbb{R}^2, \langle H, F \rangle)$, using Proposition 5. This gives, for all $(a, b) \in \mathbb{R}^2$,

$$\begin{aligned} f(a,b) &= M_H \times M_{\tilde{F}} \times (a,b) \\ &= \begin{pmatrix} 1 & 2 \end{pmatrix} \times \begin{pmatrix} \frac{1-2X}{(1-X)^2} & \frac{-X}{(1-X)^2} \\ \frac{X}{(1-X)^2} & \frac{1}{(1-X)^2} \end{pmatrix} \times \begin{pmatrix} a \\ b \end{pmatrix} \end{aligned}$$

which satisfies, as expected, $f(1,0) = \sigma$ and $f(0,1) = \sigma^{(1)}$. □

Example 14. Let $O = \mathbb{R}^2$ and consider the pair $(\tau, \sigma) \in (\mathbb{R}^\omega)^2 \cong (\mathbb{R}^2)^\omega$, with $\tau = 1/(1-2X)$ and $\sigma = 1/(1-X)^2$. Computing (pairs of) stream derivatives

$$(\tau,\sigma)^{(1)} = \left(\frac{2}{1-2X}, \frac{2-X}{(1-X)^2} \right) \qquad (\tau,\sigma)^{(2)} = \left(\frac{2^2}{1-2X}, \frac{3-2X}{(1-X)^2} \right)$$

$$(\tau,\sigma)^{(3)} = \left(\frac{2^3}{1-2X}, \frac{4-3X}{(1-X)^2} \right) = 2 \times (\tau,\sigma)^{(0)} - 5 \times (\tau,\sigma)^{(1)} + 4 \times (\tau,\sigma)^{(2)}$$

we see that $Z_{(\tau,\sigma)}$ has dimension 3 with $H : Z_{(\tau,\sigma)} \to \mathbb{R}^2$ and $F : Z_{(\tau,\sigma)} \to Z_{(\tau,\sigma)}$ given by

$$M_H = \begin{pmatrix} 1 & 2 & 4 \\ 1 & 2 & 3 \end{pmatrix} \qquad M_F = \begin{pmatrix} 0 & 0 & 2 \\ 1 & 0 & -5 \\ 0 & 1 & 4 \end{pmatrix}$$

□

Proposition 15. *Let k be a field and let $O = k^m$. A vector of streams $\sigma \in (k^\omega)^m \cong (k^m)^\omega$ is realisable by a linear k^m-system of finite dimension iff $\sigma \in (Rat(k^\omega))^m$.*

Proof: From left to right, this is Proposition 5. For the converse, it is sufficient to observe that the examples above generalise to arbitrary vectors of rational streams. This is immediate from the fact that for a rational stream $\sigma = \rho/\tau$, the dimension of Z_σ in the construction above is bounded by the maximum of the degrees of ρ and τ. □

Next we turn to systems with *input and output*. Let I and O be sets. A state s in a (not necessarily linear) (I, O)-system (S, ϕ) realises a causal stream function $g : I^\omega \to O^\omega$ if the final behaviour of s satisfies $f(s) = g$. For a given g, a (minimal) realisation is obtained by taking the smallest subsystem S of the final (I, O)-system (Γ, γ) containing g. The system S can be constructed by adding to the singleton set $\{g\}$ all successive stream function derivatives g_i, $(g_i)_j$, etc. (for $i, j, \ldots \in I$), and taking the restriction of γ to S. The inclusion $S \subseteq \Gamma$ is a homomorphism of (I, O)-systems and by finality we have $f(g) = g$. In [Rut06, HCR06], this approach is systematically applied to the realisation (synthesis) of various (non-linear) causal functions on bitstreams (with $I = O = \{0, 1\}$).

For infinite I and O, this construction will in general not be finitely computable. However, if both I and O are finite dimensional vector spaces then the realisation of linear causal stream functions can simply be reduced to the realisation problem of streams, which we have already solved above.

Proposition 16. *Let k be a field and let $I = k^l$ and $O = k^m$. Let $g : (k^\omega)^l \to (k^\omega)^m$ be given by $g(\tau) = M \times X \times \tau$, for an $m \times l$ matrix $M \in (k^\omega)^{m\times l}$. Then g is realisable by a linear (I, O)-system of finite dimension iff $M \in (Rat(k^\omega))^{m\times l}$.*

Proof: From left to right, this is Proposition 9. For the converse, we first consider the case that $l = 1$. So assume that $M \in (Rat(k^\omega))^m$. By Proposition 15, there exists a finite dimensional system $(V, \langle H, F\rangle)$ and $v \in V$ realising M; that is, $f(v) = M_H \times M_{\tilde{F}} \times v = M$. If we define $G : k \to V$ by the matrix $M_G = v$ then $(V, \Phi_{\langle H,F,G\rangle})$ with $0 \in V$ as initial state realises g since, for all $\tau \in k^\omega$,

$$\begin{aligned} f(0)(\tau) &= M_H \times M_{\tilde{F}} \times M_G \times X \times \tau \quad \text{[Proposition 9]} \\ &= M_H \times M_{\tilde{F}} \times v \times X \times \tau \\ &= M \times X \times \tau \\ &= g(\tau) \end{aligned}$$

For $l > 1$ we write M as a direct sum (product) $M = M_1 \oplus \cdots \oplus M_l$, with $M_i \in (Rat(k^\omega))^m$, for $i = 1, \ldots, l$. Then we construct realisations for each of $g_i = M_i \times X$. Their direct sum is a realisation for g. □

Acknowledgments. Discussions with Albert Benveniste – on [Rut05, Ben06] – were very instructive and are gratefully acknowledged. Thanks are also due to Jiri Adamek, H.Peter Gumm, Jan Komenda, Prakash Panangaden, Hans-E. Porst, and Jan van Schuppen, for discussions and pointers to the literature. I am also very grateful to the constructive comments of the three anonymous referees. Amongst others, their suggestions for improvements of the notation were very useful.

References

[AM74] Arbib, M.A., Manes, E.G.: Foundations of system theory: decomposable systems. Automatica 10, 285–302 (1974)

[AM75] Arbib, M.A., Manes, E.G.: Adjoint machines, state-behaviour machines, and duality. Journal of Pure and Applied Algebra 6, 313–344 (1975)

[Ben06] Benveniste, A.: A brief on realisation theory for linear systems (unpublished note)

[BM77] Birkhoff, G., MacLane, S.: A survey of modern algebra, 4th edn. MacMillan Publishing Co., Inc. (1977)

[Eil74] Eilenberg, S.: Automata, languages and machines. In: Pure and applied mathematics, vol. A, Academic Press, London (1974)

[Fuh96] Fuhrmann, P.A.: A polynomial approach to linear algebra. Springer, Berlin (1996)

[HCR06] Hansen, H., Costa, D., Rutten, J.J.M.M.: Synthesis of Mealy machines using derivatives. In: Proceedings of CMCS 2006. ENTCS, vol. 164(1), pp. 27–45. Elsevier, Amsterdam (2006)

[Kai80] Kailath, T.: Linear systems. Prentice-Hall, Englewood Cliffs (1980)

[Kal63] Kalman, R.E.: Mathematical description of linear dynamical systems. SIAM J. Control 1, 152–192 (1963)

[KFA69] Kalman, R.E., Falb, P.L., Arbib, M.A.: Topics in mathematical system theory. McGraw-Hill, New York (1969)

[Koh78] Kohavi, Z.: Switching and finite automata theory. McGraw-Hill, New York (1978)

[Lah98] Lahti, B.P.: Signal Processing & Linear Systems. Oxford University Press, Oxford (1998)

[Pap03] Pappas, G.J.: Bisimilar linear systems. Automatica 39, 2035–2047 (2003)

[Rut03] Rutten, J.J.M.M.: Behavioural differential equations: a coinductive calculus of streams, automata, and power series (Fundamental Study). Theoretical Computer Science 308(1), 1–53 (2003)

[Rut05] Rutten, J.J.M.M.: A tutorial on coinductive stream calculus and signal flow graphs. Theoretical Computer Science 343(3), 443–481 (2005)

[Rut06] Rutten, J.J.M.M.: Algebraic specification and coalgebraic synthesis of Mealy automata. In: Proceedings of FACS 2005. ENTCS, vol. 160, pp. 305–319. Elsevier Science Publishers, Amsterdam (2006)

[Son79] Sontag, E.D.: Polynomial response maps. Lecture Notes in Control and Information Sciences, vol. 13. Springer, Heidelberg (1979)

[vdS04] van der Schaft, A.J.: Equivalence of dynamical systems by bisimulation. IEEE Transactions on Automatic Control 49, 2160–2172 (2004)

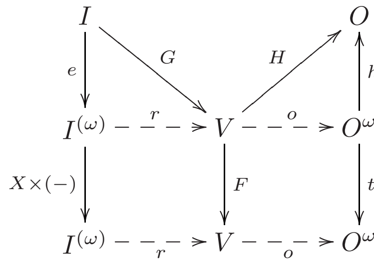
Appendix: A Comparison with Algebraic System Theory

In the wide area of (linear) system theory, our coalgebraic treatment of linear systems is probably closest related to what sometimes is called algebraic system theory. Below we give a brief overview of the approach of Kalman, who was one of the early contributors, and compare it to the present model. Classical references are [Kal63, KFA69], but see also [Kai80, Fuh96, Ben06]. Here we rely on the more categorical account of Kalman's model described in [AM74, AM75].

Let

$$I^{(\omega)} = \{ \, \sigma \in I^\omega \mid \sigma = (i_0, i_1, \ldots, i_k, 0, 0, 0, \ldots) \text{ for some } k \geq 0, i_j \in I \, \}$$

and consider the following diagram of vector spaces and linear maps:



(with $e(i) = (i, 0, 0, 0, \ldots)$.) The diagram can be viewed as a theorem stating that every choice of linear transformations G and F induces a unique *reachability* map r such that the left half of the diagram commutes, and similarly, every choice of F and H induces a unique *observability* map o fitting in the right half of the diagram. In this manner, every linear system (V, H, F, G) induces a unique map (called the *transfer function*) $o \circ r : I^{(\omega)} \to O^\omega$. It satisfies (in our notation)

$$o \circ r(\sigma) = \, H \times \tilde{F} \times r(\sigma) \tag{23}$$

where the state $r(\sigma)$ reached on input $\sigma = (i_0, i_1, \ldots, i_k, 0, 0, \ldots) \in I^{(\omega)}$ is given by

$$\begin{aligned} r(\sigma) &= \left(\tilde{F} \times G \times \sigma \right)(k) \\ &= G(i_0) \, + \, F \circ G(i_1) \, + \, F^2 \circ G(i_2) \, + \, \cdots \, + \, F^k \circ G(i_k) \end{aligned}$$

(Note that the operational interpretation is that i_k is the first input and i_0 is the last.) Comparing (23) with the final behaviour of V given in Theorem 8, we note the following differences: (i) The final behaviour allows arbitrary input streams, not only almost-everywhere-zero ones. (ii) The ordering of the inputs coincides with the input order. (iii) In (23), the behaviour of V is described in two steps: first r computes the state that is reached on finite input, then the (infinite) output stream is computed; in contrast, Theorem 8 describes the behaviour of an arbitrary initial state for all (infinite) streams of inputs.

Further differences between the two approaches can be noted regarding the way realisation is handled. In Kalman's approach, a realisation of a linear map g as in Proposition 16 is obtained by constructing its so-called (infinite) *Hankel* matrix H_g, viewing H_g as a linear transformation from $I^{(\omega)}$ to O^ω, and the observation that if H_g has finite rank then this linear transformation factors through a finite dimensional vector space V as in the diagram above. In contrast, Proposition 16 reduces realisation of linear maps to the realisation of streams, and the latter are simply given by the corresponding (t-cyclic) subspaces of O^ω.

Bootstrapping Types and Cotypes in HasCasl*

Lutz Schröder

DFKI-Lab Bremen and Department of Computer Science, University of Bremen

Abstract. We discuss the treatment of initial datatypes and final process types in the wide-spectrum language HasCasl. In particular, we present specifications that illustrate how datatypes and process types arise as bootstrapped concepts using HasCasl's type class mechanism, and we decribe constructions of types of finite and infinite trees that establish the conservativity of datatype and process type declarations adhering to certain reasonable formats. The latter amounts to modifying known constructions from HOL to avoid unique choice; in categorical terminology, this means that we establish that quasitoposes with an internal natural numbers object support initial algebras and final coalgebras for a range of polynomial functors, thereby partially generalizing corresponding results from topos theory.

1 Introduction

The formally stringent development of software in a unified process calls for wide-spectrum languages that support all stages of the development process, including requirements, design, and implementation. In the Casl language familiy [3], this role is played by the higher-order Casl extension HasCasl [19, 24]. Like in first-order Casl, a key feature of HasCasl is support for inductive datatypes, which appear in the specification of the functional correctness of software. In the algebraic-coalgebraic language CoCasl [11], this concept is complemented by coinductive types, which appear as state spaces of reactive processes. Many issues revolving around types of either kind gain in complexity in the context of the enriched language HasCasl; this is related both to the presence of additional language features such as higher order types and type class polymorphism and to the nature of the underlying logic of HasCasl, an intuitionistic higher order logic of partial functions without unique choice which may, with a certain margin of error, be thought of as the internal logic of quasitoposes.

Here, we discuss several aspects of HasCasl's concept of inductive datatype, as well as the perspective of adding coinductive types to HasCasl. To begin, we present the syntax and semantics of inductive datatypes, which may be equipped with reachability constraints or intiality constraints; both types of constraints may be relatively involved due to the fact that constructor arguments may have complex composite types. We then go on to show how initial datatypes may be specified in terms of HasCasl's type class mechanism. On the one hand,

* This work forms part of the DFG-project HasCasl (KR 1191/7-2).

T. Mossakowski et al. (Eds.): CALCO 2007, LNCS 4624, pp. 447–461, 2007.
© Springer-Verlag Berlin Heidelberg 2007

this shows that initial datatypes need not be regarded as a built-in language feature, but may be considered as belonging into a 'HasCasl prelude'. On the other hand, the specifications in question give a good illustration of how far the type class mechanism may be stretched. We then briefly discuss how a simple dualization of these specifications describes final process types in the style of CoCasl; thus, the introduction of such types into HasCasl would merely constitute additional syntactic sugar (although concerning the relationship to Casl and CoCasl, for both datatypes and process types certain caveats apply related to HasCasl's Henkin semantics).

Finally, we tackle the issue of the conservativity of datatype and process type declarations. We follow the method employed in standard HOL [14, 2], which consists in defining a universal type of trees and then carving out the desired inductive or coinductive types. However, the constructions need to be carefully adapted in order to cope with the lack of unique choice; in particular, it turns out that the existence of final process types hinges on the presence of an *internal* natural numbers object (NNO). Abstracting our results to the categorical level, we prove, in partial generalization of corresponding results for toposes [8], that any quasitopos with an internal NNO supports initial algebras and final coalgebras for certain classes of polynomial functors.

2 HasCasl

The wide-spectrum language HasCasl [19] extends the standard algebraic specification language Casl by intuitionistic partial higher order logic, equipped with a set-theoretic Henkin semantics, an extensive type class mechanism, and HOLCF-style support for recursive programming. HasCasl moreover provides support for functional-imperative specification and programming in the shape of monad-based computational logics [20, 22, 21, 25]. Tool support for HasCasl is provided in the framework of the Bremen heterogeneous tool set Hets [10]. We expect the reader to be familiar with the basic Casl syntax (whose use in our examples is, at any rate, largely self-explanatory), referring to [3, 12] for a detailed language description. Below, we briefly review the HasCasl language features most relevant for the understanding of the present work, namely type class polymorphism and certain details of HasCasl's higher order logic; cf. [24] for a full language definition.

The logic of HasCasl is based on the partial λ-calculus [9]. It is distinguished from standard HOL by having intutionistic truth values and partial function types $t \to ?\ s$ (besides total function types $t \to s$). The set-theoretic semantics is given by *intensional Henkin models*, where function types are equipped with application operators but are neither expected to contain all set-theoretic functions nor indeed to consist of functions; in particular, different elements of the function type may induce the same set-theoretic function. Such models are essentially equivalent to models in (varying!) partial cartesian closed categories (pccc's) with equality [18]; these categories are slightly more general than quasitoposes [1], which can be seen as finitely cocomplete pccc's with equality.

The difference between the HasCasl logic and the more familiar topos logic [7] is the absence of unique choice [18], where we say that a type a admits *unique choice* if a supports *unique description* terms of the form $(\iota x : a.\, \phi) : a$ designating the unique element x of a satisfying the formula ϕ (which may of course mention x), if such an element exists uniquely (this is like Isabelle/HOL's `THE` [13]). In HasCasl, the unique choice principle may be imposed if desired by means of a polymorphic axiom. The lack of unique choice requires additional effort in the construction of tree types establishing the conservativity of datatype and process type declarations; this is the main theme of Sect. 6. The motivation justifying this effort is twofold:

- Making do without unique choice essentially amounts to admitting models in quasitoposes (in fact, pccc's with equality) rather than just in toposes. Interesting set-based quasitoposes include pseudotopological spaces and reflexive relations; further typical examples are categories of extensional presheaves, including e.g. the category of reflexive logical relations, and categories of assemblies, both appearing in the context of realizability models [15, 16]. Quasitoposes also play a role in the semantics of parametric polymorphism [4].
- A discipline of avoiding unique choice leads to constructions which may be easier to handle in machine proofs than ones containing unique description operators (cf. e.g. the explicit warning in [13], Sec. 5.10).

HasCasl's shallow polymorphism revolves around a notion of type class. Type classes are syntactic subsets of *kinds*, where kinds are formed from *classes*, including a base class *Type* of all types, and the type function arrow $\to$. Classes are declared by means of the keyword **class**; e.g.

$$\textbf{class}\ \ \mathit{Functor} < \mathit{Type} \to \mathit{Type}$$

declares a class *Functor* of type constructors, i.e. operations taking types to types. Types are declared with associated classes (or with default class *Type*) by means of the keyword **type**; e.g. a type constructor F of class *Functor* is declared by writing

$$\textbf{type}\ \ F : \mathit{Functor}$$

Such declarations may be generic; e.g. if *Ord* is a class, then we may write

$$\begin{array}{ll}\textbf{var} & a, b : \mathit{Ord} \\ \textbf{type} & a \times b : \mathit{Ord}\end{array}$$

thus imposing that the class *Ord* is closed under products; note how the keyword **var** is used for both standard variables and type variables. Operations and axioms may be polymorphic over any class, i.e. types of operations and variables may contain type variables with assigned classes.

In order to ensure the institutional satisfaction condition (invariance of satisfaction under change of notation), polymorphism is equipped with an *extension semantics* [23]; the only point to note for purposes of this work is that as a consequence, a specification extension is, in Casl terminology, (model-theoretically)

conservative (i.e. admits expansions of models) iff it introduces names for entities already expressible in the present signature. In the case of types, this means that e.g. a datatype declaration is conservative iff it can be implemented as a subtype of an existing type.

3 Datatypes in HasCasl

HasCasl supports recursive datatypes in the same style as in Casl. To begin, an unconstrained datatype t is declared along with its constructors $c_i : t_{i1} \to \ldots \to t_{ik_i} \to t$ by means of the keyword **type** in the form

$$\textbf{type}\ t ::= \ c_1\ t_{11}\ \ldots\ t_{1k_1}\ \mid\ \ldots\ \mid\ c_n\ t_{n1}\ \ldots\ t_{nk_n}$$

(mutually recursive types are admitted as well, but omitted from the presentation for the sake of readability). Here, t is a pattern of the form $C\ a_1\ \ldots\ a_r$, $r \geq 0$, where C is the type constructor (or type if $r = 0$) being declared and the a_i are type variables. The t_{ij} are types whose formation may involve C and the a_i. Optionally, selectors $sel_{ij} : t \to ?\ t_{ij}$ may be declared by writing $(sel_{ij}\ :?t_{ij})$ in place of t_{ij}. All this is syntactic sugar for the corresponding declarations of types and operations, and equations stating that selectors are left inverse to constructors.

Data types may be qualified by a preceding **free** or **generated**. The **generated** constraint introduces an induction axiom; this corresponds roughly to term generatedness (‘no junk’). The **free** constraint (‘no junk, no confusion’) instead introduces an implicit fold operator, which implies both induction and a primitive recursion principle. If one of these constraints is used, then recursive occurrences (in the t_{ij}) of C are restricted to the pattern $C\ a_1\ \ldots\ a_r$ appearing on the left hand side; i.e. HasCasl does not support polymorphic recursion. If a **free** constraint is used, then additionally recursive occurrences of C are required to be strictly positive w.r.t. function arrows, i.e. occurrences in the argument type of a function type are forbidden. In more detail, the semantics of the constraints is as follows.

3.1 Generated Types

If t as above has only t and types from the local environment as arguments of constructors, then a generatedness constraint for t induces an induction axiom for t as in Casl. Unlike in Casl, the induction axiom is however expressible in HasCasl, i.e. generation constraints in HasCasl are just syntactic sugar. For constructors with composite argument types, the induction axiom more generally states that a predicate P on t holds universally if a family of *extended induction predicates* P_s on composite types s is closed under the constructors, where the P_s are subject to the following conditions. The base cases are $P_t = P$ and $P_s = \top$ if s is from the local environment or a type variable. The remaining clauses are

– *Partial function spaces*: $P_{s \to ?u} f \iff \forall x : s.(P_s x \wedge \operatorname{def} f(x)) \Rightarrow P_u f(x)$.
– *Total function spaces*: $P_{s \to u}$ is the restriction of $P_{s \to ?u}$ to $s \to u$.

- *Product types*: $P_{s\times u}(x,y) \iff P_s x \wedge P_u y$.
- *Applications* $D\ s_1\ \dots\ s_q$ of a type constructor D from the local environment to types $s_1,\dots,s_q$, where at least one s_j contains a recursive occurrence of t: $P_{D\ s_1\dots s_q}$ is required to be closed under all operations with result type $D\ s_1\ \dots\ s_q$ (which are necessarily newly arising instances of polymorphic operators). Note that $P_{D\ s_1\dots s_q}$ is not in general uniquely defined by this requirement.

Remark 1. If a type constructor D from the local environment has a generation constraint, then the closedness requirement on extended induction predicates for applications of D already follows from closedness under the operators in the constraint. However, the induction axiom also makes sense if D has no generation constraint; it then states that t is generated from the reachable part of D.

Generally, every HasCasl specification, in particular every datatype declaration, has a term model [18], which satisfies the above induction axiom. However, we stress that the induction axiom does *not* imply that elements of a generated datatype whose constructors have functional arguments are reachable by the constructors and λ-abstraction. In particular, induction axioms do not preclude interpreting functional types by full function spaces, which cannot be term generated for infinite types.

Finally, note that, due to Henkin semantics, generation constraints in HasCasl are weaker than in Casl, and in particular do not exclude non-standard models. However, proof-theoretically, this difference disappears — at least if the standard finitary induction rule is used. Only if stronger (e.g. infinitary) forms of induction are used, the difference becomes relevant.

Example 2. The following datatype declaration might form part of a specification of systems with unordered finite branching:

generated type $\mathit{Set}\ a ::= \mathit{empty} \mid \mathit{add}\ a\ (\mathit{Set}\ a)$
generated type $\mathit{Sys}\ b ::= \mathit{node}\ b\ (\mathit{Set}\ (\mathit{Sys}\ b))$

The induction axiom for *Set* is as in Casl; the induction axiom for *Sys b* is

$$\left.\begin{array}{l}(\forall x : b; s : \mathit{Set}\ (\mathit{Sys}\ b) \bullet Q\ s \Rightarrow P\ (\mathit{node}\ x\ s)) \wedge \\ Q\ \mathit{empty} \wedge \\ (\forall s : \mathit{Set}\ (\mathit{Sys}\ b); t : \mathit{Sys}\ b \bullet (Q\ s \wedge P\ t) \Rightarrow Q\ (\mathit{add}\ t\ s))\end{array}\right\} \Rightarrow \forall t : \mathit{Sys}\ b \bullet P\ t.$$

As an example with functional constructors, consider a datatype of at most countably branching trees,

generated type $\mathit{CTree}\ a ::= \mathit{leaf}\ a \mid \mathit{branch}\ (\mathit{Nat} \rightarrow ?\ \mathit{CTree}\ a)$

(with *Nat* previously declared), which for *CTree* gives rise to the induction axiom

$$\left.\begin{array}{l}(\forall x : a \bullet P\ (\mathit{leaf}\ x)) \wedge \\ (\forall f : \mathit{Nat} \rightarrow ?\ \mathit{CTree} \bullet \\ \qquad (\forall x : \mathit{Nat} \bullet \mathrm{def}\ f\ x \Rightarrow P\ (f\ x)) \Rightarrow P\ (\mathit{branch}\ f))\end{array}\right\} \Rightarrow \forall t : \mathit{CTree} \bullet P\ x.$$

3.2 Free Types

The semantics of free datatypes is determined by a fold operator, i.e. free datatypes are explicitly axiomatized as initial algebras. As indicated above, recursive occurrences of free types must be strictly positive, i.e. types like $L ::= abs\ (L \to L)$ and $L\ a ::= abs\ ((L \to a) \to a)$ are illegal, while

$$\textbf{free type}\ \mathit{Tree}\ a\ b ::= \mathit{leaf}\ b \mid \mathit{branch}\ (a \to \mathit{Tree}\ a\ b)$$

is allowed. Free datatypes are seen as initial algebras for functors. In the standard case, the functors in question are polynomial functors, with multiple arguments of constructors represented as products and alternatives represented as sums. E.g. the signature of the tree type above induces the functor F_{ab} given by

$$F_{ab}c = b + (a \to c).$$

The general mechanism for extracting functors from datatype declaration is explained in more detail in Sect. 4. This mechanism relies on type classes to ensure that user-defined type constructors appearing in constructor arguments are actually functors. The latter will in particular be the case if type constructors are defined as free datatypes with functorial parameters; e.g. the above declaration induces a functor taking b to $\mathit{Tree}\ a\ b$.

For now, we take for granted that a free datatype t as in the beginning of this section can be regarded as an initial algebra $\alpha : F\ t \to t$ for a functor F. Initiality is expressed by means of a polymorphic fold operation

$$\mathit{fold} : (F\ b \to b) \to t \to b$$

for $b : \mathit{Type}$, and an axiom stating that, for $d : F\ b \to b$, $\mathit{fold}\ d$ is the unique F-algebra morphism from α to d, i.e. the unique map f satisfying $d \circ (F\ f) = f \circ \alpha$.

Remark 3. Unlike in CASL, the meaning of **free type** does not coincide with that of the corresponding structured free extension **free { type** ... **}**, which would require all newly arising function types to be also freely term generated.

Remark 4. The reason for using an explicit fold operation in place of a combination of induction ('no junk') and term distinctness ('no confusion') is the absence of a unique choice operator, without which the existence of folds fails to be derivable from the no-junk-no-confusion principles. (E.g. in the quasitopos of pseudotopological spaces, the no-junk-no-confusion axioms determine the underlying set of an initial algebra but not its topological structure.) Conversely, one easily sees that initiality implies induction and term distinctness.

From the fold operation, one obtains a primitive recursion principle in the standard way (i.e. by means of a simultanuous recursive definition of the identity). From the latter, in turn, we obtain as a special case a case operator, denoted in the form

$$\mathit{case}\ x\ \mathit{of}\ c\ y_1\ \ldots\ y_l \to f\ y_1\ \ldots\ y_l \mid \ldots$$

Example 5. Consider the following free datatype definitions.

free type *List* a ::= *nil* | *cons* (a; *List* a)
free type *Tree* a b ::= *leaf* a | *branch* ($b \rightarrow$ *List* (*Tree* a b))

The declaration of *List* a induces the standard fold operation for lists. Moreover, the type class mechanism (cf. Sect. 4) recognizes automatically that the type constructor *List* is a functor, and in particular generates the standard *map* operation. For *Tree*, we obtain a polymorphic fold operation

$$\mathit{fold} : (a \rightarrow c) \rightarrow ((b \rightarrow \mathit{List}\ c) \rightarrow c) \rightarrow \mathit{Tree}\ a\ b \rightarrow c.$$

This operation is axiomatized as being uniquely determined by the equations

$$\mathit{fold}\ f\ g\ (\mathit{leaf}\ x) = f\ x \quad \text{and} \quad \mathit{fold}\ f\ g\ (\mathit{branch}\ s) = g\ (\mathit{map}\ (\mathit{fold}\ f\ g) \circ s).$$

4 Initiality Via the Type Class Mechanism

The concept of free datatype described in the previous section may be regarded as bootstrapped, i.e. as being a HasCasl library equipped with built-in syntactic sugar rather than a basic language feature. The crucial point here is that HasCasl's type class mechanism allows talking about functorial signatures, algebras for a functor, and, importantly, homomorphisms. The specifications shown below are real HasCasl specifications, parsed and prettyprinted using the Heterogeneous Tool Set (Hets) [10].

Figure 1 shows the constructor class of functors. Mutually recursive or parametrized datatypes require n-ary functors for $n \in \mathbb{N}$, and in fact occasionally type constructors which are functorial only in some of their arguments; since HasCasl does not feature dependent classes, the corresponding classes need to be specified one by one, as exemplified in Fig. 1 by a specification of bifunctors. This is not a problem in practice, as typically only small values of n are needed; the specification of bifunctors illustrates how $n+1$-ary functors can be specified recursively in terms of n-ary functors.

Remark 6. One might envision a single specification of functors of arbitrary finite arity by abuse of syntax, as follows: declare a class *Typelist* and type constructors *Nil* : *Typelist*, *Cons* : *Type* $\rightarrow$ *Typelist* $\rightarrow$ *Typelist*, and define *Functor* as a subclass of *Typelist* $\rightarrow$ *Type*. (Undesired semantic side effects may be eliminated by specifying the types *Nil*, *Cons* a *Nil* etc. to be singletons.) Similar tricks work in Haskell [6] but rely on multi-parameter type classes, which are currently excluded from the HasCasl design.

For purposes of conservativity of datatype declarations, the class of polynomial functors (bifunctors etc.), shown in Fig. 2, plays an important role. An n-ary functor is polynomial if it can be generated from projection functors (the identity functor if $n = 1$) and constant functors by taking finite sums and products. To avoid circularity, the required type constructors in have been specified without

```
spec FUNCTOR =
     vars  a, b, c : Type; x : a; f : a → b; g : b → c
     ops   __comp__ : (b → c) × (a → b) → a → c;
           id : a → a
     • id x = x
     • (g comp f) x = g (f x)
     class Functor < Type → Type
     {vars  a, b, c : Type; F : Functor; f : a → b; g : b → c
     op    map : (a → b) → F a → F b
     • map id = id : F a → F a
     • map (g comp f) : F a → F c = (map g) comp (map f)
     }
     class Bifunctor < Type → Functor
     {vars  a, b, c, d : Type; F : Bifunctor; f : a → b; g : b → c
     op    parmap : (a → b) → F a d → F b d
     • parmap id = id : F a d → F a d
     • parmap (g comp f) : F a d → F c d = (parmap g) comp (parmap f);
     • (parmap f) comp (map h) : F a c → F b d = (map h) comp (parmap f)
     }
```

Fig. 1. HasCasl specification of functors

recourse to free type declarations. Recall that constructor/selector pairs such as *mkId*/*getId* impose that the selector *getId* is left inverse to the constructor *mkId*. Axioms stating surjectivity of constructors are omitted in the figure except in the case of *Prod* (for *Sum*, joint surjectivity of *inl* and *inr* follows from the axiom for *sumcase*). Similarly, the obvious definitions of the associated *map* operations are omitted, except in the case of sums. Note that HasCasl does not provide a way to exclude unwanted ('junk') further instance declarations for the class *PolyFunctor*, i.e. to say that the class is generated by the given generic instances.

In Fig. 3, we present a specification of algebras for a functor. The set of algebra structures for a functor F on a type a is given by the type constructor *Alg*, which depends on both F and a and thus has the profile *Functor* → *Type* → *Type*; it is given as a type synonym for the type $F\ a \to a$. Similarly, the type constructor *AlgMor* for algebra morphisms depends on F and types a, b forming the carriers of the domain and the codomain, respectively. Algebra morphisms are treated as triples consisting of two algebra structures and a map between the carriers.

Initial algebras are then specified by means of two operations: a type constructor *InitialCarrier* that assigns to a functor the carrier set of its initial algebra, and a polymorphic constant *initialAlg* which represents the structure map of an initial algebra for F on this carrier. Initiality of this algebra is specified by means of an explicit fold operation. As initial algebras will exist only for some functors, the abovementioned operations are defined only on a subclass *DTFunctor* ('datatype functor') of *Functor*. We declare the class *PolyFunctor* (Fig. 2) to be a subclass of *DTFunctor*, thus stating that all polynomial functors have initial

```
spec PolyFunctors = Functor
then class  PolyFunctor < Functor
     vars   F, G : PolyFunctor; a, b : Type
     types  Const a, Id, Sum F G, Prod F G : PolyFunctor
     type   Const a b ::= mkConst (getConst : a)
     type   Id b ::= mkId (getId : b)
     type   Sum F G b ::= inl (F b) | inr (G b)
     vars   f : F b →? a; g : G b →? a; h : Sum F G b →? a
     op     sumcase : (F b →? a) → (G b →? a) → (Sum F G) b →? a
     • h = sumcase f g ⇔
            ∀ x : F b; y : G b • h (inl x) = f x ∧ h (inr y) = g y
     type   Prod F G b ::= pair (outl : F b; outr : G b)
     var    k : a → b
     • (map k : Sum F G a → Sum F G b) =
                 sumcase (inl comp (map k)) (inr comp (map k))
     •...
     • ∀ z : Prod F G b • z = pair (outl z, outr z);
     •...
     class  PolyBifunctor < Type → PolyFunctor; PolyBifunctor < Bifunctor
     ...
```

Fig. 2. HasCasl specification of polynomial functors

algebras as proved in Sect. 6; due to possible junk in the class *PolyFunctor* (see above), this is consistent but non-conservative. Moreover, we state that initial algebras depend functorially on parameters in the case of polynomial bifunctors and that the arising functor again has an initial algebra (as nested recursion may be coded by mutual recursion in the standard way [5]).

We conclude with a brief description of how the data above are generated by the static analysis of actual HasCasl specifications. The functor F associated to the declaration of a datatype t as in the beginning of Sect. 3 is a sum of n functors F_i, one for each constructor c_i; the functor F_i, in turn, is a product of k_i functors F_{ij}, corresponding to the t_{ij}. The t_{ij} are, by the restrictions laid out in Sect. 3, inductively generated from types in the local environment, t, and the a_i by taking products, exponentials $s \to t$ or $s \to ?\, t$, where s is a type formed from the a_i and the local environment, and applications $D\ s_1\ \dots\ s_l$ of type constructors from the local environment, the latter subject to the restriction that if s_i contains a recursive occurrence of t, then the dependence of D of its i-th argument must be functorial. The latter property is tracked by means of the type class mechanism; in particular, instances of *Functor* are generated automatically for parametrized datatypes such as the type *List* a of Example 5. Given this format of the t_{ij}, it is straightforward to associate a functor to each t_{ij} (using further generic instances of *Functor*, in particular exponentials and closure under functor composition). Finally, an instance $F : DTFunctor$ is generated if not already induced by the generic instances shown in Fig. 3, in which case consistency of the datatype declaration becomes the responsibility of the user.

spec ALGEBRA = POLYFUNCTORS
then vars $F : Functor;\ a, b : Type$
 type $Alg\ F\ a := F\ a \to a$
 op $__::__\to__ : Pred\ ((a \to b) \times (Alg\ F\ a) \times (Alg\ F\ b))$
 vars $f : a \to b;\ alpha : Alg\ F\ a;\ beta : Alg\ F\ b$
 • $(f :: alpha \to beta) \Leftrightarrow (beta\ comp\ (map\ f)) = (f\ comp\ alpha)$
 type $AlgMor\ F\ a\ b = \{(f, alpha, beta) : (a \to b) \times Alg\ F\ a \times Alg\ F\ b\ \bullet$
 $f :: alpha \to beta\ \}$
 classes $DTFunctor < Functor;\ PolyFunctor < DTFunctor$
 {**vars** $F : DTFunctor;\ a : Type$
 type $InitialCarrier\ F$
 ops $initialAlg : Alg\ F\ (InitialCarrier\ F);$
 $ifold : Alg\ F\ a \to InitialCarrier\ F \to a$
 vars $alpha : Alg\ F\ a;\ g : InitialCarrier\ F \to a;$
 • $(g :: initialAlg \to alpha) \Leftrightarrow g = ifold\ alpha;$
 }
 var $G : PolyBifunctor$
 type $ParamDT\ G\ a ::= mkPDT\ (getPDT : InitialCarrier\ (G\ a))$
 • $\forall\ l : ParamDT\ G\ a \bullet mkPDT\ (getPDT\ l) = l$
 type $ParamDT\ G : DTFunctor$
 vars $l : ParamDT\ G\ a;\ b : Type;\ f : a \to b$
 • $map\ f\ l = mkPDT\ (ifold\ (initialAlg\ comp\ (parmap\ f))\ (getPDT\ l))$

Fig. 3. HASCASL specification of initial algebras

Using the *sumcase* operation of Fig. 2, one can now gather the constructors into a structure map $c : F\ t \to t$; the freeness constraint in the above datatype declaration then translates into the declaration of a two-sided inverse g of $ifold\ c$. The fold operation on t is obtained as $fold\ \alpha = (ifold\ \alpha) \circ g$.

5 Process Types in HASCASL

Although process types in the style of COCASL, so-called *cotypes* [11], are not presently included in the HASCASL design, the results of the previous section indicate that cotypes could be integrated seamlessly into HASCASL. A cotype is a syntactic representation of a coalgebra for a signature functor. Cotypes are declared in a similar style as types; the crucial difference is that, while selectors are optional in a datatype, they are mandatory in a cotype, as they constitute the actual structure map of the coalgebra, and constructors are optional. Moreover, a cotype induces axioms guaranteeing that the domains of selectors in the same alternative agree, and that the domains of all alternatives are disjoint and jointly exhaustive. Thus, the intended models e.g. of the cotype

cotype $Proc ::= (out :?a; next :?Proc)\ |\ (spawnl, spawnr :?Proc)$

are coalgebras for the functor F given by $FX = a \times X + X \times X$. The extraction of functors from cotype signatures is analogous to the case of types as explained in Sect. 4. The semantics of cotypes, in particular *cofree* (i.e. final) cotypes, builds on a dual of the specification of algebras (Fig. 3), where the type of algebras is replaced by a type $\mathit{Coalg}\ F\ a := a \to F\ a$, the definition of homomorphisms is correspondingly modified, and initiality is replaced by finality, i.e. unique existence of morphisms *into* the final coalgebra given by an *unfold* operation. The domains of the selectors can in principle be arbitrary types; of course, this will not in general guarantee existence of final coalgebras. We omit the discussion of cogeneratedness of cotypes.

The only subtle point in the matching between cotype declarations in HasCasl and coalgebras is that, unlike in a set-based framework such as CoCasl, the abovementioned conditions on domains of selectors of a cotype t with associated functor F are insufficient to guarantee existence of a single structure map $t \to F\ t$, the point being, again, the absence of unique choice. We thus impose, instead of just disjointness and joint exhaustiveness of the domains, that the cotype is the coproduct of the domains, by introducing a polymorphic partial case operation similar to the *sumcase* operation of Fig. 2. E.g. for the cotype *Proc* above and $f, g : \mathit{Proc} \to ?\ a$, $\mathit{case}\ f\ g = h : \mathit{Proc} \to a$ is defined whenever the domains of f and g equal the domains of *out* and *spawnl*, respectively, and in this case h extends f and g. (Under unique choice, *case* is definable.)

6 Conservativity of Datatypes and Process Types

Free datatypes in HasCasl are not necessarily conservative extensions of the local environment. Already the naturals may be non-conservative: as discussed in Sect. 2, conservative extensions can only introduce names for entities already in the present signature, and a given model might interpret all types as finite sets. This problem arises already in standard HOL, where the construction of initial datatypes [14, 2] is based on the naturals. The said construction makes heavy use of unique choice, so that the question arises whether similar constructions are possible in HasCasl.

It turns out that the construction of finitely branching datatypes can be modified to avoid unique choice, assuming, besides a type *Nat* of natural numbers with associated primitive recursion principle, sum types (denoted by + in the interest of readability, with injection functions *inl* and *inr* as usual and extraction functions $\mathit{outl} : a + b \to ?\ a$, $\mathit{outr} : a + b \to ?\ b$) and an initial type, which shows up in the form of on undefined partial constant $\mathit{bot} :?a$ at every type a (under unique choice, sum types and an initial type can be constructed). We describe the construction for the simplified situation where we have a single unparametrized datatype t with n constructors c_i of arity k_i, with arguments either of type t or of some type a (extending this to mutually recursive types, complex argument types — excluding function types —, and parameters is straightforward).

To begin, the type *Path* is defined as $\mathit{Nat} \to ?\ \mathit{Nat}$. We put $\mathit{nil} = \lambda m \bullet m\ \mathit{res}\ \mathit{bot}$ and $\mathit{cons}\ m\ s = \lambda k \bullet \mathit{case}\ k\ \mathit{of}\ 0 \to m \mid r+1 \to s\ r$. The

crucial point is now the definition of the universal type: while in the construction of [14, 2], this is a type of sets of nodes, we put

$$DTree\ a = Path \to ?\ ((a + Nat) \times Nat),$$

where for $f : DTree\ a$ and $p : Path$, $f\ p = (x, n)$ indicates that the subtree at p has size n and is either a leaf labelled y, if $x = inl\ y$, or a node labelled by the constructor c_i, if $x = inr\ i$. We embed a into $DTree\ a$ and define the constructors c_i as operations on $DTree\ a$ in the obvious way (with sizes determined by counting 1 for each leaf or constructor), and then take t to be the smallest subtype of $DTree\ a$ closed under the c_i. We put $size\ z = snd\ (z\ nil)$ for $z : t$, and for $j : Nat$, we define a generic j-th selector by $sel_j\ z = z \circ (cons\ j)$. Note that $size\ z > 0$ for all $z : t$.

It remains to be shown that we can construct the function $fold\ d_1\ \dots\ d_n$, for functions d_i representing an algebra on a type b. We define a primitive recursive function $f : Nat \to ((DTree\ a) \to ?\ (a + b))$ by

$$\begin{aligned} f\ 0\ z &= bot \\ f\ (n+1)\ z &= (case\ fst\ (z\ nil)\ of\ inl\ y \to inl\ y \\ &\qquad\qquad | \ inr\ i \to inr\ (d_i\ (f\ n\ (sel_1\ z))\ \dots\ (f\ n\ (sel_{n_i}\ z))) \end{aligned}$$

(with extraction functions $outl$, $outr$ on $a+b$ appropriate for the argument types of d_i left implicit), and put $fold\ d_1\ \dots\ d_n\ z = outr\ (f\ (size\ z)\ z)$.

In summary, the main changes w.r.t. standard HOL with unique choice concern the universal type, which is a type of partial functions rather than relations (reflecting the fact that functional relations need not be functions in the absence of unique choice), and the construction of primitive recursive functions, which can no longer rely on an inductive construction of their graphs. It is an open problem whether our use of the size function for this purpose can be either generalized or avoided so as to cover also infinitely branching datatypes such as the type $Tree\ a\ b$ from Example 5.

Somewhat surprisingly, a quite similar construction works also for final coalgebras. For simplicity, assume a cotype declaration of the form

$$\textbf{cotype}\ t ::= (sel_{11} : t_{11}; \dots; sel_{1m_1} : t_{1m_1}) \mid \cdots \mid (sel_{n1} : t_{n1}; \dots; sel_{nm_n} : t_{nm_n}),$$

where for some $1 \leq k_i \leq m_i$ and some type a from the local environment (the output type), $t_{ij} = t$ if $j \leq k_i$, and otherwise $t_{ij} = a$. (The generalization to mutually recursive cotypes, several output types, and types t_{ij} of the form $b \to s$, where b is an input type from the local environment, is straightforward.) Given that initial datatypes have already been constructed, we may now take the type $Path$ to be the type $List\ Nat$ of lists of natural numbers, with constructors nil, $cons$ and the standard $snoc$ operation. Our universal type is then

$$PTree\ a = Path \to ?\ (Nat + a)$$

(where it is crucial that we omit the additional Nat component present in $DTree\ a$). For $f : PTree$ and $p : Path$, the intended reading of $f\ p = x$ is

that position p in the tree is a leaf labelled with output y if $x = inr\ y$, and a branch according to the i-th alternative in the above declaration if $x = inl\ i$. The carrier of the final cotype for the above declaration is then the subtype c of $PTree\ a$ consisting of those f such that

$$def\ (outl\ (f\ nil)) \quad \text{and}$$
$$def\ (f\ (snoc\ p\ j)) \iff (\exists i : Nat.\, f\ p = inl\ i \wedge 1 \leq j \leq k_i).$$

(Note that $f\ p = inl\ i$ entails that $f\ p$ is defined.) For $1 \leq j \leq m_i$, we can then define $sel_{ij}\ f$ to be defined iff $f\ nil = inl\ i$, and in this case

$$sel_{ij}\ f = \begin{cases} \lambda p.\ f\ (cons\ j\ p) & j \leq k_i \\ outr\ (f\ (cons\ j\ nil)) & \text{otherwise,} \end{cases}$$

and these selectors can be gathered into a single coalgebra structure on c for the appropriate functor using the information from $outl\ f\ nil$. Given a further cotype d matching the above declaration, with selectors sel^d_{ij} gathered into a coalgebra structure α, we may, using the *case* operator on d discussed in Sect. 5, define a function *alt* giving, for $x : d$, the alternative $i = alt\ x : Nat$ relevant for x. We then define the morphism $u = unfold\ \alpha : d \to c$ recursively by

$$u\ x\ nil = inl\ i \quad \text{and} \quad u\ x\ (cons\ j\ p) = \begin{cases} u\ (sel^d_{ij}\ x)\ p & 0 \leq j \leq k_i \\ inr\ sel^d_{ij}\ x & k_i < j \leq m_i,\ p = nil \\ bot & \text{otherwise.} \end{cases}$$

Importantly, primitive recursion on lists is given by an operator (rather than just by a unique existence axiom), so that the above definition can be expressed as a term defining $unfold\ \alpha$ and indeed *unfold* as a function.

While the above seems rather entangled in the specificities of HasCasl, the arguments are in fact general enough to work in any quasitopos, or indeed any pccc with equality (cf. Sect. 2). We thus obtain results of independent interest stating that a quasitopos supports certain datatypes and process types, thus partially generalizing known results for W-types in toposes (e.g. [8]). It should be noted that the definition of initial algebras given in Sect. 4 in fact relates to *internal* initial algebras, where initiality is defined by an internal universal quantifier and moreover embodied as an explicit fold operation; this requirement is stronger than the external definition of initial algebras phrased in terms of the existence of algebra morphisms in a category. An *internal natural numbers object (NNO)* in this sense is not strictly needed in the construction of initial datatypes given above, but ensures that the datatypes constructed are, in turn, internal initial algebras. For the construction of final process types, however, the requirement that the NNO is internal is crucial. We record explicitly

Theorem 7. *Let* $\mathbf{C}$ *be a quasitopos.*

(a) If $\mathbf{C}$ *has a NNO, then* $\mathbf{C}$ *has initial algebras for functors* T *of the form* $TX = \sum_{i=1}^{n} A_i \times X^{k_i}$ *(with* $k_i \in \mathbb{N}$ *and constant parameter objects* A_i*).*

(b) *If* $\mathbf{C}$ *has an internal NNO, then* $\mathbf{C}$ *has internal initial algebras for functors* T *of the form* $TX = \sum_{i=1}^{n} A_i \times X^{k_i}$.
(c) *If* $\mathbf{C}$ *has an internal NNO, then* $\mathbf{C}$ *has internal final coalgebras for functors* T *of the form* $TX = \sum_{i=1}^{n}(B_i \to A_i \times X^{k_i})$.

Example 8. In every topological universe (i.e. well-fibred topological quasitopos over **Set** [1]) $\mathbf{C}$, the set $\mathbb{N}$ equipped with the discrete structure is an internal NNO: we have to show that the fold map $A\times(A \to A)\times\mathbb{N} \to A$, $(x, f, n) \mapsto f^n(x)$ is a $\mathbf{C}$-morphism. As $\mathbb{N}$ is discrete, it suffices to show that the map $(x, f) \mapsto f^n(x)$ is a morphism for every n, which holds in any cartesian closed category.

7 Conclusion

We have laid out how initial datatypes and final process types are incorporated into HasCasl, and we have established the existence of such types for a broad class of signature formats. The main contribution in the latter respect is the avoidance of the unique choice principle, which means that, on a more abstract level, our constructions work in any quasitopos.

It remains to be seen whether the datatype constructions can be adapted to more general classes of signatures, in particular to datatype signatures with infinite branching. Support for datatypes with polynomial signatures is already implemented in Hets; support for more complex signatures, intertwined with HasCasl's type class mechanism, is forthcoming.

References

[1] Adámek, J., Herrlich, H., Strecker, G.E.: Abstract and Concrete Categories. Wiley Interscience, Chichester (1990)
[2] Berghofer, S., Wenzel, M.: Inductive datatypes in HOL - lessons learned in formal-logic engineering. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs 1999. LNCS, vol. 1690, pp. 19–36. Springer, Heidelberg (1999)
[3] Bidoit, M., Mosses, P.D. (eds.): CASL User Manual. LNCS, vol. 2900. Springer, Heidelberg (2004)
[4] Birkedal, L., Møgelberg, R.E.: Categorical models for Abadi and Plotkin's logic for parametricity. Math. Struct. Comput. Sci. 15 (2005)
[5] Gunter, E.L.: A broader class of trees for recursive type definitions for HOL. In: Joyce, J.J., Seger, C.-J.H. (eds.) HUG 1993. LNCS, vol. 780, pp. 141–154. Springer, Heidelberg (1994)
[6] Kiselyov, O., Lämmel, R., Schupke, K.: Strongly typed heterogeneous collections. In: Nilsson, H. (ed.) Haskell Workshop, pp. 96–107. ACM Press, New York (2004)
[7] Lambek, J., Scott, P.J.: Introduction to Higher-Order Categorical Logic. Cambridge (1986)
[8] Moerdijk, I., Palmgren, E.: Wellfounded trees in categories. Ann. Pure Appl. Logic 104, 189–218 (2000)
[9] Moggi, E.: Categories of partial morphisms and the λ_p-calculus. In: Poigné, A., Pitt, D.H., Rydeheard, D.E., Abramsky, S. (eds.) Category Theory and Computer Programming. LNCS, vol. 240, pp. 242–251. Springer, Heidelberg (1986)

[10] Mossakowski, T., Maeder, C., Lüttich, K.: The Heterogeneous Tool Set. In: Grumberg, O., Huth, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007)
[11] Mossakowski, T., Schröder, L., Roggenbach, M., Reichel, H.: Algebraic-coalgebraic specification in CoCASL. J. Logic Algebraic Programming 67, 146–197 (2006)
[12] Mosses, P.D. (ed.): CASL Reference Manual. LNCS, vol. 2960. Springer, Heidelberg (2004)
[13] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
[14] Paulson, L.C.: Mechanizing coinduction and corecursion in higher-order logic. J. Log. Comput. 7, 175–204 (1997)
[15] Phoa, W.: An introduction to fibrations, topos theory, the effective topos and modest sets. Research report ECS-LFCS-92-208, Lab. for Foundations of Computer Science, University of Edinburgh (1992)
[16] Rosolini, G., Streicher, T.: Comparing models of higher type computation. In: Realizability Semantics and Applications. In: Birkdedal, L., van Oosten, J., Rosolini, G., Scott, D.S. (eds.) ENTCS, vol. 23 (1999)
[17] Schröder, L.: The logic of the partial λ-calculus with equality. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 385–399. Springer, Heidelberg (2004)
[18] Schröder, L.: The HasCASL prologue - categorical syntax and semantics of the partial λ-calculus. Theoret. Comput. Sci. 353, 1–25 (2006)
[19] Schröder, L., Mossakowski, T.: HasCASL: Towards integrated specification and development of functional programs. In: Kirchner, H., Ringeissen, C. (eds.) AMAST 2002. LNCS, vol. 2422, pp. 99–116. Springer, Heidelberg (2002)
[20] Schröder, L., Mossakowski, T.: Monad-independent Hoare logic in HasCASL. In: Pezzè, M. (ed.) FASE 2003. LNCS, vol. 2621, pp. 261–277. Springer, Heidelberg (2003)
[21] Schröder, L., Mossakowski, T.: Generic exception handling and the Java monad. In: Rattray, C., Maharaj, S., Shankland, C. (eds.) AMAST 2004. LNCS, vol. 3116, pp. 443–459. Springer, Heidelberg (2004)
[22] Schröder, L., Mossakowski, T.: Monad-independent dynamic logic in HasCASL. J. Logic Comput. 14, 571–619 (2004)
[23] Schröder, L., Mossakowski, T., Lüth, C.: Type class polymorphism in an institutional framework. In: Fiadeiro, J.L., Mosses, P.D., Orejas, F. (eds.) WADT 2004. LNCS, vol. 3423, pp. 234–248. Springer, Heidelberg (2005)
[24] Schröder, L., Mossakowski, T., Maeder, C.: HasCASL – Integrated functional specification and programming. Language summary. Available under http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/HasCASL
[25] Walter, D., Schröder, L., Mossakowski, T.: Parametrized exceptions. In: Fiadeiro, J.L., Harman, N., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 424–438. Springer, Heidelberg (2005)

Author Index

Aceto, Luca 65, 80
Aiguier, Marc 356
Alexander, Scott 96
Aspinall, David 111

Baldan, Paolo 126
Bezhanishvili, Nick 143
Bloom, Stephen L. 1

Caires, Luís 16
Cîrstea, Corina 158
Clavel, Manuel 173
Corradini, Andrea 126

Droste, Manfred 179
Durán, Francisco 173

Ehrig, Hartmut 126
Ésik, Zoltán 1

Fiadeiro, José Luiz 194
Fokkink, Wan 65

Gadducci, Fabio 209
Ghani, Neil 226
Glausch, Andreas 242
Glimming, Johan 257

Hansen, Helle Hvid 279
Harman, Neal A. 294
Heckel, Reiko 126
Hendrix, Joe 173
Hoffman, Piotr 111

Ingólfsdóttir, Anna 65, 80

Johnstone, Peter T. 312

König, Barbara 36, 126
Kozen, Dexter 327
Kupke, Clemens 279
Kurz, Alexander 143, 226, 342

Lack, Stephen 312
Lluch Lafuente, Alberto 209
Longuet, Delphine 356
Lucanu, Dorel 372
Lucas, Salvador 173

Mardare, Radu 379
Meseguer, José 173
Mousavi, MohammadReza 80

Ölveczky, Peter 173

Pacuit, Eric 279
Palmigiano, Alessandra 394
Petria, Marius 409

Reisig, Wolfgang 242
Rosický, Jiří 342
Roşu, Grigore 372
Ruozzi, Nicholas 327
Rutten, J.J.M.M. 425

Sadrzadeh, Mehrnoosh 158
Schmitt, Vincent 194
Schröder, Lutz 447
Sobociński, Paweł 312

Venema, Yde 394

Winskel, Glynn 40

Zhang, Guo-Qiang 179