

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

Daniel Leivant Ruy de Queiroz (Eds.)

Logic, Language, Information and Computation

14th International Workshop, WoLLIC 2007
Rio de Janeiro, Brazil, July 2-5, 2007
Proceedings

Volume Editors

Daniel Leivant
Indiana University
Computer Science Department
Bloomington, IN 47405, USA
E-mail: leivant@cs.indiana.edu

Ruy de Queiroz
Universidade Federal de Pernambuco
Centro de Informatica
Av Prof. Luis Freire, s/n Cidade Universitaria, 50740-540 Recife, PE, Brasil
E-mail: ruy@cin.ufpe.br

Library of Congress Control Number: 2007929582

CR Subject Classification (1998): F.1, F.2.1-2, F.4.1, G.1.0, I.2.6

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

ISSN	0302-9743
ISBN-10	3-540-73443-0 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-73443-7 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: 12086542 06/3180 5 4 3 2 1 0

Preface

Welcome to the proceedings of the 14th WoLLIC meeting, which was held in Rio de Janeiro, Brazil, July 2 - 5, 2007. The Workshop on Logic, Language, Information and Computation (WoLLIC) is an annual international forum on inter-disciplinary research involving formal logic, computing and programming theory, and natural language and reasoning. The WoLLIC meetings alternate between Brazil (and Latin America) and other countries, with the aim of fostering interest in applied logic among Latin American scientists and students, and facilitating their interaction with the international applied logic community.

WoLLIC 2007 focused on foundations of computing and programming, novel computation models and paradigms, broad notions of proof and belief, formal methods in software and hardware development; logical approaches to natural language and reasoning; logics of programs, actions and resources; foundational aspects of information organization, search, flow, sharing, and protection. The Program Committee for this meeting, consisting of the 28 colleagues listed here, was designed to promote these inter-disciplinary and cross-disciplinary topics.

Like its predecessors, WoLLIC 2007 included invited talks and tutorials as well as contributed papers. The Program Committee received 52 complete submissions (aside from 15 preliminary abstracts which did not materialize). A thorough review process by the Program Committee, assisted by over 70 external reviewers, led to the acceptance of 21 papers for presentation at the meeting and inclusion in these proceedings. The conference program also included 16 talks and tutorials by 10 prominent invited speakers, who graciously accepted the Program Committee's invitation.

We are sincerely grateful to the many people who enabled the meeting and made it a success: the Program Committee members, for their dedication and hard work over extended periods; the invited speakers, for their time and efforts; the contributors, for submitting excellent work and laboriously refining their papers for the proceedings; the external reviewers, who graciously agreed to selflessly comment on submitted papers, often providing detailed and highly valuable feedback; and the Organizing Committee, whose work over many months made the meeting possible in the first place.

On behalf of the entire WoLLIC community, we would also like to express our gratitude to our institutional sponsors and supporters. WoLLIC 2007 was sponsored by the Association for Symbolic Logic (ASL), the Interest Group in Pure and Applied Logics (IGPL), the European Association for Logic, Language and Information (FoLLI), the European Association for Theoretical Computer Science (EATCS), the Sociedade Brasileira de Computacao (SBC), and the Sociedade Brasileira de Logica (SBL). Generous financial support was provided by the Brazilian government (through CAPES, grant PAEP-0755/06-0), Univ. Fed. Fluminense (UFF), and SBC.

May 2007

Daniel Leivant
(Program Chair)
Ruy de Queiroz
(General Chair)

Organization

Program Committee

Samson Abramsky (Oxford)
Michael Benedikt (Bell Labs and Oxford)
Lars Birkedal (ITU Copenhagen)
Andreas Blass (University of Michigan)
Thierry Coquand (Chalmers University, Göteborg)
Jan van Eijck (CWI, Amsterdam)
Marcelo Finger (University of Sao Paulo)
Rob Goldblatt (Victoria University, Wellington)
Yuri Gurevich (Microsoft Redmond)
Hermann Haeusler (PUC Rio)
Masami Hagiya (Tokyo University)
Joseph Halpern (Cornell)
John Harrison (Intel UK)
Wilfrid Hodges (University of London/QMW)
Phokion Kolaitis (IBM San Jose)
Marta Kwiatkowska (University of Birmingham and Oxford Univeristy)
Daniel Leivant (Indiana University) (Chair)
Maurizio Lenzerini (University of Rome)
Jean-Yves Marion (LORIA Nancy)
Dale Miller (Polytechnique, Paris)
John Mitchell (Stanford University)
Lawrence Moss (Indiana University)
Peter O'Hearn (University of London/QM)
Prakash Panangaden (McGill, Montreal)
Christine Paulin-Mohring (Paris-Sud, Orsay)
Alexander Razborov (Moscow University)
Helmut Schwichtenberg (Munich Uuniversity)
Jouko Vaananen (University of Helsinki)

Organizing Committee

Renata P. de Freitas (Universidade Fed Fluminense)
Ana Teresa Martins (Universidade Fed Ceara')
Anjolina de Oliveira (Universidade Fed Pernambuco)
Ruy de Queiroz (Universidade Fed Pernambuco, Co-chair)
Marcelo da Silva Correa (Universidade Fed Fluminense)
Petrucio Viana (Universidade Fed Fluminense, Co-chair)

External Reviewers

Anonymous (2)
Josep Argelich-Roma
Joseph Barone
Lev Beklemishev
Johan van Benthem
Mauro Birattari
Guillaume Bonfante
Anne Broadbent
Anuj Dawar
Luc DeRaedt
Ralph Debusmann
Jules Desharnais
Hans van Ditmarsch
Melvin Fitting
Matthias Fruth
Zhaohui Fu
Viktor Galliard
Michael Gasser
Giorgio Ghelli
Rajeev Gore
Cosimo Guido

Anil Gupta
Javier Gutierrez-Garcia
Masami Hagiya
Shan He
Andreas Herzig
Liang Huang
Etienne Kerre
Evelina Lamma
Chumin Li
Thorsten Liebig
Paolo Mancarella
Maarten Marx
Pascal Matsakis
Susan McRoy
Bernhard Moeller
Andrzej Murawski
Reinhard Muskens
Sara Negri
Joakim Nivre
Jeff Paris
Witold Pedrycz

Mario Porrmann
Anne Preller
Robert van Rooij
Andrzej Roslanowski
Mehrnoosh Sadrzadeh
Ulrik Schroeder
Libin Shen
Yun Shi
Guillermo Simari
Nicholas Smith
Yoshinori Tanabe
Paolo Torroni
Paul Vitanyi
Christoph Weidenbach
Laurent Wendling
Matthew Williams
Yehezkel Yeshurun
Chen Yu
Chi Zheru

Table of Contents

A Grammatical Representation of Visibly Pushdown Languages	1
<i>Joachim Baran and Howard Barringer</i>	
Fully Lexicalized Pregroup Grammars	12
<i>Denis Béchet and Annie Foret</i>	
Bounded Lattice T-Norms as an Interval Category	26
<i>Benjamín C. Bedregal, Roberto Callejas-Bedregal, and Hélida S. Santos</i>	
Towards Systematic Analysis of Theorem Provers Search Spaces: First Steps	38
<i>Hicham Bensaid, Ricardo Caferra, and Nicolas Peltier</i>	
Continuation Semantics for Symmetric Categorical Grammar	53
<i>Raffaella Bernardi and Michael Moortgat</i>	
Ehrenfeucht–Fraïssé Games on Linear Orders	72
<i>Ryan Bissell-Siders</i>	
Hybrid Logical Analyses of the Ambient Calculus	83
<i>Thomas Bolander and René Rydhof Hansen</i>	
Structured Anaphora to Quantifier Domains: A Unified Account of Quantificational and Modal Subordination	101
<i>Adrian Brasoveanu</i>	
On Principal Types of BCK- λ -Terms	120
<i>Sabine Broda and Luís Damas</i>	
A Finite-State Functional Grammar Architecture	131
<i>Alexander Dikovsky</i>	
Pregroup Calculus as a Logic Functor	147
<i>Annie Foret</i>	
A Formal Calculus for Informal Equality with Binding	162
<i>Murdoch J. Gabbay and Aad Mathijssen</i>	
Formal Verification of an Optimal Air Traffic Conflict Resolution and Recovery Algorithm	177
<i>André L. Galdino, César Muñoz, and Mauricio Ayala-Rincón</i>	
An Introduction to Context Logic (Invited Paper)	189
<i>Philippa Gardner and Uri Zarfaty</i>	

Numerical Constraints for XML	203
<i>Sven Hartmann and Sebastian Link</i>	
Modules over Monads and Linearity	218
<i>André Hirschowitz and Marco Maggesi</i>	
Hydra Games and Tree Ordinals	238
<i>Ariya Ishihara</i>	
Spin Networks, Quantum Topology and Quantum Computation (Invited Paper)	248
<i>Louis H. Kauffman and Samuel J. Lomonaco Jr.</i>	
Symmetries in Natural Language Syntax and Semantics: The Lambek-Grishin Calculus (Invited Paper)	264
<i>Michael Moortgat</i>	
Computational Interpretations of Classical Linear Logic (Invited Paper)	285
<i>Paulo Oliva</i>	
Autonomous Programmable Biomolecular Devices Using Self-assembled DNA Nanostructures (Invited Paper)	297
<i>John H. Reif and Thomas H. LaBean</i>	
Interval Valued QL-Implications	307
<i>R.H.S. Reiser, G.P. Dimuro, B.C. Bedregal, and R.H.N. Santiago</i>	
Behavioural Differential Equations and Coinduction for Binary Trees . . .	322
<i>Alexandra Silva and Jan Rutten</i>	
A Sketch of a Dynamic Epistemic Semiring	337
<i>Kim Solin</i>	
A Modal Distributive Law (abstract) (Invited Paper)	351
<i>Yde Venema</i>	
Ant Colony Optimization with Adaptive Fitness Function for Satisfiability Testing	352
<i>Marcos Villagra and Benjamín Barán</i>	
Author Index	363

A Grammatical Representation of Visibly Pushdown Languages

Joachim Baran and Howard Barringer

The University of Manchester, School of Computer Science, Manchester, UK
joachim.baran@cs.manchester.ac.uk, howard.barringer@cs.manchester.ac.uk

Abstract. Model-checking regular properties is well established and a powerful verification technique for regular as well as context-free program behaviours. Recently, through the use of ω -visibly pushdown languages (ω VPLs), defined by ω -visibly pushdown automata, model-checking of properties beyond regular expressiveness was made possible and shown to be still decidable even when the program's model of behaviour is an ω VPL. In this paper, we give a grammatical representation of ω VPLs and the corresponding finite word languages – VPL. From a specification viewpoint, the grammatical representation provides a more natural representation than the automata approach.

1 Introduction

In [AM04], ω -visibly pushdown languages over infinite words (ω VPLs) were introduced as a specialisation of ω -context-free languages (ω CFLs), i.e. they are strictly included in the ω CFLs but more expressive than ω -regular languages (ω RLs). The paper showed that the language inclusion problem is decidable for ω VPLs, and thus, the related model-checking problem is decidable as well. This work was presented in the context of (ω) VPLs¹ being represented as automata and a monadic second-order logic with matching relation.

In this paper, we define a grammatical representation of (ω) VPLs. We also propose that grammars allow us to write more natural specifications than (ω) -visibly pushdown automata ((ω) VPA). Section 2 introduces the formalisms used in this paper. In Section 3 our grammatical representation is presented. Finally, Section 4 concludes our work.

2 Preliminaries

For an arbitrary set X , we write 2^X to denote its power-set. Let Σ denote a finite alphabet over letters a, b, c, \dots , the set of finite (infinite) words over Σ is denoted by Σ^* (Σ^ω). We use ε to denote the empty word. For an arbitrary word $w \in \Sigma^*$ we will write $|w|$ to denote its length. For the empty word ε , we set

¹ Our bracketing of ω abbreviates restating the sentence without the bracketed contents.

$|\varepsilon| = 0$. The concatenation of two words w and w' is denoted by $w \cdot w'$. The length of an infinite word equals the first infinite ordinal ω . Positions in a word w are addressed by natural numbers, where the first index starts at 1. The i -th letter of a word is referred to as $w(i)$. We use a sans-serif font for meta-variables and a (meta)-variable's context is only explicitly stated once.

2.1 Visibly Pushdown Languages

(ω) VPLs are defined over a terminal alphabet of three pairwise disjoint sets Σ_c , Σ_i and Σ_r , which we will use as properties in specifications to denote calls, internal actions and returns respectively. Any call may be matched with a subsequent return, while internal actions must not be matched at all. A formalisation of (ω) VPLs has been given in terms of automata as well as in terms of logic.

Visibly Pushdown Automata. For (ω) VPA, the current input letter determines the actions the automaton can perform.

Definition 1. A visibly pushdown automaton over finite words (VPA) (visibly pushdown automaton over infinite words (ω VPA)) is a sextuple $A = (Q, \Sigma_c \cup \Sigma_i \cup \Sigma_r, \Gamma, \delta, Q', F)$, where Q is a finite set of states $\{p, q, q_0, q_1, \dots\}$, $\Sigma_c, \Sigma_i, \Sigma_r$ are finite sets of terminals representing calls c, c_0, c_1, \dots, c_k , internal actions i, i_0, i_1, \dots, i_l , and returns r, r_0, r_1, \dots, r_m respectively, Γ is a finite set of stack symbols A, B, C, \dots , including the stack bottom marker \perp , δ is a finite set of transition rules between states $p, q \in Q$ for inputs $c \in \Sigma_c$, $i \in \Sigma_i$, or $r \in \Sigma_r$ and stack contents $A, B \in (\Gamma \setminus \{\perp\})$ of the form $p \xrightarrow{c, \kappa / B\kappa} q$ for all $\kappa \in \Gamma$, $p \xrightarrow{i, \kappa / \kappa} q$ for all $\kappa \in \Gamma$, $p \xrightarrow{r, \perp / \perp} q$, or $p \xrightarrow{r, A / \varepsilon} q$, $Q' \subseteq Q$ denotes a non-empty set of designated initial states, $F \subseteq Q$ is the set of final states.

When reading a word w , instantaneous descriptions (q, w, α) are used to describe the current state, the current stack contents and a postfix of w that still has to be processed. The binary move relation \vdash_A determines possible moves an (ω) VPA A can make. Whenever A in \vdash_A is understood from the context, we write \vdash . In the following we use \vdash^* (\vdash^ω) in order to denote a finitely (infinitely) repeated application of \vdash (up to the first infinite ordinal). In conjunction with \vdash^ω , we use Q_{inf} to denote the set of states that appear infinitely often in the resulting sequence.

Definition 2. The language $L(A)$ of a (ω) VPA $A = (Q, \Sigma_c \cup \Sigma_i \cup \Sigma_r, \Gamma, \delta, Q', F)$ is the set of finite (infinite) words that are derivable from any initial state in Q' , i.e. $L(A) = \{w \mid (p, w, \perp) \vdash^* (q, \varepsilon, \gamma) \text{ and } p \in Q' \text{ and } q \in F\}$ ($L(A) = \{w \mid (p, w, \perp) \vdash^\omega (q, \varepsilon, \gamma) \text{ and } p \in Q' \text{ and } Q_{inf} \cap F \neq \emptyset\}$).

In an arbitrary word of the ω VPLs, calls and returns can appear either matched or unmatched. A call automatically matches the next following return, which is not matched by a succeeding call. A call is said to be unmatched, when there are less or equally many returns than calls following it. Unmatched calls cannot

be followed by unmatched returns, but unmatched returns may be followed by unmatched calls.

A word w of the form $c\alpha r$ is called *minimally well-matched*, iff c and r are a matching and α contains no unmatched calls or returns. The set of all minimally well-matched words is denoted by L_{mwm} ([LMS04], p. 412, par. 8). In conjunction with a given ω VPA A , a *summary-edge* is a triple (p, q, f) , $f \in \{0, 1\}$, which abstracts minimally well-matched words that are recognised by A when going from p to q , where on the corresponding run a final state has to be visited ($f = 1$) or not ($f = 0$). The set of words represented by a summary edge is denoted by $L((p, q, f))$.

Definition 3. A pseudo-run of an ω VPA $A = (Q, \Sigma_c \cup \Sigma_i \cup \Sigma_r, \Gamma, \delta, Q', F)$ is an infinite word $w = \alpha_1\alpha_2\alpha_3\ldots$ with $\alpha_i \in (\Sigma_c \cup \Sigma_i \cup \Sigma_r \cup \bigcup_{n=1}^m \{\Omega_n\})$, each Ω_n denotes a non-empty set of summary-edges of the form (p, q, f) with $f \in \{0, 1\}$, in case $\alpha_i = c$, then there is no $\alpha_j = r$ for $i < j$, and there is a word $w' = \beta_1\beta_2\beta_3\ldots$, $w' \in L(A)$, so that either $\alpha_i = \beta_i$, or $\alpha_i = \Omega_k$ and β_i is a minimally well-matched word that is generated due to A moving from state p to q and $(p, q, f) \in \Omega_k$. In case $f = 1$ ($f = 0$), then a final state is (not) on the path from p to q .

According to [AM04], p. 210, par. 6, a non-deterministic Büchi-automaton can be constructed that accepts all pseudo-runs of an arbitrary given ω VPA. For every pseudo-run that is represented by the Büchi-automaton, there exists a corresponding accepting run of the original ω VPA.

Monadic Second-Order Logic with Matched Calls/Returns. A logical representation, $(\omega)\text{MSO}_\mu$, of $(\omega)\text{VPLs}$ was given as an extension of monadic second-order logic (MSO) with a matching relation μ , which matches calls and returns, where the call always has to appear first.

Definition 4. A formula φ is a formula of monadic second-order logic of one successor with call/return matching relation $((\omega)\text{MSO}_\mu)$ over an alphabet $\Sigma_c \cup \Sigma_i \cup \Sigma_r$, iff it is of the form $\varphi \equiv \top$, $\varphi \equiv T_a(i)$, $a \in \Sigma$, $\varphi \equiv i \in X$, $\varphi \equiv i \leq j$, $\varphi \equiv \mu(i, j)$, $\varphi \equiv S(i, j)$, $\varphi \equiv \neg\psi$, $\varphi \equiv \psi_1 \vee \psi_2$, $\varphi \equiv \exists i \psi(i)$, or $\varphi \equiv \exists X \psi(X)$, where ψ , ψ_1 , and ψ_2 are $(\omega)\text{MSO}_\mu$ formula as well, V and W are sets of first-order and second-order variables respectively, and $i, j \in V$, $X \in W$.

We use the standard abbreviations for the truth constant, conjunction and universal quantification. Also, $\forall x(y \leq x \leq z \Rightarrow \varphi)$ is shortened to $\forall x \in [y, z] \varphi$. In order to simplify arithmetic in conjunction with the successor function, we will omit the successor function completely in the following and write $i + 1$ instead of j for which $S(i, j)$ holds. Second-order quantifications $\exists X_1 \exists X_2 \ldots \exists X_k$ are abbreviated in vector notation as $\exists \mathbf{X}$.

We assume the usual semantics for $(\omega)\text{MSO}_\mu$ formulae, where $\mu(i, j)$ is true when $w(i) = c$ and $w(j) = r$ are a matching call/return pair.

Definition 5. The language $\mathcal{L}(\varphi)$ of an $(\omega)\text{MSO}_\mu$ formula φ is the set of finite (infinite) words w for which there is a corresponding model of φ .

2.2 Context-Free Grammars

Definition 6. An (ω) -context-free grammar $((\omega)\text{CFG}) G$ over finite words (infinite words) is a quadruple (V, Σ, P, S) (quintuple (V, Σ, P, S, F)), where V is a finite set of non-terminals A, B, \dots , Σ is a finite set of terminals a, b, \dots , V and Σ are disjoint, P is a finite set of productions of the form $V \times (V \cup \Sigma)^*$, and S denotes a designated starting non-terminal $S \in V$ (and $F \subseteq V$ denotes the set of accepting non-terminals).

We will use the notation $A \rightarrow_G \alpha$ for a production (A, α) in G . If G is understood from the context, we write $A \rightarrow \alpha$. We also use \rightarrow_G to denote the derivation relation of G , that determines derivations of sentential forms of G . Again, we drop the sub-script when G is understood from the context. In the following we write $\xrightarrow{*}$ in order to denote a finitely repeated application of \rightarrow while $\xrightarrow{\omega}$ denotes an infinite application of \rightarrow . Similarly to the previously used set Q_{inf} , we use V_{inf} in connection with $\xrightarrow{\omega}$ in order to denote the set of non-terminals that are infinitely often replaced among the sentential forms.

Definition 7. The language $\mathcal{L}(G)$ of an $(\omega)\text{CFG}$ $G = (V, \Sigma, P, S)$ ($G = (V, \Sigma, P, S, F)$) is the set of finite (infinite) words over Σ that are derivable from the initial symbol, i.e. $\mathcal{L}(G) = \{w \mid S \xrightarrow{*} w \text{ and } w \in \Sigma^*\}$ ($\mathcal{L}(G) = \{w \mid S \xrightarrow{\omega} w, w \in \Sigma^\omega \text{ and } V_{inf} \cap F \neq \emptyset\}$).

2.3 Balanced Grammars

Balanced grammars are a specialisation of context-free grammars over finite words [BB02]. Unlike the previous definition of CFGs, balanced grammars are permitted to have an infinite set of productions. This is due to regular expressions over terminals and/or non-terminals in right-hand sides of productions.

Definition 8. A balanced grammar (BG) G over finite words is a quadruple $(V, \underline{\Sigma} \cup \Sigma \cup \overline{\Sigma}, \mathcal{P}, S)$ is a specialisation of a context-free grammar, where $\underline{\Sigma}$ and $\overline{\Sigma}$ are finite sets of terminals $\underline{a}_1, \underline{a}_2, \dots, \underline{a}_k$ and co-terminals $\overline{a}_1, \overline{a}_2, \dots, \overline{a}_k$ respectively, where each terminal \underline{a}_i is associated with its unique counterpart, \overline{a}_i , its co-terminal, and vice versa, Σ is a finite set of intermediate terminals a, b, \dots , the sets $\underline{\Sigma}$, $\overline{\Sigma}$, and Σ are mutually disjoint, \mathcal{P} is a finite or infinite set of productions of the form $V \times \underline{\Sigma}(V \cup \Sigma)^*\overline{\Sigma}$, and S denotes a designated starting non-terminal $S \in V$.

As already pointed out in [BB02], an infinite set of productions does not raise the grammars' expressiveness, but provides a succinct notation. The derivation relation of context-free grammars is still applicable to balanced grammars.

Definition 9. The language $\mathcal{L}(G)$ of a BG $G = (V, \underline{\Sigma} \cup \Sigma \cup \overline{\Sigma}, \mathcal{P}, S)$ is the set of words that are derivable from the initial symbol, i.e. $\mathcal{L}(G) = \{w \mid S \xrightarrow{*} w \text{ and } w \in (\underline{\Sigma} \cup \Sigma \cup \overline{\Sigma})^*\}$.

In the following, we are writing \mathcal{R} to denote an arbitrary regular expression over $V \cup \Sigma$.

3 Grammars for Visibly Pushdown Languages

A grammatical representation of (ω) VPLs is presented, where we take a compositional approach that builds on pseudo-runs and minimally well-matched words. We first state our grammatical representation and then decompose it into two types of grammars. We show their resemblance of pseudo-runs and minimally well-matched words, similar to the approach for (ω) VPAs.

3.1 Quasi Balanced Grammars

In order to simplify our proofs, we give an alternative – but expressively equivalent – definition of BGs, where only a finite number of productions is admitted. We reformulate occurrences of regular expressions \mathcal{R} in terms of production rules $P_{\mathcal{R}}$ and substitute each \mathcal{R} by an initial non-terminal $S_{\mathcal{R}}$ that appears on a left-hand side in $P_{\mathcal{R}}$. Therefore, matchings $\underline{a}\mathcal{R}\bar{a}$ become $\underline{a}S_{\mathcal{R}}\bar{a}$, where the derivation of $S_{\mathcal{R}}$ resembles $L(\mathcal{R})$.

Definition 10. Let $G = (V, \underline{\Sigma} \cup \Sigma \cup \overline{\Sigma}, \mathcal{P}, S)$ denote an arbitrary BG, a quasi balanced grammar (qBG) $G' = (V', \underline{\Sigma} \cup \Sigma \cup \overline{\Sigma}, P, S)$ generalises G by having a finite set of productions, where productions are either

- a) in double Greibach normal form $A \rightarrow \underline{a}S_{\mathcal{R}}\bar{a}$, or
- b) of form $A \rightarrow BC$, $A \rightarrow aC$, or $A \rightarrow \varepsilon$, where B 's productions are of the form according to a) and C 's productions are of the form according to b).

Lemma 1. For every BG $G = (V, \underline{\Sigma} \cup \Sigma \cup \overline{\Sigma}, \mathcal{P}, S)$ there is a qBG $G' = (V', \underline{\Sigma} \cup \Sigma \cup \overline{\Sigma}, P, S)$, such that $L(G) = L(G')$.

3.2 A Grammatical Representation of ω VPLs

Matchings in an ω VPL appear only as finite sub-words in the language, which was utilised in the characterisation of pseudo-runs. Summary-edges reflect sub-words of exactly this form, which are in L_{mwm} . Given an infinite word w , it can be split into sub-words that are either in L_{mwm} or in $\Sigma_c \cup \Sigma_i \cup \Sigma_r$, where no sub-word in Σ_r follows a sub-word in Σ_c . We abbreviate the latter constraint as Σ_c/Σ_r -matching avoiding. Our grammatical representation of ω VPLs utilises Σ_c/Σ_r -matching avoiding ω RGs to describe languages of pseudo-runs. Languages of summary-edges, i.e. languages with words in L_{mwm} , are separately described by qBGs under a special homomorphism. The homomorphism is required to cover matchings of calls c that can match more than one return r , which cannot be reflected as a simple terminal/co-terminal matching \underline{a}/\bar{a} . For example, the matchings c/r_1 and c/r_2 are representable as terminal/co-terminal pairs \underline{a}/\bar{a} and \underline{b}/\bar{b} under the mappings $h(\underline{a}) = h(\underline{b}) = c$, $h(\bar{a}) = r_1$ and $h(\bar{b}) = r_2$. Finally, the amalgamation of Σ_c/Σ_r -matching avoiding ω RGs and qBGs under the aforementioned homomorphism give us a grammatical representation of ω VPLs:

Definition 11. A superficial² ω -regular grammar with injected balanced grammars ($\omega\text{RG}(\text{qBG})+h$) $G = (V, \Sigma_c \cup \Sigma_i \cup \Sigma_r \cup \bigcup_{n=1}^m \{g_n\}, P, S, F, \bigcup_{n=1}^m \{G_n\}, h)$, where

- $\Sigma_c, \Sigma_i, \Sigma_r$ and $\bigcup_{n=1}^m \{g_n\}$ are mutually disjoint,
- G is Σ_c/Σ_r -matching avoiding,
- $G_n = (V_n, \Sigma_n, P_n, S_n)$ is a qBG for $n = 1, 2, \dots, m$,³

is an ωCFG $G' = (V \cup \bigcup_{n=1}^m \{V_n\}, \Sigma \cup \bigcup_{n=1}^m \{\Sigma_n\}, P', S, F)$ with

- disjoint sets V and $\{V_1, V_2, \dots, V_m\}$ as well as Σ and $\{\Sigma_1, \Sigma_2, \dots, \Sigma_m\}$, and
- P' is the smallest set satisfying
 - $A \rightarrow_{G'} aB$ if $A \rightarrow_G aB$, where $a \in (\Sigma_c \cup \Sigma_i \cup \Sigma_r)$, or
 - $A \rightarrow_{G'} S_n B$ if $A \rightarrow_G g_n B$, or
 - $A \rightarrow_{G'} \alpha$ if $A \rightarrow_{G_n} \alpha$,

and h is constrained so that it preserves terminals of the injector grammar, $h(a) = a$ for any $a \in (\Sigma_c \cup \Sigma_i \cup \Sigma_r)$, and for terminals/co-terminals of injected grammars it maps terminals $\underline{a} \in \underline{\Sigma}_n$ to calls $c \in \Sigma_c$, maps co-terminals $\bar{a} \in \bar{\Sigma}_n$ to returns $r \in \Sigma_r$, maps terminals $a \in \Sigma_n$ to internal actions $i \in \Sigma_i$.

In the following, we refer to the homomorphism h under the constraints which are given above as *superficial mapping* h .

Definition 12. The language $\mathcal{L}(G)$ of an $\omega\text{RG}(\text{qBG})+h$ $G = (V, \Sigma_c \cup \Sigma_i \cup \Sigma_r \cup \bigcup_{n=1}^m \{g_n\}, P, S, F, \bigcup_{n=1}^m \{G_n\}, h)$ denotes the set $\{h(w) \mid S \xrightarrow{G'} w \text{ and } w \in (\Sigma \cup \Sigma_1 \cup \Sigma_2 \cup \dots \cup \Sigma_m)^\omega\}$, where G' is the ω -context-free grammar corresponding to G .

Consider an arbitrary $\omega\text{RG}(\text{qBG})+h$ $G = (V, \Sigma_c \cup \Sigma_i \cup \Sigma_r \cup \bigcup_{n=1}^m \{g_n\}, P, S, F, \bigcup_{n=1}^m \{G_n\}, h)$. We call the ωRG $G_\uparrow = (V, \Sigma_c \cup \Sigma_i \cup \Sigma_r \cup \bigcup_{n=1}^m \{g_n\}, P, S, F)$ the *injector grammar* of G , while the qBGs G_1, G_2, \dots, G_m are called *injected grammars* of G .⁴ When G is clear from the context, we just talk about the injector grammar G_\uparrow and the injected grammars G_1, \dots, G_m respectively. The languages associated with these grammars are referred to as injector and injected languages respectively. In fact, injector languages resemble pseudo-runs with pseudo edges $g_n, n = 1 \dots m$, while injected language resemble matchings covered by summary-edges.

3.3 ωVPL and $\omega\text{RL}(\text{qBL})+h$ Coincide

For the equivalence proof of ωVPL s and $\omega\text{RL}(\text{qBL})+h$ s, we first show that minimally well-matched words, as described by summary-edges, can be expressed by

² Superficial – as understood as being on the surface of something.

³ Each Σ_n is a shorthand for $\underline{\Sigma}_n \cup \Sigma_n \cup \bar{\Sigma}_n$.

⁴ This should not be confused with nested words, [AM06], which describe the structure induced by matchings in finite words.

qBGs plus an appropriate superficial mapping, and vice versa. It is then straightforward to prove language equivalence, by translating an arbitrary $\omega\text{RG}(\text{qBG})+h$ into an expressively equivalent ωMSO formula and an arbitrary ωVPA into an expressively equivalent $\omega\text{RG}(\text{qBG})+h$.

Let VPA_{mwm} and MSO_{mwm} refer to VPA and MSO-formulae whose languages are subsets of L_{mwm} respectively, i.e. restricted variants of VPA and MSO-formulae that only accept minimally well-matched words. We show that any qBG can be translated into an equivalent MSO_{mwm} formula. Since MSO_{mwm} defines L_{mwm} , the inclusion $\text{qBL} \subseteq L_{\text{mwm}}$ is proven. Second, for an arbitrary VPA_{mwm} a qBG is constructed so that their languages coincide, which gives us $\text{qBL} \supseteq \text{VPL}_{\text{mwm}}$.

In the following lemma, an MSO_{mwm} formula is constructed from a qBG in such a way so that their languages coincide. The translation works by quantifying over each matching in the word and filling in the respective regular expressions. In order for the lemma to hold, we assume that all of the qBG's productions are uniquely identified by their terminal/co-terminal pairs. While this clearly restricts a grammar's language in general, the language under a superficial mapping is preserved.

Lemma 2. *Let $G = (V, \underline{\Sigma} \cup \Sigma \cup \overline{\Sigma}, P, S)$ denote an arbitrary qBG and let h be an arbitrary superficial mapping. Then the MSO_{mwm} formula*

$$\varphi \equiv \exists X \exists i \bigvee_{(S \rightarrow \underline{a} S_{\mathcal{R}} \overline{a}) \in P} (\Psi_{\underline{a}, \overline{a}}(1, i) \wedge T_{\S}(i+1) \wedge \forall k \in [1, i] \Phi(k))$$

accepts the same language as G , where

$$\Psi_{\underline{a}, \overline{a}}(i, j) \equiv T_{\underline{a}}(i) \wedge T_{\overline{a}}(j) \wedge \mu(i, j),$$

$$\Phi(k) \equiv \bigvee_{\underline{a} \in \underline{\Sigma}} T_{\underline{a}}(k) \Rightarrow \exists j (\mu(k, j) \wedge \varphi_{\underline{a}}(k+1, j)),$$

$$\Delta(s, t, k) \equiv \bigvee_{(A \rightarrow \underline{a} B) \in P_{\mathcal{R}}} (X_A(k) \wedge X_B(k+1) \wedge T_{\underline{a}}(k)) \vee \bigvee_{(A \rightarrow BC, B \rightarrow \underline{b} S_{\mathcal{R}} \overline{b}) \in P_{\mathcal{R}}} \exists j \in [s, t] (X_A(k) \wedge X_C(j+1) \wedge \Psi_{\underline{b}, \overline{b}}(k, j))$$

$$\varphi_{\underline{a}}(s, t) \equiv X_{S_{\mathcal{R}}}(s) \wedge \bigwedge_{(A, B) \in V, A \neq B} \forall k \neg (X_A(k) \wedge X_B(k)) \wedge \forall k \in [s, t-1] \varphi_{\overline{\mu}}(s, t, k) \Rightarrow \Delta(s, t, k) \wedge \bigvee_{(A \rightarrow \varepsilon) \in P_{\mathcal{R}}} X_A(t),$$

where $(A \rightarrow \underline{a} S_{\mathcal{R}} \overline{a}) \in P$, and $\varphi_{\overline{\mu}}(s, t, k) \equiv \neg \exists i, j \in [s, t] (\mu(i, j) \wedge i < k \leq j)$.

Proof. Consider an arbitrary $\text{qBG}+h$ G and its translation to a MSO_{mwm} formula φ . We show that every word $w\$ \in L(\varphi)$ is a word $w \in L(G)$ and vice versa.

$L(\varphi) \subseteq L(G)$: Let \mathcal{M} be an arbitrary model of φ that represents the word w . We write $\langle T_1, X_1 \rangle \langle T_2, X_2 \rangle \dots \langle T_{|w|}, X_{|w|} \rangle \langle T_{\S}, X_{|w|+1} \rangle$ to denote the sequence of unique predicate pairs of T and X which hold at indices 1 to $|w|+1$ in \mathcal{M} .

Occurrences of the form $\langle T_{\overline{a}}, X_B \rangle$ are replaced by $\langle T_{\overline{a}}, B \rangle$ if $(B \rightarrow \varepsilon) \in P$, occurrences of the form $\langle T_{\underline{a}}, X_A \rangle \langle T_{\overline{a}}, B \rangle$ are replaced by $\langle T_{\overline{a}}, A \rangle$ if $(A \rightarrow \underline{a} B) \in P$,

and occurrences of the form $\langle T_{\underline{a}}, X_B \rangle \langle T_{\bar{a}}, S_{\mathcal{R}} \rangle \langle T_{\bar{b}}, C \rangle$ are replaced with $\langle T_{\bar{b}}, A \rangle$ if $(A \rightarrow BC, B \rightarrow \underline{a}S_{\mathcal{R}}\bar{a}) \in P$. Eventually, $\langle T_{\underline{a}}, X_S \rangle \langle T_{\bar{a}}, S_{\mathcal{R}} \rangle$ will be left, which is replaced with S iff $(S \rightarrow \underline{a}S_{\mathcal{R}}\bar{a}) \in P$. As a result, we have established a bottom up parse in G for an arbitrary word $w\$ \in L(\varphi)$, which implies that every word in $L(\varphi)$ is in $L(G)$.

$L(G) \subseteq L(\varphi)$: Let $S \rightarrow \alpha \rightarrow \beta \rightarrow \dots \rightarrow w$ denote an arbitrary derivation in G . With each derivation step, we try to find variable assignments that satisfy φ , so that after the derivation finishes, $w\$$ is represented by the sequence $\langle T_1, X_1 \rangle \langle T_2, X_2 \rangle \dots \langle T_{\$}, X_{|w|+1} \rangle$ of φ . The construction of $\langle T_1, X_1 \rangle \langle T_2, X_2 \rangle \dots \langle T_{\$}, X_{|w|+1} \rangle$ follows the derivation $S \rightarrow \alpha \rightarrow \beta \rightarrow \dots \rightarrow w$ in the sense that there is a mapping between the n -th step of the sequence constructed by the variable assignments and the n -th sentential form reached in the derivation.

We consider triples of the form $\langle A, \psi, B \rangle$, where A is a non-terminal as it appears in some sentential form derived from S , ψ denotes the formula which is supposed to derive a word w , where $A \xrightarrow{*} w$, and B is a temporary stored non-terminal. When A derives α with $A \rightarrow \alpha$, we try to find variable assignments for ψ that represent terminals in α . Since terminals appear only in prefixes/postfixes of α , we remove the ground terms in ψ , add pairs $\langle T_k, X_k \rangle$ to the left/right of $\langle A, \psi, B \rangle$ accordingly, and replace $\langle A, \psi, B \rangle$ with $\langle C, \psi', E \rangle$ or the sequence $\langle C, \psi', E \rangle \langle D, \psi'', F \rangle$, depending if α has one non-terminal C or two non-terminals CD as sub-word. The non-terminals E and F are associated with productions $E \rightarrow \varepsilon$ and $F \rightarrow \varepsilon$ respectively, where they denote the end of a regular expression embedded between a call and return.

Starting the rewriting process with $\langle S, \varphi, A \rangle$, A is chosen arbitrarily, a sequence of tuples of the form $\langle T_k, X_k \rangle$ is eventually left, which indeed represents w , so that the model for $w\$$ is represented by adding $\langle T_{\$}, X_A \rangle$ to the sequence. \square

The reverse inclusion, i.e. $\text{qBL} \supseteq \text{VPL}$, can be shown by a number of rewriting steps of an arbitrary VPA_{mwm} to a BG equipped with a superficial mapping. Since there is a translation from BGs to qBGs, the inclusion is then proven. The VPA_{mwm} represents hereby $L((p, q, f))$ of some summary-edge (p, q, f) .

Definition 13. Let $G = (V, \Sigma_c \cup \Sigma_i \cup \Sigma_r, P, S)$ denote the CFG with productions of the form $S \rightarrow cA$, $A \rightarrow cBC$, $A \rightarrow iB$, and $A \rightarrow r$ that is obtained from a VPA_{mwm} by the standard translation [HMU01, Theorem 6.14, incl. its proof], then the immediate matching CFG $G' = (V', \Sigma_c \cup \Sigma_i \cup \Sigma_r, P', S')$ is obtained from G , so that $S' \rightarrow_{G'} c\langle A, r \rangle r$ iff $S \rightarrow_G cA$, $\langle A, r_1 \rangle \rightarrow_{G'} c\langle B, r_2 \rangle r_2 \langle C, r_1 \rangle$ iff $A \rightarrow_G cBC$, $\langle A, r \rangle \rightarrow_{G'} i\langle B, r \rangle$ iff $A \rightarrow_G iB$, $\langle A, r \rangle \rightarrow_{G'} \varepsilon$ iff $A \rightarrow_G r$.

Lemma 3. The language $L(G)$ of an immediate matching CFG G that is obtained from a VPA_{mwm} A is equal to $L(A)$.

Proof (Lemma 3). The translation of Definition 13 is preserving the language equivalence of the grammars, as it is a special case of the more general translation presented in [[Eng92, Page 292]]. \square

In the following transformation steps, productions are rewritten so that matchings cAr appear exclusively in right-hand sides. Furthermore, we remove all productions that produce no matchings by introducing language preserving regular

expressions \mathcal{R} in productions with right-hand sides of the form $c\mathcal{A}r$, so that the resulting right-hand side is $c\mathcal{R}r$. Finally, adding a homomorphism that maps fresh terminal/co-terminal pairs to calls and returns, where the productions are modified accordingly, gives us a BG.

Definition 14. Let $G = (V, \Sigma_c \cup \Sigma_i \cup \Sigma_r, P, S)$ denote an immediate matching CFG, a BG $G''' = (V''', \underline{\Sigma} \cup \Sigma \cup \overline{\Sigma}, P''', S''')$ and superficial mapping h are obtained from G in three steps as follows:

First step: $A \rightarrow_{G'} cBr$ iff $A \rightarrow_G cBr$, $A \rightarrow_{G'} A'C, A' \rightarrow cBr$ iff $A \rightarrow_G cBrC$, $A \rightarrow_{G'} iB$ iff $A \rightarrow_G iB$, $A \rightarrow_{G'} \varepsilon$ iff $A \rightarrow_G \varepsilon$.

Second step: $A \rightarrow_{G''} c\mathcal{R}_B r$ iff $A \rightarrow_{G'} cBr$, where \mathcal{R}_B describes the language $L(B)$ over $\Sigma_i \cup V_{\text{carr}}$, $V_{\text{carr}} = \{A \mid A \rightarrow_{G'} cBr\}$.

Third step: $A \rightarrow_{G'''} \underline{a}\mathcal{R}_B \overline{a}$, $h(\underline{a}) = c$, $h(\overline{a}) = r$ iff $A \rightarrow_{G''} c\mathcal{R}_B r$.

Lemma 4. For any immediate matching CFG G and its corresponding BG G' plus superficial mapping h as of Definition 14, their languages coincide.

Proof. In the first step, we only split up some productions into two separate productions $A \rightarrow A'C$ and $A' \rightarrow cBr$, which preserves language equivalence. In the second step, every non-terminal B in right-hand sides of the form cBr is substituted with its regular language over $\Sigma_i \cup V$. This is clearly just a syntactical abbreviation, and hence, does not modify the language either. Finally, in the third step, every call is replaced by a terminal and every return is replaced by a co-terminal, with an appropriate h respectively. \square

Equivalence of $\omega\text{RL}(\text{qBL})$ and ωVPL is now shown by translating an arbitrary $\omega\text{RG}(\text{qBG})$ into an ωMSO_μ formula and an arbitrary ωVPA into an $\omega\text{RG}(\text{qBG})$, where each time the languages of the characterisations coincide.

Theorem 1. The language classes $\omega\text{RL}(\text{qBL})+h$ and ωVPL coincide.

Proof. $\omega\text{RL}(\text{qBL})+h \subseteq \omega\text{VPL}$: Let $G = (V, \Sigma_c \cup \Sigma_i \cup \Sigma_r \cup \bigcup_{n=1}^m \{g_n\}, P, S, F, \bigcup_{n=1}^m \{G_n\}, h)$ denote an arbitrary $\omega\text{RG}(\text{qBG})+h$. Its injector language is regular, and hence, is representable as an ωMSO formula by the standard translation.

Each of the injected languages $L(G_n)$ is representable as MSO_{mwm} formula φ_n respectively. Let φ'_n denote a variation of φ_n , where the formula presented in Lemma 2 is modified to $(\exists X \bigvee_{(S \rightarrow \underline{a} S_{\mathcal{R}} \overline{a}) \in P} (\Psi_{\underline{a}, \overline{a}}(i, j) \wedge \forall k \in [i, j] \Phi(k))) (i, j)$ but left unchanged otherwise. With appropriate renaming of variables, each terminal g_n can then be substituted by the corresponding formula φ'_n in the injector grammar, so that we get an ωMSO formula

$$\varphi \equiv X_S(1) \wedge \forall k \left(\bigvee_{(A \rightarrow \underline{a} B) \in P} (X_A(k) \wedge X_B(k+1) \wedge T_a(k)) \vee \bigvee_{(A \rightarrow g_n B) \in P} \exists j (X_A(k) \wedge X_B(j+1) \wedge \varphi'_n(k, j)) \right) \wedge \bigvee_{A \in F} \forall k \exists j (k < j \wedge X_A(j)).$$

Language inclusion follows from the fact that every ωMSO_μ formula can be translated into an ωVPA .

$\omega\text{RL}(\text{qBL})+h \supseteq \omega\text{VPL}$: Consider an ωVPA A and let $A' = (Q', \Sigma_c \cup \Sigma_i \cup \Sigma_r \cup \bigcup_{n=1}^m \{\Omega_n\}, \delta, q_i, F')$ denote the Büchi-automaton accepting all pseudo-runs of A . A' can be represented as right-linear injector grammar G_\uparrow with productions of the form $A \rightarrow cB$, $A \rightarrow rB$, and $A \rightarrow (p, q, f)_{n'}B$ for representing sets of summary-edges Ω_n with $(p, q, f)_{n'} \in \Omega_n$. Since summary-edges $(p, q, f)_{n'}$ are treated as terminals in A' , their f component does not contribute to the acceptance of a pseudo-run. Hence, for every production $A \rightarrow (p, q, 1)_{n'}B$, it is w.l.o.g. required that $B \in F'$.

All summary-edges stand for languages in VPL_{mwm} , and hence, are representable as VPA_{mwm} s respectively. Each VPA_{mwm} representing a summary-edge $(p, q, f)_{n'}$ can be transformed into a qBG $G_{n'}$ plus additional superficial mapping h . By combining G_\uparrow and the various $G_{n'}$ to a superficial $\omega\text{RG}(\text{qBG})$, we get the language inclusion. \square

The use of qBGs is counter-productive. BGs are more accessible as well as succinct due to the use of regular expressions in their productions. Injecting BGs instead of qBGs into ωRG s does not change the expressiveness, which is trivially true as every BG can be translated into a qBG.

Corollary 1. *The language classes $(\omega)\text{RL}(\text{BL})+h$ and $(\omega)\text{VPL}$ coincide.*

4 Conclusion

In this paper, a grammatical representation of $(\omega)\text{VPL}$ s was given. We introduced an amalgamation of ωRG s and qBGs equipped with a specific homomorphisms and showed that the resulting language class defined by the new grammar coincides with the ωVPL s.

Our grammatical approach towards $(\omega)\text{VPL}$ s provides a more natural representation of language specifications. As a small example, consider the following. Figure1(a) on the next page shows the code for traversing infinitely many finite binary trees. Let c denote a call to `traverse(...)` and let r denote the return in `traverse(...)`, then the ωVPA in Figure1(b) and the $\omega\text{RG}(\text{BG})+h$ in Figure1(c) represent all possible traces that can be generated by `main`, i.e. they are behavioural specifications of the code. It is apparent that the $\omega\text{RG}(\text{BG})+h$ resembles the pseudo code in greater detail than the corresponding ωVPA .

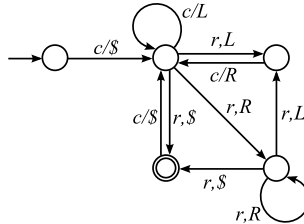
```

main :=
  do forever
    traverse(getTree())

function traverse(node n) :=
  if 'n is not a leaf' then
    traverse(n's left child)
    traverse(n's right child)
  return

```

(a) Pseudo code



(b) ωVPA

$$\begin{aligned}
 S &\rightarrow g_1 S \\
 S_1 &\rightarrow \underline{a} S_1 S_1 \bar{a} \mid \underline{a} \bar{a} \\
 h(\underline{a}) &= c, h(\bar{a}) = r \\
 F &= \{S\}
 \end{aligned}$$

(c) $\omega\text{RG}(\text{BG})+h$

Fig. 1. Representations of `traverse(...)`'s calls and returns

Acknowledgements. Joachim Baran thanks the EPSRC and the School of Computer Science for the research training awards enabling this work to be undertaken, as well as Juan Antonio Navarro-Pérez for helpful and valuable discussions regarding second-order logic.

References

- [AM04] Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, pp. 202–211. ACM Press, New York (2004)
- [AM06] Alur, R., Madhusudan, P.: Adding nesting structure to words. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 1–13. Springer, Heidelberg (2006)
- [BB02] Berstel, J., Boasson, L.: Balanced grammars and their languages. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.) Formal and Natural Computing. LNCS, pp. 3–25. Springer, Heidelberg (2002)
- [[Eng92] Engelfriet, J.: An elementary proof of double Greibach normal form. *Information Processing Letters* 44(6), 291–293 (1992)
- [HMU01] Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison-Wesley, Reading (2001)
- [LMS04] Löding, C., Madhusudan, P., Serre, O.: Visibly pushdown games. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 408–420. Springer, Heidelberg (2004)

Fully Lexicalized Pregroup Grammars

Denis Béchet¹ and Annie Foret²

¹ LINA – Université de Nantes
2, rue de la Houssinière – BP 92208
44322 Nantes Cedex 03 – France
Denis.Bechet@univ-nantes.fr
² IRISA – Université de Rennes 1
Avenue du Général Leclerc
35042 Rennes Cedex – France
Annie.Foret@irisa.fr

Abstract. Pregroup grammars are a context-free grammar formalism introduced as a simplification of Lambek calculus. This formalism is interesting for several reasons: the syntactical properties of words are specified by a set of types like the other type-based grammar formalisms ; as a logical model, compositionality is easy ; a polytime parsing algorithm exists.

However, this formalism is not completely lexicalized because each pregroup grammar is based on the free pregroup built from a set of primitive types together with a partial order, and this order is not lexical information. In fact, only the pregroup grammars that are based on primitive types with an order that is equality can be seen as fully lexicalized.

We show here how we can transform, using a morphism on types, a particular pregroup grammar into another pregroup grammar that uses the equality as the order on primitive types. This transformation is at most quadratic in size (linear for a fixed set of primitive types), it preserves the parse structures of sentences and the number of types assigned to a word.

Keywords: Pregroups, Lambek Categorical Grammars, Simulation.

1 Introduction

Pregroup grammars (PG) [1] have been introduced as a simplification of Lambek calculus [2]. Several natural languages has been modeled using this formalism: English [1], Italian [3], French [4], German [5,6], Japanese [7], etc. PG are based on the idea that sentences are produced from words using lexical rules. The syntactical properties of each word is characterized by a finite set of types stored in the lexicon. The categories are types of a free pregroup generated by a set of primitive types together with a partial order on the primitive types. This partial order is not independent of the language that corresponds to a PG. For instance, this order is used for English to specify that a yes-or-no question (primitive type q) is a correct sentence (primitive type s). Thus the partial order says that $q \leq s$.

This partial order is not lexicalized but corresponds to global rules that are specific to a particular PG. Is it then possible to find a PG that is equivalent to a

given PG but where the partial order on primitive types is *universal*? Moreover, we hope that the computed PG is not *too big* if we compare it to the source PG and if it works in a *similar* way as the source PG.

Using mathematical words, it means that transformation must be polynomial in space (the size of a resulting grammar is polynomially bounded by the size of the source grammar) and should be a homomorphism from the source free pregroup to the resulting free pregroup that must also satisfy the converse of the monotonicity condition.

The paper defines such a transformation. The size of the resulting grammar is bounded by a second degree polynomial of the initial grammar. Moreover, for all the PG that are based on the same free pregroup, this transformation is linear. The size of the tranformed PG is bounded by the size of the source PG four times the number of primitive types of the free pregroup of this PG, plus a constant. This transformation defines a homomorphism of (free) pregroups. A pregroup grammar is transformed into another pregroup grammar that associates the same number of types to a word as the source grammar: a k -valued grammar is transformed into a k -valued grammar.

After an introduction about PG in section 2, the paper proves several lemmas that are necessary for the correctness of our transformation (the source and the resulting PG must define the same language). The transformation is defined in section 4.2, its properties are detailed in the remaining sections. Section 7 concludes.

2 Background

Definition 1 (Pregroup). A pregroup is a structure $(P, \leq, \cdot, l, r, 1)$ such that $(P, \leq, \cdot, 1)$ is a partially ordered monoid¹ and l, r are two unary operations on P that satisfy for all primitive type $a \in P$ $a^l a \leq 1 \leq aa^l$ and $aa^r \leq 1 \leq a^r a$.

Definition 2 (Free pregroup). Let (P, \leq) be an ordered set of primitive types, $P^{(\mathbb{Z})} = \{p^{(i)} \mid p \in P, i \in \mathbb{Z}\}$ is the set of atomic types and $T_{(P, \leq)} = (P^{(\mathbb{Z})})^* = \{p_1^{(i_1)} \cdots p_n^{(i_n)} \mid 0 \leq k \leq n, p_k \in P \text{ and } i_k \in \mathbb{Z}\}$ is the set of types. The empty sequence in $T_{(P, \leq)}$ is denoted by 1. For X and $Y \in T_{(P, \leq)}$, $X \leq Y$ iff this relation is deducible in the following system where $p, q \in P$, $n, k \in \mathbb{Z}$ and $X, Y, Z \in T_{(P, \leq)}$:

$$\begin{array}{c}
 X \leq X \text{ (Id)} \qquad \frac{X \leq Y \quad Y \leq Z}{X \leq Z} \text{ (Cut)} \\
 \\
 \frac{XY \leq Z}{Xp^{(n)}p^{(n+1)}Y \leq Z} \text{ (A}_L\text{)} \quad \frac{X \leq YZ}{X \leq Yp^{(n+1)}p^{(n)}Z} \text{ (A}_R\text{)}
 \end{array}$$

¹ We briefly recall that a *monoid* is a structure $\langle M, \cdot, 1 \rangle$, such that \cdot is associative and has a neutral element 1 ($\forall x \in M : 1 \cdot x = x \cdot 1 = x$). A partially ordered monoid is a monoid $\langle M, \cdot, 1 \rangle$ with a partial order \leq that satisfies $\forall a, b, c : a \leq b \Rightarrow c \cdot a \leq c \cdot b$ and $a \cdot c \leq b \cdot c$.

$$\frac{Xp^{(k)}Y \leq Z}{Xq^{(k)}Y \leq Z} (IND_L) \quad \frac{X \leq Yq^{(k)}Z}{X \leq Yp^{(k)}Z} (IND_R)$$

$q \leq p$ if k is even, and $p \leq q$ if k is odd

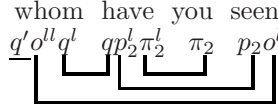
This construction, proposed by Buskowski, defines a pregroup that extends \leq on primitive types P to $T_{(P, \leq)}$.^{2,3}

The Cut Elimination. As for L and NL , the cut rule can be eliminated: every derivable inequality has a cut-free derivation.

Definition 3 (Simple free pregroup). A simple free pregroup is a free pregroup where the order on primitive type is equality.

Definition 4 (Pregroup grammar). Let (P, \leq) be a finite partially ordered set. A pregroup grammar based on (P, \leq) is a lexicalized⁴ grammar $G = (\Sigma, I, s)$ such that $s \in T_{(P, \leq)}$; G assigns a type X to a string v_1, \dots, v_n of Σ^* iff for $1 \leq i \leq n$, $\exists X_i \in I(v_i)$ such that $X_1 \cdots X_n \leq X$ in the free pregroup $T_{(P, \leq)}$. The language $\mathcal{L}(G)$ is the set of strings in Σ^* that are assigned s by G .

Example 1. Our example is taken from [8] with the primitive types: π_2 = second person, p_2 = past participle, o = object, q = yes-or-no question, q' = question. This sentence gets type q' ($q' \leq s$):



Remark on Types for Correct Sentences. Usually, type s associated to correct sentences must be a primitive type ($s \in P$). However, we use here a more general definition where s can be any type in $T_{(P, \leq)}$. Our definition does not lead to a significant modification of PG (the classes of languages is the same) because proving that $X_1 \cdots X_n \leq X$ is equivalent⁵ to prove that $X_1 \cdots X_n X^r \leq 1$.

We must notice that the transformation given in Section 4 works if we use a generalized definition of PG as above, where the type associated to correct sentences is not necessarily a primitive type or if s is neither \leq nor \geq to any other primitive type and thus does not need to be changed by the transformation.

In fact, it is always possible to transform a PG using a composed type S for correct sentences into a PG using a primitive type s for correct sentences that is

² Left and right adjoints are defined by $(p^{(n)})^l = p^{(n-1)}$, $(p^{(n)})^r = p^{(n+1)}$, $(XY)^l = Y^l X^l$ and $(XY)^r = Y^r X^r$. We write p for $p^{(0)}$. We also iterate left and right adjoints for every $X \in T_{(P, \leq)}$: $X^{(0)} = X$, $X^{(n+1)} = (X^r)^{(n)}$ and $X^{(n-1)} = (X^l)^{(n)}$.

³ \leq is only a preorder. Thus, in fact, the pregroup is the quotient of $T_{(P, \leq)}$ by the equivalence relation $X \leq Y \& Y \leq X$.

⁴ A lexicalized grammar is a triple (Σ, I, s) : Σ is a finite alphabet, I assigns a finite set of categories (or types) to each $c \in \Sigma$, s is a category (or type) associated to correct sentences.

⁵ If $Y \leq X$ then $YX^r \leq XX^r \leq 1$; if $YX^r \leq 1$ then $Y \leq YX^r X \leq X$.

not related to other primitive types by the addition of a right *wall* S^r s (the type can be seen as the type of the final point of a sentence) for the parsing with the second PG because $X \leq S \Leftrightarrow XS^r s \leq s$.

3 A Preliminary Fact

Proposition 1 (PG equivalence and generative capacity). *Let (P, \leq) be an ordered set of primitive types, for any pregroup grammar G on (P, \leq) , we can construct a PG G' based on a simple free pregroup on $(P', =)$, a set of primitive types with a discrete order, such that G and G' have the same language.*

Proof. This is obvious because PGs define context-free languages and every context-free language corresponds to an order 1 classical categorial grammars⁶ that can be easily simulated by an “order 1” simple PG [9].

Another construct is to duplicate the lexical types for each occurrence involved in an ordering.

Note. However both transformations do not preserve the size of the lexicon in general because the initial types associated to a word can correspond to an exponential number of types in the transformed PG. Moreover, the transformations do not define a pregroup homomorphism.

We look for a transformation that is polynomial in space (the size of a resulting grammar is polynomially bounded by the size of the source grammar) and that is defined using a homomorphism from the source free pregroup to the resulting free pregroup (that must also satisfy the converse of the monotonicity condition). Next section defines such a transformation. The correctness is given in the last sections and Appendix.

4 A Pregroup Morphism

4.1 Properties of Free Pregroup Morphisms

In order to simulate a free pregroup by another one with less order postulates, we consider particular mappings on posets that can be extended as a pregroup homomorphism from the free pregroup based on (P, \leq) to the free pregroup based on another poset (P', \leq') .

Definition 5 (Order-preserving mapping, pregroup homomorphism)

- A mapping from a poset (P, \leq) to a poset (P', \leq') is said order-preserving iff for all $p_i, p_j \in P : p_i \leq p_j$ implies $h(p_i) \leq' h(p_j)$.
- A mapping h from the free pregroup on (P, \leq) to the free pregroup on (P', \leq') , is a pregroup homomorphism iff

⁶ The order of primitive types is 0 and the order $order(A)$ of compound types is: $order(A/B) = order(B \setminus A) = \max(order(A, 1 + order(B)))$.

1. $\forall X \in T_{(P, \leq)} : h(X^{(n)}) = h(X)^{(n)}$
2. $\forall X, Y \in T_{(P, \leq)} : h(XY) = h(X)h(Y)$
3. $h(1) = 1$
4. $\forall X, Y \in T_{(P, \leq)} : \text{if } X \leq Y \text{ then } h(X) \leq' h(Y) \quad [Monotonicity]$

Proposition 2. *Each order-preserving mapping from (P, \leq) to (P', \leq') can be uniquely extended to a unique pregroup homomorphism on $T_{(P, \leq)}$ to $T_{(P', \leq')}$.*

Proof. The unicity comes from the three first points of the definition of pregroup homomorphism. The last point is a consequence of order-preservation which is easy by induction on a derivation \mathcal{D} for $X \leq Y$, considering the last rule:

- this holds clearly for an axiom: $h(X) \leq' h(X)$;
- if the last rule introduces $p_i^{(n)} p_i^{(n+1)}$ on the left, we have $X = Z_1 p_i^n p_i^{n+1} Z_2$ and $Z_1 Z_2 \leq Y$ as previous conclusion in \mathcal{D} , by induction we get $h(Z_1)h(Z_2) \leq' h(Y)$, and by morphism: $h(p_i^{(n)} p_i^{(n+1)}) = h(p_i)^{(n)} h(p_i)^{(n+1)} \leq' 1$, therefore $h(X) = h(Z_1)h(p_i)^{(n)} h(p_i)^{(n+1)} h(Z_2) \leq' h(Y)$
- if the last rule uses $p_i \leq p_j$ on the left if n is even, we have: $X = Z_1 p_i^{(n)} Z_2$ and $Z_1 p_j^{(n)} Z_2 \leq Y$ as previous conclusion in \mathcal{D} , by induction we get $h(Z_1)h(p_j^{(n)})h(Z_2) \leq' h(Y)$, where by morphism $h(p_j^{(n)}) = h(p_j)^{(n)}$ then: $h(X) = h(Z_1)h(p_i)^{(n)} h(Z_2) \leq' h(Y)$ since by order-preservation $h(p_i) \leq' h(p_j)$
- if n is odd, we proceed similarly from $Z_1 p_i^{(n)} Z_2 \leq Y$ as previous conclusion in \mathcal{D}
- Rules on the right are similar (or consider $Y = 1$, then morphism properties)

In order to have a simulation, we need a converse of monotonicity.

Proposition 3 (Order-reflecting homomorphism). *Every homomorphism h from the free pregroup on (P, \leq) to the free pregroup on (P', \leq') is such that (1) and (2) are equivalent conditions and define order-reflecting homomorphisms:*

- (1). $\forall X, Y \in T_{(P, \leq)} \text{ if } h(X) \leq' h(Y) \text{ then } X \leq Y \text{ by } (P, \leq).$
- (2). $\forall X \in T_{(P, \leq)} \text{ if } h(X) \leq' 1 \text{ then } X \leq 1 \text{ by } (P, \leq).$

Proof. In fact, (2) is obviously a subcase of (1) and (1) is a corollary of (2) as follows: suppose (2) holds for a pregroup-morphism h with (3) $h(X) \leq' h(Y)$, we get (4) $h(X Y^r) \leq' 1$ by adding $h(Y)^r$ on the right of (3) as below:

$$h(X Y^r) = h(X) h(Y)^r \leq' h(Y) h(Y)^r \leq' 1$$

property (2) then gives (5) $X Y^r \leq 1$,

hence $X \leq Y$ (by adding Y on the right of (5): $X \leq X Y^r Y \leq Y$).

These order-reflecting properties will be shown in the simulation-morphism defined in the next subsection.

4.2 Defining a Simulation-Morphism

Preliminary Remark. The definition has to be chosen carefully as shown by the following pregroup homomorphisms, that are not order-reflecting in the simplest case of one order postulate.

Example 2. Let $P = \{p_0, p_1, \dots, p_n\}$ and \leq be reduced to one postulate $p_0 \leq p_1$, and let \leq' be $=$:

- take $h(p_1) = q$, and $h(p_0) = \beta \beta^{(1)} q \gamma \gamma^{(1)}$ with $h(p_i) = p_i$ if $i \notin \{0, 1\}$ where β and γ are fixed types.
This defines a homomorphism (using $\beta\beta^{(1)} \leq' 1$ and $\gamma\gamma^{(1)} \leq' 1$).
- If we drop β in h of this example ($\beta = 1$), the resulting translation h_γ is a homomorphism but is not order-reflecting as exemplified by: $X = p_0^{(2n-2)} p_0^{(2n-1)} p_1^{(2n)} p_1^{(2n-1)}$, such that $X \not\leq 1$, whereas $h_\gamma(X) \leq' 1$; similarly, if we drop γ in h , defining h_β , $X = p_1^{(2n+1)} p_1^{(2n)} p_0^{(2n+1)} p_0^{(2n+2)}$ is a counter-example for h_β ⁷

The Unrestricted Case. In this presentation, we allow to simulate either a fragment (represented as Pr below) or the whole set of primitive types ; for example, we may want not to transform isolated (i.e. not related to another type) primitive types, or to proceed incrementally.

Let $Pr = \{p_1, \dots, p_n\}$ and $P = Pr \cup Pr'$, where no element of Pr is related by \leq to an element of Pr' , and each element of Pr is related by \leq to another element of Pr .

We introduce new letters q and β_k, γ_k , for each element p_k of Pr .⁸

We take as poset $P' = Pr' \cup \{q\} \cup \{\beta_k, \gamma_k \mid p_k \in Pr\}$, with \leq' as the restriction of \leq on Pr' (\leq' is reduced to identity if Pr' is empty, corresponding to the case of a unique connex component).

We then define the simulation-morphism h for Pr as follows:

Definition 6 (Simulation-morphism h for Pr)

$h(X^{(n)}) = h(X)^{(n)}$	$h(p_i) = \underbrace{\lambda_i^{(0)} q^{(0)} \delta_i^{(0)}}_{\text{for } p_i \in Pr}$	$h(1) = 1$
$h(X.Y) = h(X).h(Y)$	$h(p_i) = p_i \text{ if } p_i \in Pr'$	

where λ_i is the concatenation of α and all the terms $\gamma_{k'}^{(1)} \gamma_{k'}$ for all the indices of the primitive types $p_{k'} \in P$ less than or equal to p_i and similarly for δ_i :

$$\lambda_i = \alpha \underbrace{\gamma_{k'}^{(1)} \gamma_{k'}}_{\text{if } p_{k'} \leq p_i} \underbrace{\gamma_{k'}^{(1)} \gamma_{k'}}_{\text{if } p_{k'} \leq p_i} \dots \underbrace{\gamma_{k'}^{(1)} \gamma_{k'}}_{\text{if } p_{k'} \leq p_i}, \quad \delta_i = \underbrace{\beta_k \beta_k^{(1)}}_{\text{if } p_k \geq p_i} \underbrace{\beta_k \beta_k^{(1)}}_{\text{if } p_k \geq p_i} \dots \underbrace{\beta_k \beta_k^{(1)}}_{\text{if } p_k \geq p_i} \alpha'$$

that we also write:

$$\lambda_i = \alpha \left(\prod_{\substack{\text{for } k'=n..1 \\ \text{if } p_{k'} \leq p_i}} (\gamma_{k'}^{(1)} \gamma_{k'}) \right) \quad \text{and} \quad \delta_i = \left(\prod_{\substack{\text{for } k=1..n, \\ \text{if } p_k \geq p_i}} (\beta_k \beta_k^{(1)}) \right) \alpha'$$

Note the inversion by odd exponents : $h(p_i)^{(1)} = \underbrace{\delta_i^{(1)} q^{(1)} \lambda_i^{(1)}}_{\text{for } p_i \in Pr}$

Proposition 4 (h is order-preserving). if $p_i \leq p_j$ then $h(p_i) \leq' h(p_j)$.

It is then a pregroup homomorphism.

⁷ Details: $X = p_0^{(2n-2)} p_0^{(1)} p_1^{(2)} p_1^{(1)} \xrightarrow{h_\gamma} p_1 \gamma \gamma^{(1)} \gamma^{(2)} \gamma^{(1)} p_1^{(1)} p_1^{(2)} p_1^{(1)} \mapsto^* 1$

$X = p_1^{(1)} p_1^{(0)} p_0^{(1)} p_0^{(2)} \xrightarrow{h_\beta} p_1^{(1)} p_1^{(0)} p_1^{(1)} \beta^{(2)} \beta^{(1)} \beta^{(2)} \beta^{(3)} p_1^{(2)} \mapsto^* 1$

⁸ The symbol q can also be written q_{Pr} if necessary w.r.t. Pr .

Proof. Easy by construction (if $p_i \leq p_j$, we can check that $\lambda_i \leq \lambda_j$ and $\delta_i \leq \delta_j$, using the transitivity of \leq and inequalities of the form $\beta\beta^{(1)} \leq 1$ and $1 \leq \gamma^{(1)}\gamma$)

We get the first property required for a simulation, as a corollary:

Proposition 5 (Monotonicity of h)

$$\forall X, Y \in T_{(P, \leq)} : \text{if } X \leq Y \text{ then } h(X) \leq' h(Y)$$

5 Order-Reflecting Property : Overview and Lemmas

We show in the next subsection that if $h(X) \leq' 1$ then $X \leq 1$. We proceed by reasoning on a deduction \mathcal{D} for $h(X) \leq' 1$. We explain useful facts on pregroup derivations when the right side is 1.

5.1 Reasoning with Left Derivations

Let \mathcal{D} be the succession of sequents starting from $1 \leq' 1$ to $h(X) \leq' 1$, such that q is not related to any other primitive type by \leq' .

$$\underbrace{\Delta_0}_{=1} \leq' 1 \text{ (by rule (Id))} \dots \Delta_i \leq' 1 \dots \underbrace{\Delta_m}_{=h(X)} \leq' 1$$

Each occurrence of q in $h(X)$ is introduced by rule (A_L) . Let k denote the step of the leftmost introduction of q , let $(n), (n+1)$ be the corresponding exponents of q , we can write:

$$\begin{array}{ll} \vdots & \\ \Delta_{k-1} = \Gamma_0 \Gamma'_0 & \text{with } \Delta_{k-1} \leq' 1 \\ \Delta_k = \Gamma_0 q^{(n)} q^{(n+1)} \Gamma'_0 & \text{with } \Delta_k \leq' 1 \\ \vdots & \\ \Delta_m = \Gamma_{m-k} q^{(n)} \Gamma''_{m-k} q^{(n+1)} \Gamma'_{m-k} & \text{with } h(X) = \Delta_m \leq' 1 \end{array}$$

Γ_{m-k} is obtained from Γ_0 by successive applications of rule (A_L) or (IND_L) ; similarly for Γ'_{m-k} from Γ'_0 :

$$\begin{aligned} \Gamma_{m-k} &\leq' \Gamma_{m-k-1} \dots \leq' \Gamma_i \leq' \dots \Gamma_1 \leq' \Gamma_0 \\ \Gamma'_{m-k} &\leq' \Gamma'_{m-k-1} \dots \leq' \Gamma'_i \leq' \dots \Gamma'_1 \leq' \Gamma'_0 \end{aligned}$$

Γ''_{m-k} is also obtained from Γ_0 by successive applications of rule (A_L) or (IND_L) but from an empty sequence, in particular:

$$\Gamma''_{m-k} \leq' 1$$

We shall discuss according to some properties of the distinguished occurrence $q^{(n)}$ and $q^{(n+1)}$: the parity of n and whether they belong to images of p_i and of p_j in $h(X)$ such that $p_i \leq p_j$, $p_j \leq p_i$ or not.

5.2 Technical Lemmas

Lemma 1. let $X = \underbrace{X_1 \alpha^{(2u_1)} Y_1 \alpha^{(2u_1+1)}} \dots \underbrace{X_k \alpha^{(2u_k)} Y_k \alpha^{(2u_k+1)}} \dots \underbrace{X_n \alpha^{(2u_n)} Y_n \alpha^{(2u_n+1)}} X_{n+1}$

where all X_k, Y_k have no occurrence of α , and α is not related by \leq to other primitive types ;

- (i) if $X \leq 1$ then :
 - (1) $\forall k : Y_k \leq 1$
 - (2) $X_1 X_2 \dots X_k \dots X_n X_{n+1} \leq 1$
- (ii) if $Y_0 \alpha^{(2v+1)} X \alpha^{(2v+2)} \leq 1$ where Y_0 has no α , then :
 - (1) $\forall k : Y_k \leq 1$
 - (2) $X_1 X_2 \dots X_k \dots X_n X_{n+1} \leq 1$

The proof is given in Appendix.

Variants. A similar result holds in the odd case (called “Bis version”) for

$$X = \underbrace{X_1 \alpha^{(2u_1-1)} Y_1 \alpha^{(2u_1)}} \dots \underbrace{X_k \alpha^{(2u_k-1)} Y_k \alpha^{(2u_k)}} \dots \underbrace{X_n \alpha^{(2u_n-1)} Y_n \alpha^{(2u_n)}} X_{n+1}$$

Lemma 2.⁹ Let I_{Pr} denote the set of indices of elements in Pr . Let $\Gamma = \underbrace{h(X_1) Y_1 \dots h(X_k) Y_k \dots h(X_m) Y_m}_{m \geq 0} h(X_{m+1})$

where all Y_k have the following form: $\lambda_i^{2u} \lambda_j^{2u+1}$ or $\delta_i^{2u-1} \delta_j^{2u}$ with $i, j \in I_{Pr}$ ¹⁰ where some X_k may be empty (then considered as 1, with $h(1) = 1$)

- (1) If $\Gamma \leq' 1$ then $\forall k_1 \leq m : (\forall k \leq k_1 : h(X_k) \text{ has no } q) \Rightarrow (\forall k \leq k_1 : Y_k \leq' 1)$
- (2) If $\Gamma \leq' 1$ then $X_1 X_2 \dots X_m X_{m+1} \leq 1$
- (3) If $\delta_i^{(2k)} \Gamma \delta_j^{(2k+1)} \leq' 1$, or $\lambda_i^{(2k+1)} \Gamma \lambda_j^{(2k+2)} \leq' 1$ then $\forall k_1 \leq m : (\forall k \leq k_1 : h(X_k) \text{ has no } q) \Rightarrow (\forall k \leq k_1 : Y_k \leq' 1)$
- (4) If $\delta_i^{(2k)} \Gamma \delta_j^{(2k+1)} \leq' 1$, or $\lambda_i^{(2k+1)} \Gamma \lambda_j^{(2k+2)} \leq' 1$ then $X_1 X_2 \dots X_m X_{m+1} \leq 1$

Notation : $\sigma(\Gamma')$ will then denote $X_1 X_2 \dots X_m X_{m+1}$, this writing is unique.¹¹

The proof is technical and is given in Appendix : a key-point is that some derivations are impossible.

6 Main Results

Proposition 6 (Order-reflecting property). The simulation-morphism h from the free pregroup on (P, \leq) to the free pregroup on (P', \leq') satisfies (1) and (2):

- (1). $\forall X, Y \in T_{(P, \leq)}$ if $h(X) \leq' h(Y)$ then $X \leq Y$ by (P, \leq) .
- (2). $\forall X, Y \in T_{(P, \leq)}$ if $h(X) \leq' 1$ then $X \leq 1$ by (P, \leq) .

⁹ Some complications in the formulation (1) and (3) simplify in fact the discussion in the proof.

¹⁰ We get later $p_i \not\leq p_j$ in the first form, $p_j \not\leq p_i$ in the second form.

¹¹ Due to the form of $h(p_i)$ and of δ_i, λ_i .

In fact, (1) can be shown from Lemma 2 (2) (case $m = 0$) and (2) is equivalent to (1) as explained before in Proposition 3.

As a corollary of monotonicity and previous proposition, we get:

Proposition 7 (Pregroup Order Simulation). *The simulation-morphism h from the free pregroup on (P, \leq) to the free pregroup on (P', \leq') enjoys the following property:*

$$\forall X, Y \in T_{(P, \leq)} \quad h(X) \leq' h(Y) \text{ iff } X \leq Y$$

Proposition 8 (Pregroup Grammar Simulation). *Given a pregroup grammar $G = (\Sigma, I, s)$ on the free pregroup on a poset (P, \leq) , we consider $Pr = P$ and we construct the simulation-morphism h from the free pregroup on (P, \leq) to the simple free pregroup on (P', \leq') ; we construct a grammar $G' = (\Sigma, h(I), h(s))$, where $h(I)$ is the assignment of $h(X_i)$ to a_i for each $X_i \in I(a_i)$, as a result we have : $\mathcal{L}(G) = \mathcal{L}(G')$*

This proposition applies the transformation to the whole set of primitive types, thus providing a fully lexicalized grammar G' (with no order postulate). A similar result holds when only a fragment of P is transformed as discussed in the construction of h .

7 Conclusion

In the framework of categorial grammars, pregroups were introduced as a simplification of Lambek calculus. Several aspects have been modified. The order on primitive types has been introduced in PG to simplify the calculus for simple types. The consequence is that PG is not fully lexicalized and a parser should take into account this information while performing syntactical analysis.

We have proven that this restriction is not so important because a PG using an order on primitive types can be transformed into a PG based on a simple free pregroup using a pregroup morphism whose size is bound by the size of the initial PG times the number of primitive types (times a constant which is approximatively 4). Usually the order on primitive type is not very complex, thus the size of the resulting PG is often much less than this bound. Moreover, this transformation does not change the number of types that are assigned to a word (a k -valued PG is transformed into a k -valued PG).

References

1. Lambek, J.: Type grammars revisited. In: Lecomte, A., Perrier, G., Lamarche, F. (eds.) LACL 1997. LNCS (LNAI), vol. 1582, pp. 22–24. Springer, Heidelberg (1999)
2. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958)
3. Casadio, C., Lambek, J.: An algebraic analysis of clitic pronouns in italian. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, Springer, Heidelberg (2001)

4. Bargelli, D., Lambek, J.: An algebraic approach to french sentence structure. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, Springer, Heidelberg (2001)
5. Lambek, J.: Type grammar meets german word order. Theoretical Linguistics 26, 19–30 (2000)
6. Lambek, J., Preller, A.: An algebraic approach to the german noun phrase. Linguistic Analysis 31, 3–4 (2003)
7. Cardinal, K.: An algebraic study of Japanese grammar. McGill University, Montreal (2002)
8. Lambek, J.: Mathematics and the mind. In: Abrusci, V., Casadio, C. (eds.) New Perspectives in Logic and Formal Linguistics. In: Proceedings Vth ROMA Workshop, Bulzoni Editore (2001)
9. Buszkowski, W.: Lambek grammars based on pregroups. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, Springer, Heidelberg (2001)

Appendix

Proof of Lemma 1 (i): By induction on the number n of terms $\underbrace{X_k \alpha^{(2u_k)} Y_k \alpha^{(2u_k+1)}}_{}$.

- If $n > 1$, let $(2u_j + 1)$ denote the rightmost exponent of α such that it is greater or equal than all exponents of α in X ; consider a derivation of $X \leq 1$ and the step when $\alpha^{(2u_j+1)}$ appears first in the derivation:
 $Z_1 \alpha^{(2u_j)} \alpha^{(2u_j+1)} Z_2 \leq 1$

Next steps lead to independent introductions in Z_1 , Z_2 and between $\alpha^{(2u_j)} \alpha^{(2u_j+1)}$, by construction of derivations, its ending can be written:

$$\boxed{Z'_1 \alpha^{(2u_j)} Z'_3 \alpha^{(2u_j+1)} Z'_2 \leq 1} \text{ where } \boxed{Z'_3 \leq 1} \text{ and } \boxed{Z'_1 Z'_2 \leq 1}$$

We now discuss whether $Z'_3 = Y_j$ or not.

- If $Z'_3 = Y_j$ (where Y_j has no α) then :

$$Z'_1 Z'_2 = \underbrace{X_1 \alpha^{(2u_1)} Y_1 \alpha^{(2u_1+1)}}_{\dots} \underbrace{X_{j-1} \alpha^{(2u_{j-1})} Y_{j-1} \alpha^{(2u_{j-1}+1)}}_{\dots} \underbrace{X_j \alpha^{(2u_j)} Y_j \alpha^{(2u_j+1)}}_{\dots} \underbrace{X_{j+1} \alpha^{(2u_{j+1})} Y_{j+1} \alpha^{(2u_{j+1}+1)}}_{\dots} \underbrace{X_n \alpha^{(2u_n)} Y_n \alpha^{(2u_n+1)}}_{\dots} X_{n+1}$$

We then apply the hypothesis to $n - 1$ terms $\underbrace{X'_k \alpha^{(2u_k)} Y'_k \alpha^{(2u_k+1)}}_{}$ in $X' = Z'_1 Z'_2$, taking $X'_j j := X_j X_{j+1}$,

$$X'_{j+m} := X_{j+m-1} \text{ and } Y'_{j+m} := Y_{j+m-1}:$$

$$(1)' \forall k \neq j : Y_k \leq 1$$

$$(2) X_1 X_2 \dots X_k \dots X_n X_{n+1} \leq 1$$

we get full (1) for X from above : $Z'_3 = Y_j$ and $Z'_3 \leq 1$.

- If $Z'_3 \neq Y_j$, since X_k, Y_k have no α , and $(2u_j + 1)$ is a highest exponent, there must exist u_i , with $i < j$ such that $u_i = u_j$, therefore:

$$Z'_3 = Y_i \alpha^{(2u_i+1)} Z'_4 X_j \alpha^{(2u_i)} Y_j \text{ where } Z'_4 \text{ is either empty or the sequence of successive terms } \underbrace{X_k \alpha^{(2u_k)} Y_k \alpha^{(2u_k+1)}}_{}, \text{ for } i < k < j.$$

This is impossible, since the first $\alpha^{(2u_i+1)}$ on the left of Z'_3 , requires an occurrence of a greater exponent on the right in a derivation for $Z'_3 \leq 1$, whereas $\alpha^{(2u_i+1)}$ is a highest exponent.

- If $n = 1$, the discussion starts as above, the derivation ending is:
 $Z'_1 \alpha^{(2u_j)} Z'_3 \alpha^{(2u_j+1)} Z'_2 \leq 1$ where $Z'_3 \leq 1$ and $Z'_1 Z'_2 \leq 1$
 which coincides with (1) and (2) for X ■

Proof of Lemma 1 (ii) : By induction on the number $n \geq 1$ of terms $\underbrace{X_k \alpha^{(2u_k)} Y_k \alpha^{(2u_k+1)}}$.

Let $(2u_j + 1)$ denote the rightmost exponent of α in X such that it is greater or equal than all exponents of α in X ; we consider a derivation \mathcal{D} of $Y_0 \alpha^{(2v+1)} X \alpha^{2v+2} \leq 1$ and the step when $\alpha^{(2u_j+1)}$ (rightmost maximum in X) appears first in the derivation.

- If the rightmost $\alpha^{(2u_j+1)}$ is introduced with the $\alpha^{(2v+2)}$ on its right, the introduction step in \mathcal{D} is of the form:

$$Z_1 \alpha^{(2u_j+1)} \alpha^{(2u_j+2)} \leq 1 \text{ with } u_j = v \text{ and } Z_1 \leq 1 \text{ as antecedent in } \mathcal{D} .$$

The next steps in \mathcal{D} are introductions in Z_1 and between $\alpha^{(2u_j)} \alpha^{(2u_j+1)}$, by construction of derivations, its ending can be written:

$$\boxed{Z'_1 \alpha^{(2u_j+1)} Z'_3 \alpha^{(2u_j+2)} \leq 1} \text{ where } \boxed{Z'_3 \leq 1} \text{ and } \boxed{Z'_1 \leq Z_1 \leq 1}$$

The beginning of Z'_1 must be $Y_0 \alpha^{(2v+1)}$, where $2v+1$ is the highest exponent of α in Z'_1 (Y_0 having no α). This case is thus impossible, because in a derivation of $Z'_1 \leq 1$, the $\alpha^{(2v+1)}$ in the left of Z'_1 should combine with an $\alpha^{(2v+2)}$ on its right.

- The rightmost $\alpha^{(2u_j+1)}$ is thus introduced with an $\alpha^{(2u_j)}$ on its left, this step is:

$$Z_1 \alpha^{(2u_j)} \alpha^{(2u_j+1)} Z_2 \leq 1 \text{ with } Z_1 Z_2 \leq 1 \text{ as antecedent in } \mathcal{D} .$$

Next steps lead to independent introductions in Z_1 , Z_2 and between $\alpha^{(2u_j)} \alpha^{(2u_j+1)}$, by construction of derivations, the ending of \mathcal{D} can be written:

$$\boxed{Z'_1 \alpha^{(2u_j)} Z'_3 \alpha^{(2u_j+1)} Z'_2 \leq 1} \text{ where } \boxed{Z'_3 \leq 1} \text{ and } \boxed{Z'_1 Z'_2 \leq 1}$$

The ending of Z'_3 must be Y_j . We now detail the possibilities.

If $n = 1$, by hypothesis $Y_0 \alpha^{(2v+1)} \underbrace{X_1 \alpha^{(2u_1)} Y_1 \alpha^{(2u_1+1)}}_{=X} X_2 \alpha^{(2v+2)} \leq 1$,

the inequality $Z'_3 \leq 1$ is $Y_1 = Z'_3 \leq 1$, that is a part of (1) and $Z'_1 Z'_2 \leq 1$ is: $Y_0 \alpha^{(2v+1)} X_1 X_2 \alpha^{(2v+2)}$ where Y_0, X_1, X_2 have no α , therefore, by construction of derivations, we must have $Y_0 \leq 1$ (the end of (1)) and $X_1 X_2 \leq 1$ that is (2).

If $n > 1$, we first show that we must have $Z'_3 = Y_j$.

- If $Z'_3 \neq Y_j$, since X_k, Y_k have no α , and $(2u_j + 1)$ is a highest exponent in X , if it not combined with α^{2v+2} , there must exist u_i , with $i < j$ such that $u_i = u_j$, therefore:

$Z'_3 = Y_i \alpha^{(2u_i+1)} Z'_4 X_j \alpha^{(2u_i)} Y_j$ where Z'_4 is either empty or the sequence of successive terms $\underbrace{X_k \alpha^{(2u_k)} Y_k \alpha^{(2u_k+1)}}_{\text{for } i < k < j}$.

This is impossible, since the first $\alpha^{(2u_i+1)}$ on the left of Z'_3 , requires an occurrence of a greater exponent on the right in a derivation for $Z'_3 \leq 1$, whereas $\alpha^{(2u_i+1)}$ is a highest exponent.

- $Z'_3 = Y_j$ (where Y_j has no α) then :

$$\begin{aligned} Z'_1 Z'_2 = & Y_0 \alpha^{2v+1} \underbrace{X_1 \alpha^{(2u_1)} Y_1 \alpha^{(2u_1+1)}} \dots \underbrace{X_{j-1} \alpha^{(2u_{j-1})} Y_{j-1} \alpha^{(2u_{j-1}+1)}} \\ & X_j \underbrace{X_{j+1} \alpha^{(2u_{j+1}+1)} Y_{j+1} \alpha^{(2u_{j+1}+1)}} \dots \underbrace{X_n \alpha^{(2u_n)} Y_n \alpha^{(2u_n+1)}} X_{n+1} \alpha^{2v+2} \end{aligned}$$

we apply the hypothesis to $n-1$ terms $\underbrace{X'_k \alpha^{(2u_k)} Y'_k \alpha^{(2u_k+1)}}$

in $X' = Z'_1 Z'_2$, taking $X'_j := X_j X_{j+1}$, $X'_{j+m} := X_{j+m-1}$ and $Y'_{j+m} := Y_{j+m-1}$:

$$(1)' \forall k \neq j : Y_k \leq 1$$

$$(2) X_1 X_2 \dots X_k \dots X_n X_{n+1} \leq 1$$

we get full (1) for X from above : $Z'_3 = Y_j$ and $Z'_3 \leq 1$. ■

Detailed writing and properties of the auxiliary types

Lemma 2 uses the following facts on auxiliary types :

- if $p_i \leq p_j$ then $\lambda_i^{2u} \lambda_j^{2u+1} \leq' 1$
- if $p_i \not\leq p_j$ then $\lambda_i^{2u} \lambda_j^{2u+1} \not\leq' 1$
- if $p_j \leq p_i$ then $\delta_i^{2u-1} \delta_j^{2u} \leq' 1$
- if $p_j \not\leq p_i$ then $\delta_i^{2u-1} \delta_j^{2u} \not\leq' 1$

This can be summarized as : the only $Y_k \not\leq 1$ of Lemma 2 are $\lambda_i^{2u} \lambda_j^{2u+1}$ for $p_i \not\leq p_j$ and $\delta_i^{2u-1} \delta_j^{2u}$ for $p_j \not\leq p_i$

This can be checked using the following detailed writing for the auxiliary types

$$\begin{aligned} \lambda_i^{2u} \lambda_j^{2u+1} \ (i=j): & \quad \alpha^{(2u)} \left(\prod_{\substack{\text{for } k'=n..1 \\ \text{if } p_{k'} \leq p_i}} (\gamma_{k'}^{(2u+1)} \gamma_{k'}^{(2u)}) \right) \left(\prod_{\substack{\text{for } k=1..n \\ \text{if } p_k \leq p_i}} (\gamma_k^{(2u+1)} \gamma_k^{(2u+1+1)}) \right) \alpha^{(2u+1)} \\ \delta_i^{2u-1} \delta_j^{2u} \ (i=j): & \quad \alpha'^{(2u-1)} \left(\prod_{\substack{\text{for } k'=n..1 \\ \text{if } p_{k'} \geq p_i}} (\beta_{k'}^{(2u-1+1)} \beta_{k'}^{(2u-1)}) \right) \left(\prod_{\substack{\text{for } k=1..n, \\ \text{if } p_k \geq p_i}} (\beta_k^{(2u)} \beta_k^{(2u+1)}) \right) \alpha'^{(2u)} \end{aligned}$$

$$\lambda_i^{2u} \lambda_j^{2u+1} \ (i \neq j): \quad \alpha^{(2u)} \left(\prod_{\substack{\text{for } k'=n..1 \\ \text{if } p_{k'} \leq p_i}} (\gamma_{k'}^{(2u+1)} \gamma_{k'}^{(2u)}) \right) \left(\prod_{\substack{\text{for } k=1..n \\ \text{if } p_k \leq p_j}} (\gamma_k^{(2u+1)} \gamma_k^{(2u+1+1)}) \right) \alpha^{(2u+1)}$$

$$\delta_i^{2u-1} \delta_j^{2u} \ (i \neq j): \quad \alpha'^{(2u-1)} \left(\prod_{\substack{\text{for } k'=n..1 \\ \text{if } p_{k'} \geq p_i}} (\beta_{k'}^{(2u-1+1)} \beta_{k'}^{(2u-1)}) \right) \left(\prod_{\substack{\text{for } k=1..n, \\ \text{if } p_k \geq p_j}} (\beta_k^{(2u)} \beta_k^{(2u+1)}) \right) \alpha'^{(2u)}$$

Proof of Lemma 2. We consider the *leftmost introduction* of q in a derivation of $\Gamma \leq' 1$, involving p_i^n, p_j^{n+1} . The inductives Cases for (3)(4) can be treated similarly as (1)(2).

If there is no q , we apply lemma1: if there is no q , each Y_k is of the form $\alpha^{(2u)} Z_k \alpha^{(2u+1)}$ where Z_k has no $\alpha^{(\cdot)}$ or of the form $\alpha'^{(2u-1)} Z_k \alpha'^{(2u)}$ where Z_k has no $\alpha'^{(\cdot)}$; we apply lemma 1, using $\alpha^{(\cdot)}$, then the second version of lemma 1, using $\alpha'^{(\cdot)}$

We now suppose there is an occurrence of q .

- $n = 2k$ even, case $q^{(n)}$ belongs to $h(p_i)^{(2k)}, q^{(n+1)}$ belongs to $h(p_j)^{(2k+1)}$

We can write: $\Gamma = Z h(p_i)^{(2k)} Z'' h(p_j)^{(2k+1)} Z'$ where Z has no q

$$\text{i.e. } \Gamma = Z \underbrace{\lambda_i^{(2k)} q^{(2k)} \delta_i^{(2k)}}_{\text{where } \Gamma_1, \Gamma_2, \Gamma_3 \text{ have a form similar to that of } \Gamma} Z'' \underbrace{\delta_j^{(2k+1)} q^{(2k+1)} \lambda_j^{(2k+1)}}_{\text{where } \Gamma_1, \Gamma_2, \Gamma_3 \text{ have a form similar to that of } \Gamma} Z'$$

where $\Gamma_1, \Gamma_2, \Gamma_3$ have a form similar to that of Γ .

By construction of \mathcal{D} , we have two sequents for which the lemma applies:

$$\boxed{Z \lambda_i^{(2k)} \lambda_j^{(2k+1)} Z' \leq' 1} \quad \text{and} \quad \boxed{\delta_i^{(2k)} Z'' \delta_j^{(2k+1)} \leq' 1}$$

by induction

- from (1): this entails (1) for $Y_k \in Z$, which proves (1) for Γ ;
- from (2): $\sigma(Z)\sigma(Z') \leq 1$
- and from (3): $\sigma(Z'') \leq 1$.

For each $p_i \leq p_j$, we get a derivation of $\sigma(Z)p_i^{(n)} p_j^{(n+1)} \sigma(Z') \leq 1$, then of $\sigma(Z)p_i^{(n)} \sigma(Z'') p_j^{(n+1)} \sigma(Z') \leq 1$.

That is $\sigma(\Gamma) \leq 1$, when $p_i \leq p_j$.

- $n = 2k+1$ odd, case $q^{(n)}$ belongs to $h(p_i)^{(2k+1)}, q^{(n+1)}$ belongs to $h(p_j)^{(2k+2)}$

We then have: $\Gamma = Z h(p_i)^{(2k+1)} Z'' h(p_j)^{(2k+2)} Z'$

$$\text{i.e. } \Gamma = Z \underbrace{\delta_i^{(2k+1)} q^{(2k+1)} \lambda_i^{(2k+1)}}_{\text{where } \Gamma_1, \Gamma_2, \Gamma_3 \text{ have a form similar to that of } \Gamma} Z'' \underbrace{\lambda_j^{(2k+2)} q^{(2k+2)} \delta_j^{(2k+2)}}_{\text{where } \Gamma_1, \Gamma_2, \Gamma_3 \text{ have a form similar to that of } \Gamma} Z'$$

By construction of \mathcal{D} :

$$\boxed{Z \delta_i^{(2k+1)} \delta_j^{(2k+2)} Z' \leq' 1} \quad \text{and} \quad \boxed{\lambda_i^{(2k+1)} Z'' \lambda_j^{(2k+2)} \leq' 1}$$

by induction,

- from (1) this entails (1) for $Y_k \in Z$, which proves (1) for Γ ;
- from (2): $\sigma(Z)\sigma(Z') \leq 1$
- and from (3): $\sigma(Z'') \leq 1$.

For each $p_j \leq p_i$, we get a derivation of $\sigma(Z)p_i^{(n)} p_j^{(n+1)} \sigma(Z') \leq 1$, then of $\sigma(Z)p_i^{(n)} \sigma(Z'') p_j^{(n+1)} \sigma(Z') \leq 1$. That is $\sigma(\Gamma) \leq 1$, when $p_j \leq p_i$.

$$- n = 2k, p_i \not\leq p_j : \Gamma = \underbrace{Z}_{no\ q} \underbrace{\lambda_i^{(2k)} q^{(2k)} \delta_i^{(2k)}}_{Z''} \underbrace{\delta_j^{(2k+1)} q^{(2k+1)} \lambda_j^{(2k+1)}}_{Z'} ,$$

such that, by construction of \mathcal{D} : $Z \lambda_i^{(2k)} \lambda_j^{(2k+1)} Z' \leq' 1$ which is impossible by induction (1) applied to it (key point: $\lambda_i^{(2k)} \lambda_j^{(2k+1)} \not\leq 1$)

$$- n = 2k + 1, p_j \not\leq p_i : \Gamma = \underbrace{Z}_{no\ q} \underbrace{\delta_i^{(2k+1)} q^{(2k+1)} \lambda_i^{(2k+1)}}_{Z''} \underbrace{\lambda_j^{(2k+2)} q^{(2k+2)} \delta_j^{(2k+2)}}_{Z'} \text{ such that,}$$

by construction of \mathcal{D} : $Z \delta_i^{(2k+1)} \delta_j^{(2k+2)} Z' \leq' 1$

which is impossible by induction (1) applied to it (key point: $\delta_i^{(2k+1)} \delta_j^{(2k+2)} \not\leq 1$)

Bounded Lattice T-Norms as an Interval Category

Benjamín C. Bedregal¹, Roberto Callejas-Bedregal², and Hélida S. Santos¹

¹ Federal University of Rio Grande do Norte, Department of Informatics and Applied Mathematics, Campus Universitário s/n, Lagoa Nova, 59.072-970 Natal, Brazil
bedregal@dimap.ufrn.br

² Federal University of Paraíba, Department of Mathematics, Cidade Universitária - Campus I, CEP: 58.051-900 João Pessoa-PB, Brazil
roberto@mat.ufpb.br

Abstract. Triangular norms or t-norms, in short, and automorphisms are very useful to fuzzy logics in the narrow sense. However, these notions are usually limited to the set $[0, 1]$.

In this paper we will consider a generalization of the t-norm notion for arbitrary bounded lattices as a category where these generalized t-norms are the objects and generalizations of automorphisms are the morphisms of the category. We will prove that, this category is an interval category, which roughly means that it is a Cartesian category with an interval covariant functor.

1 Introduction

Triangular norms were introduced by Karl Menger in [23] with the goal of constructing metric spaces using probabilistic distributions, i.e. values in the interval $[0, 1]$, instead of the real numbers set as whole, to describe the distance between two elements. However, that original proposal is very wide allowing to model (fuzzy) conjunction as well as (fuzzy) disjunction. But, with the work of Berthold Schweizer and Abe Sklar in [25] it was given an axiomatic for t-norms as they are used today. In [1], Claudi Alsina, Enric Trillas, and Llorenç Valverde, using t-norms, model the conjunction in fuzzy logics, generalizing several previous fuzzy conjunctions, provided, among others, by Lotfi Zadeh in [31], Richard Bellman and Zadeh in [6] and Ronald Yager in [30]. From a t-norm it is also possible to obtain, canonically, fuzzy interpretations for the other usual connectives [7].

Automorphisms act on t-norms generating in most of the cases a new t-norm. When we see t-norms as a semi-group operation (see for example [21,22]) the automorphism yields an isomorphism between t-norms.

On the other hand in fuzzy logic in the narrow sense, it has been very common the use of lattice theory to deal with fuzzy logic in a more general framework, see for example the L -fuzzy set theory [15], BL-algebras [18], Brouwerian lattices [28], Complete Heyting algebras [13], etc.

In [12,11] it was generalized the t-norm notion to bounded partially ordered sets, which is a more general structure than the bounded lattice. In [24] it was considered an extension of t-norms for bounded lattice which coincides with the one given by [12,11]. In this paper we will consider this notion of t-norm for arbitrary bounded lattices.

The category theory is an abstract structure composed by a collection of “objects”, together with a collection of “morphisms” between them [27]. The morphisms establish relationships between the objects such that every property of the objects must be specified through the properties of the morphisms (existence of particular morphisms, its unicity, some equations which are satisfied by them, etc.). Thus, categories provide a strongly formalized language which is appropriated in order to establish abstract properties of mathematical structures. More information about category theory can be found in [2,3,27]. So, seeing the t-norm theory as a category we gain in elegancy and in the comprehension of the general property of t-norms.

A first contribution of this paper is to provide a generalization of the notion of automorphism to bounded lattices. Since automorphism presuppose the use of the same lattice, we also generalize this notion to t-norm morphism which considers different lattices for domain and co-domain. Another contribution is to consider the product and interval lattice constructions (see for example [17,26]) to construct t-norms and t-norms morphisms. We also analyze some categorical properties of these constructions, in particular we show that this category is an interval category in the sense of [8] with the goal of providing a category theoretical foundation for the parametric interval data type. Roughly, it means that this category is Cartesian and that has an interval t-norm constructor which is a covariant functor. The interval t-norms constructor proposed in this paper extends for bounded lattices the interval t-norms constructor introduced by Bedregal and Takahashi in [5] for usual t-norms.

2 The Interval Constructor and the Category POSET

This section is based on the paper [8]. For more information on category theory see [2,3,27].

We will denote by $\mathbb{I}(\mathbb{R})$ the set $\{[r, s] / r, s \in \mathbb{R} \text{ and } r \leq s\}$ called the real intervals set. Each interval could be seen as an ordered pair or as a set ($[r, s] = \{x \in \mathbb{R} / r \leq x \leq s\}$). There are several orders which can be defined on $\mathbb{I}(\mathbb{R})$, but in this work we will take in account the order of Kulisch-Miranker [20] since it is compatible with the order of the cartesian product:

$$\begin{aligned} [a, b] \sqsubseteq [c, d] &\Leftrightarrow \forall x \in [a, b] \exists y \in [c, d], x \leq y \text{ and } \forall y \in [c, d] \exists x \in [a, b], x \leq y \\ &\Leftrightarrow a \leq c \text{ and } b \leq d \end{aligned}$$

Notice that this order depends upon the usual real order. Thus, we can generalize this construction by considering, instead of the real set with its usual order, any partially order set, it means that we can think of intervals as a constructor on posets.

Let $\mathbf{D} = \langle D, \leq \rangle$ be a poset. The poset $I(\mathbf{D}) = \langle I(D), \sqsubseteq \rangle$, where

- $I(D) = \{[a, b] / a, b \in D \text{ and } a \leq b\}$
- $[a, b] \sqsubseteq [c, d] \Leftrightarrow a \leq c \text{ and } b \leq d$

is called the **poset of intervals of D**.

Posets as objects and monotonic functions as morphisms are a Cartesian closed category, denoted by **POSET**.

Let $\mathbf{D}_1 = \langle D_1, \leq_1 \rangle$ and $\mathbf{D}_2 = \langle D_2, \leq_2 \rangle$ be posets. Let $f : D_1 \longrightarrow D_2$ be a monotonic function. Then the function $I(f) : I(D_1) \longrightarrow I(D_2)$, defined by $I(f)([a, b]) =$ [f(a), f(b)]

$[f(a), f(b)]$ is also monotonic. Thus we have a covariant functor $I : \mathbf{POSET} \longrightarrow \mathbf{POSET}$.

Let \mathcal{C} be a category with product. Then $Prod : \mathcal{C} \longrightarrow \mathcal{C}$ defined by $Prod(A) = A \times A$ for each object A of \mathcal{C} and if $f : A \longrightarrow B$ is a morphism then $Prod(f) = f \times f : A \times A \longrightarrow B \times B$ is covariant functor.

Let $\mathbf{D} = \langle D, \leq \rangle$ be a poset. Then $m(\mathbf{D}) : I(D) \longrightarrow D \times D$ defined by $m(\mathbf{D})([a, b]) = (a, b)$ is a morphism from $I(\mathbf{D})$ into $\mathbf{D} \times \mathbf{D}$. The collection

$$m = \{m(\mathbf{D}) : I(D) \longrightarrow Prod(D) \mid \mathbf{D} \text{ is a poset} \}$$

of morphisms is a natural transformation from $I : \mathbf{POSET} \longrightarrow \mathbf{POSET}$ to $Prod : \mathbf{POSET} \longrightarrow \mathbf{POSET}$.

An **interval category**¹ is a triple $(\mathcal{C}, \mathbb{I}, m)$ such that

1. \mathcal{C} is a category with product
2. $\mathbb{I} : \mathcal{C} \longrightarrow \mathcal{C}$ is a covariant functor such that $\mathbb{I}(A \times B)$ is isomorphic to $\mathbb{I}(A) \times \mathbb{I}(B)$ for all pair of objects A and B of \mathcal{C}
3. m is an injective natural transformation from $\mathbb{I} : \mathcal{C} \longrightarrow \mathcal{C}$ to $Prod : \mathcal{C} \longrightarrow \mathcal{C}$
4. There exists a covariant functor $F : \mathcal{C} \longrightarrow \mathbf{POSET}$ such that for each $A, B \in Obj_{\mathcal{C}}$ and for each $f : A \longrightarrow B$ morphism we have that
 - (a) $F(\mathbb{I}(A)) \cong I(F(A))$,
 - (b) $F(\mathbb{I}(f)) \cong I(F(f))$,
 - (c) $F(A \times A) \cong F(A) \times F(A)$,
 - (d) $F(f \times f) \cong F(f) \times F(f)$ and
 - (e) $F(m(A)) \cong m(F(A))$

3 Lattices

Let $\mathbf{L} = \langle L, \wedge, \vee \rangle$ be an algebraic structure where L is a nonempty set and \wedge and \vee are binary operations. \mathbf{L} is a **lattice**, if for each $x, y, z \in L$

1. $x \wedge y = y \wedge x$ and $x \vee y = y \vee x$
2. $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ and $x \vee (y \vee z) = (x \vee y) \vee z$
3. $x \wedge (x \vee y) = x$ and $x \vee (x \wedge y) = x$

In a lattice $\mathbf{L} = \langle L, \wedge, \vee \rangle$, if there exist two distinguish elements, 0 and 1, such that for each $x \in L$, $x \wedge 1 = x$ and $x \vee 0 = x$ then $\langle L, \wedge, \vee, 1, 0 \rangle$ is said a **bounded lattice**.

Example 1. Some examples of bounded lattices:

1. $\mathbf{L}_{\top} = \langle \{1\}, \wedge, \vee, 1, 1 \rangle$, where $1 \wedge 1 = 1 \vee 1 = 1$.
2. $\mathbf{B} = \langle \mathbb{B}, \wedge, \vee, 1, 0 \rangle$, where $\mathbb{B} = \{0, 1\}$, \wedge and \vee are as in the boolean algebra.
3. $\mathbf{I} = \langle [0, 1], \wedge, \vee, 1, 0 \rangle$, where $x \wedge y = \min\{x, y\}$ and $x \vee y = \max\{x, y\}$.

¹ This definition is weaker than the original in [8] because it considers isomorphism order instead of the equality in the property 4. Nevertheless, as in category theory two isomorphic objects are essentially the same, the above definition seem more reasonable.

4. $\mathbf{N} = \langle \mathbb{N}^\top, \wedge, \vee, \top, 0 \rangle$, where \mathbb{N} is the set of natural numbers and
- (a) $\mathbb{N}^\top = \mathbb{N} \cup \{\top\}$,
 - (b) $x \wedge \top = \top \wedge x = x$ and if $x, y \in \mathbb{N}$ then $x \wedge y = \min\{x, y\}$,
 - (c) $x \vee \top = \top \vee x = \top$ and if $x, y \in \mathbb{N}$ then $x \vee y = \max\{x, y\}$.

As it is well known, each lattice establishes a partial order. Let $\mathbf{L} = \langle L, \wedge, \vee \rangle$ be a lattice. Then $\leq_L \subseteq L \times L$ defined by

$$x \leq_L y \Leftrightarrow x \wedge y = x$$

is a partial order where \wedge coincides with the greatest lower bound (infimum) and \vee with the least upper bound (supremum).

Let $\mathbf{L} = \langle L, \wedge_L, \vee_L, 1_L, 0_L \rangle$ and $\mathbf{M} = \langle M, \wedge_M, \vee_M, 1_M, 0_M \rangle$ be bounded lattices. A function $h : L \longrightarrow M$ is a **lattice homomorphism**² if

1. $h(0_L) = 0_M$,
2. $h(1_L) = 1_M$,
3. for each $x, y \in L$ then
 - (a) $h(x \wedge_L y) = h(x) \wedge_M h(y)$,
 - (b) $h(x \vee_L y) = h(x) \vee_M h(y)$.

Proposition 1. *Let \mathbf{L} and \mathbf{M} be bounded lattices. $h : L \longrightarrow M$ is a lattice homomorphism iff $h(0_L) = 0_M$, $h(1_L) = 1_M$, and h is monotonic w.r.t. the lattice orders.*

Proof. It is a well known fact.

Example 2. Let \mathbf{L} be a bounded lattice. Then for each $\alpha \in (0, 1)$, the function $h_\alpha : [0, 1] \longrightarrow L$ defined by

$$h_\alpha(x) = \begin{cases} 0_L & \text{if } x \leq \alpha \\ 1_L & \text{if } x > \alpha \end{cases}$$

is a lattice homomorphism from \mathbf{I} into \mathbf{L} .

3.1 Operators on Bounded Lattices

Let \mathbf{L} and \mathbf{M} be bounded lattices. The **product** of \mathbf{L} and \mathbf{M} , is $\mathbf{L} \times \mathbf{M} = \langle L \times M, \wedge, \vee, (1_L, 1_M), (0_L, 0_M) \rangle$, where $(x_1, x_2) \wedge (y_1, y_2) = (x_1 \wedge_L y_1, x_2 \wedge_M y_2)$ and $(x_1, x_2) \vee (y_1, y_2) = (x_1 \vee_L y_1, x_2 \vee_M y_2)$, is also a bounded lattice.

Let \mathbf{L} be a bounded lattice. The **interval** of \mathbf{L} , is $\mathbf{IL} = \langle IL, \wedge, \vee, [1, 1], [0, 0] \rangle$ where $IL = \{[\underline{X}, \overline{X}] / \underline{X}, \overline{X} \in L \text{ and } \underline{X} \leq_L \overline{X}\}$, $[\underline{X}, \overline{X}] \wedge [\underline{Y}, \overline{Y}] = [\underline{X} \wedge_L \underline{Y}, \overline{X} \wedge_L \overline{Y}]$ and $[\underline{X}, \overline{X}] \vee [\underline{Y}, \overline{Y}] = [\underline{X} \vee_L \underline{Y}, \overline{X} \vee_L \overline{Y}]$, is also a bounded lattice.

The associated order for this lattice agrees with the product order. That is,

$$[\underline{X}, \overline{X}] \leq [\underline{Y}, \overline{Y}] \text{ iff } \underline{X} \leq_L \underline{Y} \text{ and } \overline{X} \leq_L \overline{Y} \quad (1)$$

This partial order (1) generalizes a partial order used for the first time by Kulisch and Miranker [20] in the interval mathematics context.

Clearly, bounded lattices are closed under product and interval operators.

² In [10] the term “homomorphism” was used for a lattice not necessarily bounded. For homomorphism preserving bottom and top elements, as considered here, Davey and Priestley use the term $\{0, 1\}$ -homomorphism.

4 T-Norms and Automorphisms on Bounded Lattices

Let \mathbf{L} be a bounded lattice. A binary operation T on L is a **triangular norm** on \mathbf{L} , **t-norm** in short, if for each $v, x, y, z \in L$ the following properties are satisfied:

1. *commutativity*: $T(x, y) = T(y, x)$,
2. *associativity*: $T(x, T(y, z)) = T(T(x, y), z)$,
3. *neutral element*: $T(x, 1) = x$ and
4. *monotonicity*: If $y \leq_L z$ then $T(x, y) \leq_L T(x, z)$.

Notice that for the lattice \mathbf{I} in particular, this notion of t-norm coincides with the usual one. The well known Gödel and weak t-norms (also known by minimum and drastic product t-norm [21]) can be generalized for arbitrary bounded lattice in a natural way. In particular, the Gödel t-norm (T_G) coincides with \wedge itself and the weak t-norm is defined by

$$T_W(x, y) = \begin{cases} 0 & \text{if } x \neq 1 \text{ and } y \neq 1 \\ x \wedge y & \text{otherwise} \end{cases}$$

Analogously as we have a partial order on the set of t-norm we can establish a partial order on the set of all t-norms on a bounded lattice. Let T_1 and T_2 be t-norms on a bounded lattice \mathbf{L} . Then T_1 is **weaker** than T_2 or, equivalently, T_2 is **stronger** than T_1 , denoted by $T_1 \leq T_2$ if for each $x, y \in L$, $T_1(x, y) \leq_L T_2(x, y)$.

Proposition 2. *Let T be a t-norm on a bounded lattice \mathbf{L} . Then*

$$T_W \leq T \leq T_G$$

Proof. Similar to the classical result (see for example remark 1.5.(i) in [21]).

Corollary 1. *Let T be a t-norm on a bounded lattice \mathbf{L} . Then $T(x, y) = 1_L$ iff $x = y = 1_L$.*

Proof. Straightforward.

4.1 T-Norm Morphisms

Let T_1 and T_2 be t-norms on the bounded lattices \mathbf{L} and \mathbf{M} , respectively. A lattice homomorphism $\rho : L \rightarrow M$ is a **t-norm morphism** from T_1 into T_2 if for each $x, y \in L$

$$\rho(T_1(x, y)) = T_2(\rho(x), \rho(y)) \quad (2)$$

Straightforward from the fact that ρ is a lattice morphism, ρ is monotonic.

If ρ is bijective then ρ is a **t-norm isomorphism**. When \mathbf{L} and \mathbf{M} are equals, t-norm isomorphisms are called **automorphisms**. In fact, this notion coincides with the usual notion of automorphism when the lattice is \mathbf{I} .

Clearly the composition of two t-norm morphisms is also a t-norm morphism. In fact, let \mathbf{K} , \mathbf{L} and \mathbf{M} be bounded lattices, T_1 , T_2 and T_3 be t-norms on \mathbf{K} , \mathbf{L} , and \mathbf{M} , respectively, and ρ_1 and ρ_2 be morphisms between T_1 into T_2 and between T_2 into T_3 , respectively. So, $T_3(\rho_2 \circ \rho_1(x), \rho_2 \circ \rho_1(y)) = \rho_2(T_2(\rho_1(x), \rho_1(y))) = \rho_2(\rho_1(T_1(x, y)))$.

Since the composition of functions is associative, then the composition of t-norm morphisms is also associative. Notice that for any bounded lattice \mathbf{L} , the identity $Id_L(x) = x$ is an automorphism such that for each t-norm on \mathbf{L} , $Id_L(T(x, y)) = T(Id_L(x), Id_L(y))$.

Thus, considering the t-norm morphism notion as a morphism and t-norms as objects, we have a category, denoted by **T-NORM**.

5 T-NORM as a Cartesian Category

5.1 Terminal Object

Proposition 3. *Let $T_\top : \{(1, 1)\} \longrightarrow \{1\}$ defined by $T_\top(1, 1) = 1$. Then T_\top is a t-norm on the bounded lattice \mathbf{L}_\top .*

Proof. Straightforward.

Proposition 4. *Let T be a t-norm on a bounded lattice \mathbf{L} . Then $\rho_\top : L \rightarrow \{1\}$ defined by $\rho_\top(x) = 1$ is the unique t-norm morphism from T into T_\top .*

Proof. Straightforward.

Thus, T_\top is a terminal object of **T-NORM**.

5.2 Cartesian Product

Proposition 5. *Let T_1 and T_2 be t-norms on bounded lattices \mathbf{L} and \mathbf{M} , respectively. Then $T_1 \times T_2 : (L \times M)^2 \rightarrow L \times M$ defined by*

$$T_1 \times T_2((x_1, x_2), (y_1, y_2)) = (T_1(x_1, y_1), T_2(x_2, y_2))$$

is a t-norm on the bounded lattice $\mathbf{L} \times \mathbf{M}$.

Proof. Straightforward.

Proposition 6. *Let T_1 and T_2 be t-norms on the bounded lattices \mathbf{L} and \mathbf{M} , respectively. Then the usual projections $\pi_1 : L \times M \longrightarrow L$ and $\pi_2 : L \times M \longrightarrow M$ defined by*

$$\pi_1(x, y) = x \text{ and } \pi_2(x, y) = y$$

are t-norm morphisms from $T_1 \times T_2$ into T_1 and T_2 , respectively.

Proof. As it is well known, π_1 and π_2 are lattice morphisms. So, it only remains to prove that it satisfies the equation (2). Let $(x_1, x_2), (y_1, y_2) \in L \times M$. Then

$$\begin{aligned} \pi_1(T_1 \times T_2((x_1, x_2), (y_1, y_2))) &= \pi_1(T_1(x_1, y_1), T_2(x_2, y_2)) \\ &= T_1(x_1, y_1) \\ &= T_1(\pi_1(x_1, x_2), \pi_1(y_1, y_2)) \end{aligned}$$

The case for π_2 is analogous.

Next we will prove that **T-NORM** satisfies the universal property of cartesian product.

Theorem 1. *Let T , T_1 and T_2 be t-norms on the bounded lattices \mathbf{K} , \mathbf{L} and \mathbf{M} , respectively. If ρ_1 and ρ_2 are t-norm morphisms from T into T_1 and T_2 , respectively, then there exists only one t-norm morphism ρ from T into $T_1 \times T_2$ such that the following diagram commutes:*

$$\begin{array}{ccccc}
 & & T & & \\
 & \swarrow \rho_1 & \vdots \rho & \searrow \rho_2 & \\
 T_1 & \xleftarrow{\pi_1} & T_1 \times T_2 & \xrightarrow{\pi_2} & T_2
 \end{array}$$

Proof. Let $\rho : K \longrightarrow L \times M$ be the function:

$$\rho(x) = (\rho_1(x), \rho_2(x))$$

First we will prove that, ρ is a t-norm morphism.

$$\begin{aligned}
 \rho(T(x, y)) &= (\rho_1(T(x, y)), \rho_2(T(x, y))) \\
 &= (T_1(\rho_1(x), \rho_1(y)), T_2(\rho_2(x), \rho_2(y))) \\
 &= T_1 \times T_2((\rho_1(x), \rho_2(x)), (\rho_1(y), \rho_2(y))) \\
 &= T_1 \times T_2(\rho(x), \rho(y))
 \end{aligned}$$

Suppose that $\rho' : K \longrightarrow L \times M$ is a t-norm morphism which commutes the diagram. Then, $\pi_1(\rho'(T(x, y))) = \rho_1(T(x, y))$ and $\pi_2(\rho'(T(x, y))) = \rho_2(T(x, y))$.

So, $\rho'(T(x, y)) = (\rho_1(T(x, y)), \rho_2(T(x, y))) = \rho(T(x, y))$. Therefore, ρ is the unique t-norm morphism commuting the above diagram.

Therefore, we can claim that **T-NORM** is a cartesian category.

6 T-NORM as an Interval Category

Interval t-norms have been widely studied in the unit lattice (see for example [32,14,5]) as well as in certain classes of lattice (see for example [9,26,4]).

6.1 The Functor \mathbb{I}

Since the interval constructor is closed on the bounded lattices, the t-norm notion on bounded lattice is sufficient, however, here we see how to transform an arbitrary t-norm on a bounded lattice into a t-norm on its interval bounded lattice.

Proposition 7. *Let T be a t-norm on the bounded lattice \mathbf{L} . Then $\mathbb{I}[T] : IL^2 \longrightarrow IL$ defined by*

$$\mathbb{I}[T](X, Y) = [T(\underline{X}, \underline{Y}), T(\overline{X}, \overline{Y})]$$

is a t-norm on the bounded lattice $\mathbb{I}\mathbf{L}$.

Proof. Commutativity, monotonicity and neutral element $([1, 1])$ properties of $\mathbb{I}[T]$ follow straightforward from the same properties of T . The associativity requests a bit of attention.

$$\begin{aligned}\mathbb{I}[T](X, \mathbb{I}[T](Y, Z)) &= \mathbb{I}[T](X, [T(\underline{Y}, \underline{Z}), T(\overline{Y}, \overline{Z})]) \\ &= [T(\underline{X}, T(\underline{Y}, \underline{Z})), T(\overline{X}, T(\overline{Y}, \overline{Z}))] \\ &= [T(T(\underline{X}, \underline{Y}), \underline{Z}), T(T(\overline{X}, \overline{Y}), \overline{Z})] \\ &= \mathbb{I}[T]([T(\underline{X}, \underline{Y}), T(\overline{X}, \overline{Y})], Z) \\ &= \mathbb{I}[T](\mathbb{I}[T](X, Y), Z).\end{aligned}$$

Proposition 8. Let T be a t-norm on the bounded lattice \mathbf{L} . Then the projections $l : \mathbb{I}L \longrightarrow L$ and $r : \mathbb{I}L \longrightarrow M$ defined by

$$l(X) = \underline{X} \text{ and } r(X) = \overline{X}$$

are t-norm morphisms from $\mathbb{I}[T]$ into T .

Proof. $l(\mathbb{I}[T](X, Y)) = \mathbb{I}[T](X, Y) = [T(\underline{X}, \underline{Y}), T(\overline{X}, \overline{Y})] = T(\underline{X}, \underline{Y}) = T(l(X), l(Y))$

So, l and, by analogy, r are t-norm morphisms.

Proposition 9. Let T_1 and T_2 be t-norms on the bounded lattices \mathbf{L} and \mathbf{M} , respectively. If ρ is a t-norm morphism from T_1 into T_2 , then there exists a unique t-norm morphism $\mathbb{I}[\rho]$ from $\mathbb{I}[T_1]$ into $\mathbb{I}[T_2]$ such that the following diagram commutes:

$$\begin{array}{ccccc} T_1 & \xleftarrow{l} & \mathbb{I}[T_1] & \xrightarrow{r} & T_1 \\ \rho \downarrow & & \mathbb{I}[\rho] \downarrow & & \rho \downarrow \\ T_2 & \xleftarrow{l} & \mathbb{I}[T_2] & \xrightarrow{r} & T_2 \end{array}$$

Proof. Let $\mathbb{I}[\rho] : \mathbb{I}L_1 \longrightarrow \mathbb{I}L_2$ defined by

$$\mathbb{I}[\rho](X) = [\rho(l(X)), \rho(r(X))].$$

Since,

$$\begin{aligned}\mathbb{I}[\rho](\mathbb{I}[T_1](X, Y)) &= [\rho(l(\mathbb{I}[T_1](X, Y))), \rho(r(\mathbb{I}[T_1](X, Y)))] \\ &= [\rho(l([T_1(\underline{X}, \underline{Y}), T_1(\overline{X}, \overline{Y})])), \rho(r([T_1(\underline{X}, \underline{Y}), T_1(\overline{X}, \overline{Y})]))] \\ &= [\rho(T_1(\underline{X}, \underline{Y})), \rho(T_1(\overline{X}, \overline{Y}))] \\ &= [T_2(\rho(\underline{X}), \rho(\underline{Y})), T_2(\rho(\overline{X}), \rho(\overline{Y}))] \\ &= \mathbb{I}[T_2](\mathbb{I}[\rho](X), \mathbb{I}[\rho](Y)).\end{aligned}$$

$\mathbb{I}[\rho]$ is a t-norm morphism from $\mathbb{I}[T_1]$ into $\mathbb{I}[T_2]$.

Now suppose that ρ' is a t-norm morphism from $\mathbb{I}[T_1]$ into $\mathbb{I}[T_2]$ which commutes the diagram above. Then

$$l(\rho'(X)) = \rho(l(X)) \text{ and } r(\rho'(X)) = \rho(r(X)).$$

Therefore, $\rho'(X) = [l(\rho'(X)), r(\rho'(X))] = [\rho(l(X)), \rho(r(X))] = \mathbb{I}[\rho](X)$.

So \mathbb{I} is a covariant functor from **T-NORM** into **T-NORM**.

6.2 The Natural Transformation m

Let T be a t-norm on a bounded lattice \mathbf{L} . Define $m_T : \mathbb{I}[T] \longrightarrow \text{Prod}(T)$ by

$$m_T(\mathbb{I}[T](X, Y)) = T \times T((\underline{X}, \underline{Y}), (\overline{X}, \overline{Y}))$$

where $\text{Prod}(T) = T \times T$.

Proposition 10. *Let T_1 and T_2 be t-norms on bounded lattices \mathbf{L}_1 and \mathbf{L}_2 , respectively. Let $\rho : T_1 \longrightarrow T_2$ be a t-norm morphism. Then the following diagram*

$$\begin{array}{ccc} \mathbb{I}[T_1] & \xrightarrow{m_{T_1}} & T_1 \times T_1 \\ \mathbb{I}[\rho] \downarrow & & \downarrow \text{Prod}(\rho) \\ \mathbb{I}[T_2] & \xrightarrow{m_{T_2}} & T_2 \times T_2 \end{array}$$

commutes.

Proof. Let $X, Y \in IL_1$. Then

$$\begin{aligned} \text{Prod}(\rho)(m_{T_1}(\mathbb{I}[T_1](X, Y))) &= \text{Prod}(\rho)(T_1 \times T_1((\underline{X}, \underline{Y}), (\overline{X}, \overline{Y}))) \\ &= \rho \times \rho(T_1(\underline{X}, \underline{Y}), T_1(\overline{X}, \overline{Y})) \\ &= (\rho(T_1(\underline{X}, \underline{Y})), \rho(T_1(\overline{X}, \overline{Y}))) \\ &= (T_2(\rho(\underline{X}), \rho(\underline{Y})), T_2(\rho(\overline{X}), \rho(\overline{Y}))) \end{aligned}$$

On the other hand, by Proposition 9,

$$\begin{aligned} m_{T_2}(\mathbb{I}[\rho](\mathbb{I}[T_1](X, Y))) &= m_{T_2}(\mathbb{I}[T_2](\mathbb{I}[\rho](X), \mathbb{I}[\rho](Y))) \\ &= T_2 \times T_2((\underline{\mathbb{I}[\rho](X)}, \underline{\mathbb{I}[\rho](Y)}), (\overline{\mathbb{I}[\rho](X)}, \overline{\mathbb{I}[\rho](Y)})) \\ &= (T_2(\rho(\underline{X}), \rho(\underline{Y})), T_2(\rho(\overline{X}), \rho(\overline{Y}))) \end{aligned}$$

So, the above diagram commutes.

The above proposition proves that m is a natural transformation from the functor \mathbb{I} to the functor Prod .

7 The Functor F

Let T be a t-norm on a bounded lattice \mathbf{L} . Define the poset $F(T) = \langle L^T, \leq_T \rangle$ where

$$L^T = \{(x, y, z) \in L^3 / T(x, y) = z\} \quad (3)$$

$$(x_1, y_1, z_1) \leq_T (x_2, y_2, z_2) \text{ iff } x_1 \leq_L x_2 \text{ and } y_1 \leq_L y_2 \text{ and } z_1 \leq_L z_2 \quad (4)$$

On the other hand, let T_1 and T_2 be t-norms on the bounded lattice \mathbf{L}_1 and \mathbf{L}_2 , respectively. Let ρ be a t-norm morphism from T_1 into T_2 . Then $F(\rho) : L^{T_1} \longrightarrow L^{T_2}$ defined by

$$F(\rho)(x, y, z) = (\rho(x), \rho(y), T_2(\rho(x), \rho(y))) \quad (5)$$

Proposition 11. *let T_1 and T_2 be t-norms on the bounded lattices \mathbf{L}_1 and \mathbf{L}_2 , respectively. Let ρ be a t-norm morphism from T_1 into T_2 . Then $F(\rho)$ is monotonic, i.e.,*

$$\text{If } (x_1, y_1, z_1) \leq_{T_1} (x_2, y_2, z_2) \text{ then } F(\rho)(x_1, y_1, z_1) \leq_{T_2} F(\rho)(x_2, y_2, z_2)$$

Proof. If $(x_1, y_1, z_1) \leq_{T_1} (x_2, y_2, z_2)$ then by equation (4) and monotonicity of ρ and T_2 , $(\rho(x_1), \rho(y_1), T_2(\rho(x_1), \rho(y_1))) \leq_{T_2} (\rho(x_2), \rho(y_2), T_2(\rho(x_2), \rho(y_2)))$. Therefore, by equation (5), $F(\rho)(x_1, y_1, z_1) \leq_{T_2} F(\rho)(x_2, y_2, z_2)$.

The above proposition guarantees that $F(\rho)$ is a morphism from the poset $F(T_1)$ to the poset $F(T_2)$. Thus, $F : \mathbf{T-NORM} \rightarrow \mathbf{POSET}$ is a covariant functor.

7.1 The Interval Category

Theorem 2. *$\langle \mathbf{T-NORM}, \mathbb{I}, m \rangle$ is an interval category.*

Proof. Properties 1 to 3 of interval category definition follows straightforward from the discussions in this section and section 5. The property 4 follows straightforward from the definition of F , m and \mathbb{I} . We will only show that $F(\mathbb{I}[T]) \cong I(F(T))$ for each t-norm T on a bounded lattice \mathbf{L} . First we will prove that $IL^{\mathbb{I}[T]} \cong I(L^T)$.

Because of that it is sufficient to note that the function $i : IL^{\mathbb{I}[T]} \rightarrow I(L^T)$, defined by

$$i(X, Y, Z) = [(\underline{X}, \underline{Y}, \underline{Z}), (\overline{X}, \overline{Y}, \overline{Z})]$$

is an isomorphism, i.e. is bijective and mononotonic. By monotonicity of T , clearly $\underline{Z} = T(\underline{X}, \underline{Y})$ and so i is well defined. The bijectivity of i is clear and the monotonicity follows from the interval and cartesian product order.

8 Final Remarks

This is an introductory paper which considers a well known generalization of the t-norm notion for arbitrary bounded lattices and introduces a generalization of the automorphism notion for t-norms on arbitrary bounded lattices, named t-norm morphisms. With these two generalizations we can consider a rich category having t-norms as objects and t-norm morphism as morphism. We then prove that this category joint with a natural transformation here introduced and the interval constructor on bounded lattice t-norms and t-norm morphisms are an interval category in the sense of [8].

There are several works on categorical or topos-theoretic treatments of fuzzy sets, e.g. [13,15,16,19,29], but none of them consider the t-norms as category and the interval constructor. The choice of bounded lattices as subjacent structure instead of complete Heyting algebras, for example, is due to the generality of the first.

As further works we intend to show other category properties of $\mathbf{T-NORM}$, for example that it is a bi-Cartesian closed category. We also hope to extend for bounded lattices other usual notions of fuzzy theory, such as t-conorms, implications, negations, additive generators, copulas, etc. and see them as categories.

Acknowledgements

Paul Taylor commutative diagram package was used in this paper. The authors are grateful to the reviewers of the paper by their comments and suggestions. This work was supported in part by the Brazilian Research Council (CNPq) under the process number 470871/2004-0.

References

1. Alsina, C., Trillas, E., Valverde, L.: On non-distributive logical connectives for fuzzy set theory. *Busefal* 3, 18–29 (1980)
2. Asperti, A., Longo, G.: *Categories, Types and Structures: An introductionn to category theory for the working computer scientist*. MIT Press, Cambridge (1991)
3. Barr, M., Well, C.: *Category Theory for Computing Scientist*. Prentice Hall International (UK) Ltda, Englewood Cliffs (1990)
4. Bedregal, B.C., Santos, H.S., Callejas-Bedregal, R.: T-Norms on Bounded Lattices: t-norm morphisms and operators. In: *IEEE Int. Conference on Fuzzy Systems*, Vancouver, July 16–21, 2006, pp. 22–28. IEEE Computer Society Press, Los Alamitos (2006)
5. Bedregal, B.C., Takahashi, A.: The best interval representation of t-norms and automorphisms. *Fuzzy Sets and Systems* 157, 3220–3230 (2006)
6. Bellman, R.E., Zadeh, L.A.: Decision-making in a fuzzy environment. *Management Science* 17, B141–B164 (1970)
7. Bustince, H., Burillo, P., Soria, F.: Automorphisms, negations and implication operators. *Fuzzy Sets and Systems* 134, 209–229 (2003)
8. Callejas-Bedregal, R., Bedregal, B.C.: Interval categories. In: de Janeiro, R. (ed.) *Proceedings of IV Workshop on Formal Methods*, pp. 139–150 (2001) (available in <http://www.dimap.ufrn.br/~bedregal/main-publications.html>)
9. Cornelis, G., Deschrijver, G., Kerre, E.E.: Advances and Challenges in Interval-Valued Fuzzy Logic. *Fuzzy Sets and Systems* 157, 622–627 (2006)
10. Davey, B.A., Priestley, H.A.: *Introduction to Lattices and Order*. Cambridge University Press, Cambridge (2002)
11. De Baets, B., Mesiar, R.: Triangular norms on product lattices. *Fuzzy Sets and Systems* 104, 61–75 (1999)
12. De Cooman, G., Kerre, E.E.: Order norms on bounded partially ordered sets. *Journal Fuzzy Mathematics* 2, 281–310 (1994)
13. Eytan, M.: Fuzzy Sets: A topos-logical point of view. *Fuzzy Sets and Systems* 5, 47–67 (1981)
14. Gehrke, M., Walker, C., Walker, E.: De Morgan systems on the unit interval. *International Journal of Intelligent Systems* 11, 733–750 (1996)
15. Goguen, J.: L-fuzzy sets. *Journal of Mathematics Analisis and Applications* 18, 145–174 (1967)
16. Gottwald, S.: Universes of Fuzzy Sets and Axiomatizations of Fuzzy Set Theory. Part II: Category Theoretic Approaches. *Studia Logica* 84, 23–50 (2006)
17. Grätzer, G.: *General Lattice Theory*. Academic Press, New York (1978)
18. Hájek, P.: *Metamathematics of Fuzzy Logic*. Kluwer Academic Publisher, Dordrecht (1998)
19. Höhle, U.: GL-quantales: Q-valued sets and their singletons. *Studia Logica* 61, 123–148 (1998)
20. Kulisch, U., Miranker, W.: *Computer Arithmetic in Theory and Practice*. Academic Press, London (1981)

21. Klement, E.P., Mesiar, R., Pap, E.: *Triangular Norms*. Kluwer academic publisher, Dordrecht (2000)
22. Klement, E.P., Mesiar, R.: *Semigroups and Triangular Norms*. In: Klement, E.P., Mesiar, R. (eds.) *Logical, Algebraic, and Probabilistic Aspects of Triangular Norms*, Elsevier, Amsterdam (2005)
23. Menger, K.: Statical metrics. *Proc. Nat. Acad. Sci* 37, 535–537 (1942)
24. Ray, S.: Representation of a Boolean algebra by its triangular norms. *Mathware & Soft Computing* 4, 63–68 (1997)
25. Schweizer, B., Sklar, A.: Associative functions and statistical triangle inequalities. *Publ. Math. Debrecen* 8, 169–186 (1961)
26. Van Gasse, B., Cornelis, G., Deschrijver, G., Kerre, E.E.: On the Properties of a Generalized Class of t-Norms in Interval-Valued Fuzzy Logics. *New Mathematics and Natural Computation* 2, 29–42 (2006)
27. Walters, R.F.C.: *Categories and Computer Sciences*. Cambridge University Press, Cambridge (1991)
28. Wang, Z.D., Yu, Y.D.: Pseudo t-norms an implication operators on a complete Brouwerian lattice. *Fuzzy Sets and Systems* 132, 113–124 (2002)
29. Wyler, O.: *Lecture Notes on Topoi and Quasitopoi*. World Scientific, Singapore (1991)
30. Yager, R.R.: An approach to inference in approximate reasoning. *International Journal on Man-Machine Studies* 13, 323–338 (1980)
31. Zadeh, L.A.: Fuzzy sets. *Information and Control* 8, 338–353 (1965)
32. Zuo, Q.: Description of strictly monotonic interval and/or operations. In: *APIC'S Proceedings: International Workshop on Applications of Interval Computations*, pp. 232–235 (1995)

Towards Systematic Analysis of Theorem Provers Search Spaces: First Steps

Hicham Bensaid¹, Ricardo Caferra², and Nicolas Peltier²

¹ INPT

Avenue Allal Al Fassi, Madinat Al Irfane, Rabat, Morocco,

bensaid@inpt.ac.ma

² LIG INPG/CNRS

46, avenue Félix Viallet

38031 Grenoble Cedex, France

Ricardo.Caferra@imag.fr, Nicolas.Peltier@imag.fr

Abstract. Being able to automatically analyze the search spaces of automated theorem provers is of the greatest importance. A method is proposed to do that. It is inspired by *inductive* tasks (e.g. *discovering* in mathematics and natural sciences). The key idea is to replace implicit *genetic* descriptions of the consequences in search spaces (i.e. the problem specification) by *structural* ones (describing the *form* of such consequences). The approach profits from the expressive power of term schematization languages and from the capabilities offered by existing powerful symbolic computation systems (Mathematica,...). A running software based on these ideas show evidence of the adequacy of the approach.

1 Introduction

Since the historical beginning of Automated Deduction (AD), the problem of handling overwhelming sets of useless generated formulas (clauses), and consequently the importance of the analysis of search spaces were recognized as *crucial* in order to make theorem provers able to deal with non ad hoc problems (see e.g. [18]). The main idea behind the present work is to investigate methods allowing to analyze *dynamically* theorem provers search spaces. To do so we look for *regularities* in the set of logical consequences produced by automated theorem provers (ATPs). This problem is in some sense related to the one attacked in other domains (automated learning, inductive logic programming,...): produce a short description, or a general law, explaining (or describing) a set of (so called) positive examples and excluding possible (so called) negative ones. Here the pioneering work by Plotkin [14,15] that brought inductive generalization into reasoning systems should be mentioned. What we are looking for is a *structural description* of a given set of consequences (depending on their rank in a list) instead of its *genetic* (see [11]) description, i.e. a description assimilable to a grammar (e. g. a set of clauses). Admittedly, a structural description of the *full* search only exists for sets of formulas belonging to decidable classes. But

this theoretical limit should not prevent us from studying particular cases of the problem (e.g. describing subsets of search spaces).

Three kinds of problems in which such capabilities are clearly useful (of course the list is not limitative) are:

- The first one is immediate. Given a recursively defined list of non-structured objects obtain a formula specifying them in function of their rank in the list. For instance, in Section 4.1, we show how to compute the rank of a given pair (x, y) in the usual enumeration of rational numbers. The other two are based on it:
- Theorem provers yield structured objects (terms, formulas) and combines them. Their combination give patterns that can be studied via non-structured objects (typically integers powers, e.g. $f(\dots)$, $f^3(\dots)$, $f^7(\dots)$, \dots). If a law capturing structurally the whole set of consequences of a formula is discovered, it can be decided if it will contain or not contradictions thus allowing to conclude to the satisfiability of such a formula and (in some cases) to specify (possibly infinite) models.
- It is well known that introducing lemmas can strongly (sometimes non elementarily) decrease the lengths of proofs, thus transforming cumbersome proofs into relatively simple, or at least readable, ones. The capability of describing (abstracting from, generalizing from) a (necessarily finite) particular cases may be most useful in proposing lemmata, specially in proof by induction. The example chosen here follow the so called “induction-inductionless” (see below) approach to proof by induction.

Our approach benefits from the huge amount of existing work on finding general forms of sequences of integers (see for instance <http://www.research.att.com/~njas/sequences/>). Two points deserve to be emphasized. The first one is that we are not, of course, interested in competing with Mathematica or other excellent symbolic computation tools. We are using them in order to *add new capabilities* to ATP. The second one is that high complexity is not, in principle, crippling for us: our main aim *is not proving* but *structuring* search spaces, thus obtaining useful information when theorem provers alone fail or are useless.

In this paper, we apply these ideas in the context of resolution-based theorem proving (see Section 3.1). Although resolution is less powerful from a theoretical point of view than full logical calculi such as natural deduction or sequent calculi with cut, it has the advantage that powerful, *purely automatic* systems are available. We describe a tool able to induce in a pure automatic way structural descriptions of sequences of clauses and we show some (hopefully promising) examples of applications, mainly in satisfiability detection (model building and proof by consistency).

2 The Description Language

We first introduce the language used to express the regularities we are looking for and to structurally describe sets of clauses.

First-order terms are of course not expressive enough. Fortunately, languages have been defined for denoting sequences of structurally similar terms. They are called *term schematization languages* and have been thoroughly studied in the past decades [5,8,10].

In the context of resolution-based approaches, term schematizations allow to denote infinite sequences of structurally similar clauses, which are frequently obtained by applying repeatedly the same axiom or sequence of axioms. In this work, we use the language of terms with integer exponents (*I-terms*), which has the advantage to be simple to use, very natural, and expressive enough to denote several interesting sequences arising in practice. In this section we recall some of the definitions, in order to make the paper self-contained.

Let Σ be a set of function symbols containing at least one constant, together with an arity function ar . Let X be an infinite set of variables. \diamond is a symbol of arity 0, called *hole*. V_N is an infinite set of symbols denoting *integer variables*. A denotes the set of *arithmetic expressions* that can be built on V_N (i.e. the set of terms built on V_N , using the constant 0 and the function symbols s and $+$).

Definition 1. *The set T of terms with integer exponents (*I-terms*) and the set T_\diamond of contexts (or terms with a hole) are the least sets that satisfy the following properties:*

$$\begin{aligned} f(s_1, \dots, s_k) \in T &\Leftarrow (s_1, \dots, s_k) \in T^{ar(f)} \\ X \subset T, \diamond \in T_\diamond \\ f(s_1, \dots, s_k) \in T_\diamond &\Leftarrow k = ar(f) \wedge \forall i \in [1..k], s_i \in T \cup T_\diamond \wedge (\exists j \in [1..k]) s_j \in T_\diamond \\ s^n.t \in T &\Leftarrow s \in T_\diamond \wedge t \in T \wedge n \in A \wedge s \neq \diamond \end{aligned}$$

For instance, the term $t = f(x, g(\diamond))^n.a$ is an *I-term*. Instances of t are: $f(x, g(a))$ for $n = 1$, $f(x, g(f(x, g(a))))$, for $n = 2, \dots$

In [8] a unification algorithm for *I-terms* are proposed. In [12], it is proven that the first-order equational theory of *I-terms*, including disjunction, negation and existential quantifiers, but using *syntactic* equality as the only predicate symbol, is actually decidable. Problems such as the equivalence problem (given two sets of *I-terms* T and S , check whether T and S denote the same set of ground terms), inclusion problem (check whether the set of terms denoted by T is included into the one denoted by S), the complement problem (describe the terms not denoted by T) can be formalized as first-order equational formulae on *I-terms*, thus are decidable by this result.

3 A New Tool for Analyzing Search Spaces

We use the expressive power of *I-terms* in order to automatically compute – when possible – a structural description of the sequences of standard terms that are generated during proof search. We *only* use for this purpose the information contained in the set of deduced clauses (the proof procedure itself is not taken into account).

The proposed ideas would have been just a jeu d’esprit without a tool implementing them. We have therefore developed a software based in the ideas

presented in section 3.1. It is termed DSSS or DS3 (**D**escribing **S**tructurally **S**earch **S**paces) and is implemented in JAVA.

3.1 The Algorithm

Our algorithm works on three passes: a *syntactic parsing* associated with simplification, a *classification* of (simplified) clauses and a *generalization* of these classes (using an external symbolic computation tool).

Resolution-based Prover. The *input* of our algorithm is the *output* of the theorem prover. Presently only resolution-based provers are considered, but other inference rules such as paramodulation, superposition etc. are taken into account. DS3 can deal with terms and clauses produced by (basic) resolution and equality-handling rules such as superposition.

Thus the input of our generalization algorithm is the set of clauses that are produced by the resolution procedure.

We use currently the theorem prover *E* [16]. Other resolution-based theorem provers could of course be used instead, with little programming effort.

Simplification: From Trees to *I*-Terms. The first operation consists in compacting terms of the form $\underbrace{f(f(f(\dots(f(a))))}_{n \text{ times}}$ in order to represent them as

$f(\diamond)^n.a$ for example. The aim is to identify patterns (represented as contexts) that are iterated in the considered terms and clauses.

One can view this process as a systematic (non deterministic) application of the following rules:

- **Context Creation.** $t \rightarrow t[\diamond]^1.s$ where s is a subterm of t such that $s \neq t$ and $t[\diamond]$ denotes the context obtained by replacing any occurrence of s in t by \diamond .
- **Normalization.** $t^n.t^m.s \rightarrow t^{n+m}.s$. This rule allows one firstly to provide a more compact representation of the clause set and secondly to detect potential sequences.

For instance, the term $f(a, f(a, f(a, b)))$ is rewritten by the first rule into $f(a, \diamond)^1.(f(a, \diamond)^1.(f(a, \diamond)^1.b))$. Using the second rule we obtain $f(a, \diamond)^3.b$.

Clearly, the Context Creation rule can be applied on any term t and any position p , yielding a potentially huge number of different contexts. In order to prune the search space, it is only applied on *iterated* contexts, i.e. when the Normalization rule becomes immediately applicable on the obtained term. Terms are analyzed bottom-up (i.e. the two rules above are applied using an *innermost* rewriting strategy). *Currently, our implementation is restricted to contexts in which the hole occurs at depth 1* (the depth of the context itself is not bounded). This restriction is clearly one of the present limits of our approach, but it makes the analysis of huge sets of clauses realistic (in particular, no backtracking is needed when applying the two rules above). For more complex contexts, heuristics are needed to prune the search space. Since only contexts of depth 1 are

considered, all the obtained I -terms must be of the form $f(t_1, \dots, t_n)^n$ where f is a function symbol, and for any $i \in [1..n]$ t_i is either a hole \diamond or an I -term.

Classifying Clauses. At the end of the previous step, the terms occurring in the clauses are represented using I -terms (but without integer variables). Then, the clauses can be classified according to the common contexts that have been identified. In order to explain this properly, we need to introduce some further definitions.

Definition 2. *For any integer l , the set C of l -term classes and the set C_\diamond of l -context classes are inductively defined as follows:*

$$\begin{aligned} f(s_1, \dots, s_k) \in C &\Leftarrow (s_1, \dots, s_k) \in C^{ar(f)} \\ X \in C, \diamond \in C_\diamond \\ f(s_1, \dots, s_k) \in C_\diamond &\Leftarrow k = ar(f) \wedge \forall i \in [1..k], s_i \in C \cup C_\diamond \wedge (\exists j \in [1..k]) s_j \in C_\diamond \\ s^{n_1, \dots, n_l}.t \in C &\Leftarrow s \in C_\diamond \wedge t \in C \wedge n_1, \dots, n_k \in A \wedge s \neq \diamond \end{aligned}$$

Obviously, any I -term is also a 1-term class. Let l be an integer and let $i \leq l$. A term t is said to be an i -instance of a l -term class \hat{t} if one of the following conditions holds:

- $t = \hat{t}$.
- $t = f(t_1, \dots, t_n)$, $\hat{t} = f(\hat{t}_1, \dots, \hat{t}_n)$ and for any $j = 1, \dots, k$, t_j is a i -instance of \hat{t}_i .
- $\hat{t} = f(\hat{t}_1, \dots, \hat{t}_n)^{n_1, \dots, n_l}.\hat{s}$ and $t = f(t_1, \dots, t_n)^{n_i}.s$ where t_1, \dots, t_n, s are i -instances of $\hat{t}_1, \dots, \hat{t}_n, \hat{s}$ respectively.

The set of i -instances of a term class \hat{t} is denoted by $I(\hat{t})$. In order to classify the obtained I -terms, we need an algorithm that, given a term class \hat{t} and a term s , finds a new term class, denoted by $\hat{t} \star s$, s.t. $I(\hat{t}) \cup \{s\} = I(\hat{t} \star s)$. Of course, this term class does not necessarily exist. If it exists, it is computed as follows (if N is a sequence and m an integer, $N.m$ denotes the concatenation of N and m).

- $\hat{t} \star s \stackrel{\text{def}}{=} f(\hat{s}_1, \dots, \hat{s}_n)^{(N.m)}.\hat{v}$ if $\hat{t} = f(\hat{t}_1, \dots, \hat{t}_n)^N.\hat{u}$, $s = f(s_1, \dots, s_n)^m.v$, and where for any $i \in [1..n]$, $\hat{s}_i = \hat{t}_i \star s_i$ and $\hat{v} = \hat{u} \star v$.
- $\hat{t} \star s \stackrel{\text{def}}{=} f(\hat{s}_1, \dots, \hat{s}_n)$ if $\hat{t} = f(\hat{t}_1, \dots, \hat{t}_n)$, $s = f(s_1, \dots, s_n)$ and $\hat{s}_i = \hat{t}_i \star s_i$, for any $i \in [1..n]$.
- Otherwise, $\hat{t} \star s$ does not exist.

It is easy to show that if there exists \hat{s} s.t. $\hat{s} = \hat{t} \star s$, then \hat{s} must be a k -term class (for some integer k) and we have $I(\hat{t}) \cup \{s\} = I(\hat{s})$.

Literals and clauses are handled in a similar way (of course for clauses the rule should be applied modulo the AC properties of \vee). Terms corresponding to the same class (modulo a renaming of integer variables) can be regrouped. The algorithm classifying the clauses is depicted below (where Δ denotes the set of clause classes).

Algorithm 1. The classification algorithm

```

for each clause  $C$  (after transformation to  $I$ -term form using the rules
above)
  for each clause class  $\hat{C}$  in  $\Delta$ 
    if  $\hat{C}' = \hat{C} \star C$  exists
       $\Delta := \Delta \setminus \{\hat{C}\} \cup \{\hat{C}'\}$ 
    end if
  done
  if no class is found
     $\Delta := \Delta \cup \{C\}$ 
  end if
done

```

The Generalization Pass. The last pass aims at analyzing the integer sequences contained in the obtained classes in order to find a mathematical formula generalizing each sequence occurring in a given term.

We currently restrict the type of sequence generalization to polynomial sequences (which are the ones occurring in practice). A lot of work has been done in the area of finding the general term of a sequence given its first terms. The OEIS <http://www.research.att.com/~njas/sequences/> web site provides more than 100000 sequences. To find the polynomial generalizing a sequence we use the so called binomial transformation [2]. A Mathematica implementation of this transformation (and many other) can be found in this link <http://www.research.att.com/~njas/sequences/seqtranslib.txt>

An important problem remains unsolved: it is sometimes necessary to separate “imbricated” sequences. Indeed, there may exist lists of integers Σ such that we can not find a general description of the elements in Σ , but s.t. if we decompose Σ into $\Sigma_1, \dots, \Sigma_k$: $\Sigma = \Sigma_1 \cup \dots \cup \Sigma_k$ then a general sequence for each Σ_i exists. The problem then is that this decomposition must be the same for all the integer sequences of a given class. This situation can occur if the clauses of a given class are not all generated by the same rule pattern. Considering the information given by the ATP about the rules used to generate each clause helps to overcome this problem.

Our generalization algorithm is “sound” in the sense that the obtained I -terms are always (strict) generalizations of the sequences of terms generated by the prover. But, of course, since we use an *inductive* approach, we cannot ensure that the obtained formulae are logical consequences of the initial one. They must be treated as lemmata, which may be useful e.g. to avoid divergence. Thus some a posteriori verification step is needed. Currently, this verification step is not integrated to our tool.

3.2 Comparison with Existing Approaches

To the best of our knowledge there are neither approaches nor tools based on the same ideas as those presented in this work. The approaches that have originally been proposed to generate automatically I -terms from sequences of standard terms mostly rely on the detection of *self-resolving clauses* e.g. $\neg p(x) \vee p(f(x))$,

which obviously entail a trivial “cycle”. In many cases, these cycles can be detected using simple syntactic criteria, and the corresponding I -terms can be introduced to describe the whole set of deducible clauses (here $\neg p(x) \vee p(f^n(x))$). In [13] a more general approach is proposed, using a sophisticated analysis on the proof tree in order to detect more complex cycles that are not explicitly generated by the proof procedure.

The approach we investigate in this paper is different: *it is completely independent on the considered proof procedure and uses inductive generalization techniques instead of deduction*. Consequently, the generated sequences are *not* in general logical consequence of the axioms, they can be viewed as useful “conjectures”. Thus a verification step is needed. But on the other hand the possibility of getting more information than that conveyed by logical consequences of the axioms is often useful either to detect satisfiability or to suggest useful lemmata (see Section 4.3).

A work that can be compared in spirit with ours is that of [1] in which a generalization algorithm à la Plotkin [15] is proposed in order to obtain an I -term generalizing sets of standard terms. Their approach is essentially based on an enumeration of the set of I -terms, guided by the form of the considered set of terms. Disunification [6,12] is used to discard non minimal generalizations. Iterated I -terms are introduced as follows: given two terms t, s one has first to decompose t and s as iterated I -terms: $t = u^n.v$ and $s = w^m.z$, where u, v, w, s are terms and v, z are integers (greater than 1). Then, the authors try to find a generalization of u, w and v, z respectively. As far as we know, there is no implementation of this approach.

This algorithm is suitable for small sets of terms, but when huge sequences are considered (as in our case) checking all possible decompositions of each term will become exceedingly costly. Using, as in our work, a generalization algorithm based on symbolic computation tools has three advantages: firstly, only minimal decompositions need to be considered¹. Secondly, finer characterizations can be obtained because we are not restricted to *linear* sequences (see Section 4.1). Third, it is very hard to isolate in the whole set of generated clauses, the ones corresponding to a particular sequence. Applying the generalization algorithm of [1] to the whole set of clauses would not yield any relevant generalization (other than a variable). Our approach allows to overcome this limitation. On the other hand, our approach has the drawback that only one integer variable can be introduced at a given point (since we only handle *sequences* of terms). This variable corresponds to a particular “direction” in the search space. Several variables can of course be introduced by repeated applications of our algorithm (see Section 4.1).

Our approach is related to the *divergence critic* developed in [17] to avoid divergence in induction proofs. As DS3, the divergence critic is independent of the proof procedure. It uses clever heuristics based on rippling in order to

¹ We do not need to consider contexts that can be themselves decomposed as an iteration of a smaller context: $t[\underbrace{t[\dots[\diamond]\dots]}_n]$.

generate general descriptions of sequences of inferred terms. The essential difference with our tool is that the formalism we use to describe the search space (term schematization) is more expressive than first-order terms. *Thus more precise generalizations can be obtained.*

4 Case Studies

4.1 Characterizing Sequences

The first example is only intended to illustrate how our approach works. We consider the following set of Horn clauses, which corresponds to the usual enumeration of rational numbers (i.e. of the pairs (x, y)).

$$\{p(s(0), s(0), s(0)), p(s(0), s(x), s(n)) \vee \neg p(x, s(0), n), \\ p(s(x), y, s(n)) \vee \neg p(x, s(y), n)\}.$$

$p(x, y, z)$ means that $\frac{x}{y}$ is the z -th rational number. Using unit positive resolution (and/or hyper-resolution) a prover generates an infinite number of clauses of the form $p(1, 1, 1), p(1, 2, 2), p(2, 1, 3), p(1, 3, 4), p(2, 2, 5), p(3, 1, 6), p(1, 4, 7), \dots$ where the integer k denotes the term $s^k(0)$. However, we cannot find *directly* a relevant generalization, because the value of z depends on *two* integer variables corresponding to x and y respectively.

Let us fix (interactively) the value of x , for instance $x = 1$. When restricted to the corresponding subsequence, DS3 generates the following sequence:

```
Found 1 different form(s)
++p([s,1](0), [s,Plus[1,n]](0), [s,Plus[1,Times[Rational[1,
2],n],Times[Rational[1, 2],Power[n,2]]]](0))

Current Form has 14 different instances found

Printing Instances of the current Form
++p([s,1](0), [s,1](0), [s,1](0))
++p([\textbf{s},1](0), [s,2](0), [s,2](0))
++p([s,1](0), [s,3](0), [s,4](0))
++p([s,1](0), [s,4](0), [s,7](0))
++p([s,1](0), [s,5](0), [s,11](0))
% ... [skipped]
*****
```

Some explanations about the syntax are necessary here. DS3 displays the list of clause classes found by applying the algorithm in Section 3.1. For each class, the number of instances is printed and the whole list of instances (i.e. the clauses belonging to this clause class) is displayed.

A term class $f(t_1, \dots, t_n)^n.s$ is written $[f, n](s_1, \dots, s_n)$ where s_i is $\$$ if t_i is the hole and s_i is t_i otherwise. In particular, if f is monadic, then $[f, n](x)$ denotes the I -term $f(\diamond)^n.x$, i.e. $f^n(x)$. The arithmetic expressions $x + y, x \times$

$y, \frac{x}{y}, x^y$ are respectively denoted by `Plus[x,y]`, `Times[x,y]`, `Rational[x,y]` and `Power[x,y]`. A positive literal $p(x,y)$ is written `++p(x,y)` and a negative literal $\neg p(x,y)$ is written `--p(x,y)`. Some parts of the output files are skipped for the sake of readability.

Thus, the above sequence is: $p(s(0), s(\diamond)^{1+n}.0, s(\diamond)^{1+\frac{1}{2} \times n + \frac{1}{2} \times n^2}.0)$.

Note that the generalization algorithm of [1] would only compute the sequence $p(s(0), s(s^n(0)), s(s^m(0)))$, which is obviously more general (i.e. less precise) than the one obtained with DS3. For $x = 2$, we get:

```
Found 1 different form(s)
++p([s,2](0),[s,Plus[1,n]](0),[s,Plus[3,Times[Rational[3,
2],n],Times[Rational[1, 2],Power[n,2]]]](0))
```

By repeating this process, we obtain a *sequence of I-terms*. DS3 can be recursively applied on this sequence, yielding:

```
=====
Found 1 different form(s)
++p([s,Plus[1,x]](0),[s,Plus[1,n]](0),[s,Plus[1,Times[Rational[1,
2],n],Times[Rational[1, 2],Power[n,2]],Times[Rational[3,
2],x],Times[n,x],Times[Rational[1, 2],Power[x,2]]]](0))

Printing Instances of the current Form
++p([s,1](0),[s,Plus[1,n]](0),[s,Plus[1,Times[Rational[1,
2],n],Times[Rational[1, 2],Power[n,2]]]](0))
++p([s,2](0),[s,Plus[1,n]](0),[s,Plus[3,Times[Rational[3,
2],n],Times[Rational[1, 2],Power[n,2]]]](0))
++p([s,3](0),[s,Plus[1,n]](0),[s,Plus[6,Times[Rational[5,
2],n],Times[Rational[1, 2],Power[n,2]]]](0)) ... [skipped]
***** End processing
```

Thus we get: $p(s(\diamond)^{(1+x)}.0, s(\diamond)^{1+n}.0, s(\diamond)^{1+\frac{1}{2} \times n + \frac{1}{2} \times n^2 + \frac{3}{2} \times x + n \times x + \frac{1}{2} \times x^2}.0)$.

4.2 Detecting Satisfiability and Building Models

We look for a model of the following set of clauses

$$S = \{p(x, x, z), p(f(x, g(z), x), y, z) \vee \neg p(x, f(y, g(z), y), z), \neg p(x, f(x, a, x), z)\}.$$

Though apparently simple, this example is difficult because S has only models of infinite cardinality. Thus, finite model builders are useless. Moreover, the usual resolution strategies (as well as other proof procedures such as tableaux or connection methods) do not terminate on S .

The E -prover – using standard setting – would produce an infinite number of distinct clauses, thus is not able to detect that S is satisfiable. When applied on the obtained search space, our algorithm generates the following sequence of clauses:

```

Printing the simplified clauses
=====
Found 5 different form(s) ++p(X1,X1,X2)
Current Form has 1 different
instances found Printing Instances of the current Form ++p(X1,X1,X2)
*****
++p([f,1](X1,[g,1](X2),X1),X3,X2)  V --p(X1,[f,1](X3,[g,1](X2),X3),X2)
Current Form has 1 different
instances found Printing Instances of the current Form
++p([f,1](X1,[g,1](X2),X1),X3,X2)  V --p(X1,[f,1](X3,[g,1](X2),X3),X2)
*****
--p(X1,[f,1](X1,[g,1](X2),X1),X2)
Current Form has 1 different
instances found Printing Instances of the current Form
--p(X1,[f,1](X1,[g,1](X2),X1),X2)
*****
++p({{[f,Plus[2,Times[2,n]]}($,[g,1](X2),$)(X1)}} ,X1,X2)
Current
Form has 7 different instances found
Printing Instances of the current Form
++p({{[f,2]($,[g,1](X2),$)(X1)}} ,X1,X2)
++p({{[f,4]($,[g,1](X2),$)(X1)}} ,X1,X2)
++p({{[f,6]($,[g,1](X2),$)(X1)}} ,X1,X2)
++p({{[f,8]($,[g,1](X2),$)(X1)}} ,X1,X2)
++p({{[f,10]($,[g,1](X2),$)(X1)}} ,X1,X2)
++p({{[f,12]($,[g,1](X2),$)(X1)}} ,X1,X2)
++p({{[f,14]($,[g,1](X2),$)(X1)}} ,X1,X2)
*****
++p({{[f,Plus[2,n]]($,[g,1](X2),$)(X1)}} ,X3,X2)
V --p(X1,{{[f,Plus[2,n]]($,[g,1](X2),$)(X3)}} ,X2)
Current Form has 9 different instances found
Printing Instances of the current Form
++p({{[f,2]($,[g,1](X2),$)(X1)}} ,X3,X2)
V --p(X1,{{[f,2]($,[g,1](X2),$)(X3)}} ,X2)
++p({{[f,3]($,[g,1](X2),$)(X1)}} ,X3,X2)
V --p(X1,{{[f,3]($,[g,1](X2),$)(X3)}} ,X2)
++p({{[f,4]($,[g,1](X2),$)(X1)}} ,X3,X2)
V --p(X1,{{[f,4]($,[g,1](X2),$)(X3)}} ,X2)
... [skipped]
*****
End processing

```

Thus the following sequence is obtained: $p(f(\diamond, g(z), \diamond)^{2 \times n}.x, x, z)$.

Then, the unit clause $\{p(f(\diamond, g(z), \diamond)^{2 \times n}.x, x, z)\}$ defines an Herbrand model of the above clause set: $p(u, v, w)$ holds if u is of the form $f(\diamond, g(w), \diamond)^{2 \times n}.v$ for $n \in \mathbb{N}$ (see [12,4] for more details on the use of I -terms for representing Herbrand interpretations). This can be automatically checked (but this is **not** done currently by the DS3 tool) since the first-order equational theory of I -terms is decidable [12] (the interested reader should consult [4] for more details on the

relationship between the evaluation problem and the solving of purely equational formulae).

We give another, somewhat simpler, example, taken from [9].

$$S' = \{q(x, f(y)) \vee \neg p(x, y), q(x, f(y)) \vee \neg q(x, y), \\ p(x, f(x)) \vee \neg q(x, f(x)), p(x, f(x)), \neg q(x, f(x))\}.$$

S' is satisfiable but *has no finite model*. The E -prover does not terminate on S' . By analyzing the set of generated clauses, DS3 produces the following sequences:

```
Printing the simplified clauses
=====
Found 9 different form(s)
% ... [skipped]
*****
--q([f,Plus[1,n]](X1),X1)
Current Form has
329 different instances found
Printing Instances of the current Form
--q([f,1](X1),X1)
--q([f,2](X1),X1)
--q([f,3](X1),X1)
--q([f,4](X1),X1)
--q([f,5](X1),X1) --q([f,6](X1),X1)
% ... [skipped]
*****
--p([f,Plus[1,n]](X1),X1)
Current Form has 329 different instances
found
Printing Instances of the current Form
--p([f,1](X1),X1)
--p([f,2](X1),X1)
--p([f,3](X1),X1)
--p([f,4](X1),X1)
--p([f,5](X1),X1)
--p([f,6](X1),X1)
% ... [skipped]
*****
End processing
```

Again, these unit clauses immediately define a model of S' : $p(x, y)$ and $q(x, y)$ are *true* iff $x \neq f(\diamond)^n.y$.

4.3 Generating Inductive Lemmata

Inductionless-induction (II) (see e.g. [7]) illustrates particularly well the usefulness of the aforementioned ideas and techniques.

A first-order formula φ is an *inductive consequence* of a set of first-order sentences E iff for every Herbrand interpretation \mathcal{H} , $\mathcal{H} \models E$ implies $\mathcal{H} \models \varphi$.

The idea behind II is to axiomatize the minimal model of E with a finite set of axioms, say A , in such a way that φ is an inductive consequence of E iff $A \cup E \cup \varphi$ is consistent (satisfiable). II is therefore also named *proof by consistency*. A is called an *I-axiomatization*.

A very interesting feature of this approach is that a general-purpose theorem prover can be used for proving inductive theorems (instead of having to use specialized inductive provers such as SPIKE [3]). The only requirement is to design dedicated strategies [7]. Of course, in most cases, the theorem provers will not terminate on $A \cup E \cup \varphi$ thus we will not be able to detect satisfiability [17]. In order to obtain a saturated set of clauses, it is often needed to add additional inductive lemmata (stating properties that are not true in general, but that are true in Herbrand interpretations). These lemmata must be provided by the user. An interesting application of our technique is its ability to *automatically* generate such inductive lemmata.

We give a simple example. Assume that we define a function $l0$ mapping each integer n to a list of n elements “0”, and a function $count(l)$ counting the number of occurrences of 0 in l . $l0$ and $count$ can be defined by the following axiomatization A :

$$\begin{array}{ll} l0(0) = nil & l0(s(n)) = cons(0, l0(n)) \\ count(l) = count'(l, 0) & count'(nil, n) = n \\ count'(cons(0, l), n) = count'(l, s(n)) & count'(cons(s(n), l), m) = count'(l, m) \end{array}$$

Assume that we want to check that for all lists l , we have $count(l0(n)) = n$. This is an *inductive* property, because it is easy to check that $count(l0(n)) = n$ is *not* a logical consequence of A (but any ground instance of $count(l0(n)) = n$ is a consequence of A). We can use II to prove that this inductive property holds.

To this aim we need to add to A both the theorem to prove and an *I-axiomatisation* (see [7]). Then, we have to check that the obtained clause set is satisfiable. This ensures that there is at least one Herbrand model of A satisfying $count(l0(n)) = n$, and the *I-axiomatisation* guarantees that this model is minimal.

We use the usual *I-axiomatisation*:

$$\begin{array}{l} \{0 \neq s(x), cons(u, v) \neq nil, s(x) \neq s(y) \vee x = y, \\ cons(x, y) \neq cons(u, v) \vee x = u, cons(x, y) \neq cons(u, v) \vee y \neq v\}. \end{array}$$

The E -prover does not terminate on the obtained clause set, thus *fails to detect satisfiability*. The induction theorem prover SPIKE [3] also diverges on this example. But DS3 generates the following sequence: $s(\diamond)^n.x = count'(l0(x), n)$.

Printing the simplified clauses

=====

Found 20 different form(s)

++equal([l0,1](0),nil)

Current Form has 1 different instances found

Printing Instances of the current Form

++equal([l0,1](0),nil)

% ... [skipped]

```

++equal([count_aux,1]([10,1](X1),[s,Plus[1,n]](0)),[s,Plus[1,n]](X1))
Current Form has 202 different instances found
Printing Instances of the current Form
++equal([count_aux,1]([10,1](X1),[s,1](0)),[s,1](X1))
++equal([count_aux,1]([10,1](X1),[s,2](0)),[s,2](X1))
++equal([count_aux,1]([10,1](X1),[s,3](0)),[s,3](X1))
++equal([count_aux,1]([10,1](X1),[s,4](0)),[s,4](X1))
++equal([count_aux,1]([10,1](X1),[s,5](0)),[s,5](X1))
++equal([count_aux,1]([10,1](X1),[s,6](0)),[s,6](X1))
++equal([count_aux,1]([10,1](X1),[s,7](0)),[s,7](X1))
++equal([count_aux,1]([10,1](X1),[s,8](0)),[s,8](X1))
++equal([count_aux,1]([10,1](X1),[s,9](0)),[s,9](X1))
++equal([count_aux,1]([10,1](X1),[s,10](0)),[s,10](X1))
++equal([count_aux,1]([10,1](X1),[s,11](0)),[s,11](X1))
*****
% ... [skipped]
*****
++equal([count_aux,1](nil,[s,Plus[1,n]](0)),[s,Plus[1,n]](0))
Current Form has 202 different instances found
Printing Instances of the current Form
++equal([count_aux,1](nil,[s,1](0)),[s,1](0))
++equal([count_aux,1](nil,[s,2](0)),[s,2](0))
++equal([count_aux,1](nil,[s,3](0)),[s,3](0))
++equal([count_aux,1](nil,[s,4](0)),[s,4](0))
++equal([count_aux,1](nil,[s,5](0)),[s,5](0))
++equal([count_aux,1](nil,[s,6](0)),[s,6](0))
++equal([count_aux,1](nil,[s,7](0)),[s,7](0))
++equal([count_aux,1](nil,[s,8](0)),[s,8](0))
++equal([count_aux,1](nil,[s,9](0)),[s,9](0))
++equal([count_aux,1](nil,[s,10](0)),[s,10](0))
++equal([count_aux,1](nil,[s,11](0)),[s,11](0))
*****
++equal([s,Plus[2,n]](X1),[count_aux,1]([10,1](X1),[s,Plus[2,n]](0)))
Current Form has 201 different instances found
Printing Instances of the current Form
++equal([s,2](X1),[count_aux,1]([10,1](X1),[s,2](0)))
++equal([s,3](X1),[count_aux,1]([10,1](X1),[s,3](0)))
++equal([s,4](X1),[count_aux,1]([10,1](X1),[s,4](0)))
++equal([s,5](X1),[count_aux,1]([10,1](X1),[s,5](0)))
++equal([s,6](X1),[count_aux,1]([10,1](X1),[s,6](0)))
++equal([s,7](X1),[count_aux,1]([10,1](X1),[s,7](0)))
++equal([s,8](X1),[count_aux,1]([10,1](X1),[s,8](0)))
++equal([s,9](X1),[count_aux,1]([10,1](X1),[s,9](0)))
++equal([s,10](X1),[count_aux,1]([10,1](X1),[s,10](0)))
++equal([s,11](X1),[count_aux,1]([10,1](X1),[s,11](0)))
++equal([s,12](X1),[count_aux,1]([10,1](X1),[s,12](0)))
*****
++equal([s,Plus[1,n]](0),[s,Plus[1,n]](0))
Current Form has 201 different instances found
Printing Instances of the current Form

```

```

++equal([s,1](0),[s,1](0))
++equal([s,2](0),[s,2](0))
++equal([s,3](0),[s,3](0))
++equal([s,4](0),[s,4](0))
++equal([s,5](0),[s,5](0))
++equal([s,6](0),[s,6](0))
++equal([s,7](0),[s,7](0))
++equal([s,8](0),[s,8](0))
++equal([s,9](0),[s,9](0))
++equal([s,10](0),[s,10](0))
++equal([s,11](0),[s,11](0))
*****
End processing

```

This obviously suggests to add the above property as an inductive lemma. However, since the E -prover does not handle I -terms, we need to express this property in first-order logic. This is done by introducing an additional symbol f . $f(x, n)$ encodes $s(\diamond)^n.x$. f is defined by the set of axioms: $\{f(x, 0) = x, f(x, s(n)) = s(f(x, n))\}$. The above property becomes: $f(x, n) = \text{count}'(l(x), n)$. By adding this property and the above axioms in the clause set, we obtain a new set of clauses on which the E -prover terminates. Thus, satisfiability is detected and the **above induction theorem is proven**.

Obviously, the whole process can be automatized. Of course for more complicated examples the help of the user may be needed to select the appropriate lemma in a list of candidates.

Remark 1. The above process introduces additional function symbols in the clause set at hand. But this is not a problem. Indeed, it is obvious that if the clause set obtained after transformation is satisfiable, then a model of the original clause set exists. Thus, proving that the obtained clause set is satisfiable (by saturation) suffices to prove that the initial clause set is satisfiable hence that the inductive theorem holds.

5 Conclusion

We have introduced a technique to analyze search spaces of ATP and shown its usefulness by presenting three kinds of applications. The most promising one is – to our opinion – the possibility of generating (suggesting) inductive lemmata. Obviously our approach can be useful beyond AD, in all domains in which it is relevant characterizing structurally (possibly infinite) sets of particular items, as for example discovering infinite models, analogy, learning or analysis of programs traces, ... Hopefully this work will help to develop research on search spaces of automated theorem provers by characterizing sets of particular consequences and it will open new fruitful possibilities of *cooperation* between theorem provers and symbolic computation tools. It could be worth comparing our approach with existing techniques used by model checkers in order to detect loops in system executions.

References

1. Amaniss, A., Hermann, M., Lugiez, D.: Set operations for recurrent term schematizations. In: Bidoit, M., Dauchet, M. (eds.) CAAP 1997, FASE 1997, and TAPSOFT 1997. LNCS, vol. 1214, pp. 333–344. Springer, Heidelberg (1997)
2. Bernstein, M., Sloane, N.J.A.: Some Canonical Sequences of Integers. *Linear Algebra and its Applications* 228(1–3), 57–72 (1995)
3. Bouhoula, A., Kounalis, E., Rusinowitch, M.: SPIKE, an automatic theorem prover. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624, pp. 460–462. Springer, London, UK (1992)
4. Caferra, R., Leitsch, A., Peltier, N.: Automated Model Building. *Applied Logic Series*, vol. 31. Kluwer Academic Publishers, Boston (2004)
5. Chen, H., Hsiang, J.: Logic programming with recurrence domains. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) Automata, Languages and Programming. LNCS, vol. 510, pp. 20–34. Springer, Heidelberg (1991)
6. Comon, H.: Disunification: A survey. In: Lassez, J.-L., Plotkin, G. (eds.) Computational Logic: Essays in Honor of Alan Robinson, pp. 322–359. MIT Press, Cambridge (1991)
7. Comon, H.: Inductionless induction. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, ch. 14, pp. 913–962. North-Holland (2001)
8. Comon, H.: On unification of terms with integer exponents. *Mathematical System Theory* 28, 67–88 (1995)
9. Dreben, B., Goldfarb, W.D.: The Decision Problem, Solvable Classes of Quantificational Formulas. Addison-Wesley, Reading (1979)
10. Hermann, M., Galbavý, R.: Unification of Infinite Sets of Terms schematized by Primal Grammars. *Theoretical Computer Science* 176(1–2), 111–158 (1997)
11. Kleene, S.C.: Introduction to Metamathematics. North-Holland, Amsterdam (1952)
12. Peltier, N.: Increasing the capabilities of model building by constraint solving with terms with integer exponents. *Journal of Symbolic Computation* 24, 59–101 (1997)
13. Peltier, N.: A General Method for Using Terms Schematizations in Automated Deduction. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 578–593. Springer, Heidelberg (2001)
14. Plotkin, G.D.: A note on inductive generalization. *Machine Intelligence* 5, 153–163 (1970)
15. Plotkin, G.D.: A Further Note on Inductive Generalization. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence 6, chapter 8, pp. 101–124. Edinburgh University Press, Edinburgh (1971)
16. Schulz, S.: The E Equational Theorem Prover,
<http://www4.informatik.tu-muenchen.de/~schulz/WORK/eprover.html>
17. Walsh, T.: A divergence critic for inductive proof. *Journal of Artificial Intelligence Research* 4, 209–235 (1996)
18. Wos, L.: Automated Reasoning, 33 Basic Research Problems. Prentice-Hall, Englewood Cliffs (1988)

Continuation Semantics for Symmetric Categorial Grammar

Raffaella Bernardi¹ and Michael Moortgat^{2,*}

¹ Free University of Bozen-Bolzano, Italy
bernardi@inf.unibz.it

² Utrecht Institute of Linguistics OTS, The Netherlands
moortgat@let.uu.nl

Abstract. Categorial grammars in the tradition of Lambek [1,2] are asymmetric: sequent statements are of the form $\Gamma \Rightarrow A$, where the succedent is a single formula A , the antecedent a structured configuration of formulas A_1, \dots, A_n . The absence of structural context in the succedent makes the analysis of a number of phenomena in natural language semantics problematic. A case in point is scope construal: the different possibilities to build an interpretation for sentences containing generalized quantifiers and related expressions. In this paper, we explore a symmetric version of categorial grammar based on work by Grishin [3]. In addition to the Lambek product, left and right division, we consider a dual family of type-forming operations: coproduct, left and right difference. Communication between the two families is established by means of structure-preserving distributivity principles. We call the resulting system **LG**. We present a Curry-Howard interpretation for **LG**(/, \, \odot , \oslash) derivations. Our starting point is Curien and Herbelin's sequent system for $\lambda\mu$ calculus [4] which capitalizes on the duality between logical implication (i.e. the Lambek divisions under the formulas-as-types perspective) and the difference operation. Importing this system into categorial grammar requires two adaptations: we restrict to the subsystem where linearity conditions are in effect, and we refine the interpretation to take the left-right symmetry and absence of associativity/commutativity into account. We discuss the continuation-passing-style (CPS) translation, comparing the call-by-value and call-by-name evaluation regimes. We show that in the latter (but not in the former) the types of **LG** are associated with appropriate denotational domains to enable a proper treatment of scope construal.

1 Background

Lambek-style categorial grammars offer an attractive computational perspective on the principle of compositionality: under the Curry-Howard interpretation,

* We thank Chris Barker and Ken Shan for sharing their view on continuation semantics with us at an earlier presentation of Lambek-Grishin calculus at the workshop Proof Theory at the Syntax-Semantics Interface (LSA Institute, Harvard/MIT, July 2005). Special thanks to Peter Selinger, for helpful discussion on the duality between call-by-value and call-by-name during GEOCAL'06 (Marseille-Luminy, February 2006) and to Philippe de Groote for bringing Curien and Herbelin's work to our attention. All errors remain our own.

derivations are associated with instructions for meaning assembly. In natural language semantics, scope construal of generalized quantifier expressions presents an ideal testing ground to bring out the merits of this approach. Scope construal exemplifies a class of phenomena known as *in situ* binding. An *in situ* binder syntactically occupies the position of a phrase of type A ; semantically, it binds an A -type variable in that position within a context of type B , producing a value of type C as a result. The inference pattern of (1) (from [5]) schematically captures this behaviour in the format of a sequent rule. The challenge is to solve the equation for the type alias $q(A, B, C)$ in terms of the primitive type-forming operations.

$$\frac{\Delta[x : A] \Rightarrow N : B \quad \Gamma[y : C] \Rightarrow M : D}{\Gamma[\Delta[z : q(A, B, C)]] \Rightarrow M[y := (z \lambda x.N)] : D} . \quad (1)$$

It is a poignant irony that precisely in the area of scope construal, the performance of the original Lambek calculus (whether in its associative or non-associative incarnation) is disappointing. For a sentence-level generalized quantifier (GQ) phrase, we have $A = np$, $B = C = s$ in (1). The type-forming operations available to define $q(np, s, s)$ are the left and right slashes. A first problem is the lack of *type uniformity*. Given standard modeltheoretic assumptions about the interpretation of the type language, an assignment $s/(np \backslash s)$ to a GQ phrase is associated with an appropriate domain of interpretation (a set of sets of individuals), but with such a type a GQ is syntactically restricted to subject positions: for phrase-internal GQ occurrences, context-dependent extra lexical type assignments have to be postulated. Second, this lexical ambiguity strategy breaks down as soon as one considers *non-local* scope construal, where the distance between the GQ occurrence and the sentential domain where it establishes its scope can be unbounded.

The solutions that have been proposed in the type-logical literature we consider suboptimal. The type-shifting approach of Hendriks [6] and the multimodal accounts based on wrapping operations of Morrill and co-workers [7,8] each break the isomorphic relation between derivations and terms that is at the heart of the Curry-Howard interpretation. Hendriks introduces a one-to-many dichotomy between syntactic and semantic derivations. Morrill makes the opposite choice: a multiplicity of syntactically distinct implicational operations which collapse at the semantic level.

The approach we develop in the sections below sticks to the *minimal* categorial logic: the pure logic of residuation. We overcome the expressive limitations of the Lambek calculi by lifting the single succedent formula restriction and move to a symmetric system where the Lambek connectives (product, left and right division) coexist with a dual family (coproduct, right and left difference). The communication between these two families is expressed in terms of Grishin's [3] distributivity principles. Figure 1 schematically presents the outline of the paper. In §2 we present **LG** in algebraic format and discuss the symmetries that govern the vocabulary of type-forming operations. In §3 we present a 'classical' term language for the **LG** type system, and we discuss how a term τ of type A is obtained as the Curry-Howard image of an **LG** sequent derivation π . In §4 we then study the

CPS interpretation of types and terms, comparing the dual call-by-value $[\cdot]$ and call-by-name $\llbracket \cdot \rrbracket$ regimes. Under the CPS interpretation, the classical terms for **LG** derivations are transformed into terms of the simply typed lambda calculus — the terms that code proofs in positive intuitionistic logic. The λ_{\rightarrow} terms thus obtained adequately reflect NL meaning composition, and (unlike the terms for Multiplicative Linear Logic or its categorical equivalent **LP**) they are obtained in a structure-preserving way. In §5 we illustrate the approach with a discussion of scope construal. We investigate under what conditions the lexical constants of the original Lambek semantics can be lifted to the call-by-value $[\cdot]$ and/or call-by-name $\llbracket \cdot \rrbracket$ level, and study how the λ_{\rightarrow} terms one obtains after this transformation and β normalisation encode the different possibilities for scope construal. In the concluding section §6, we point to some directions for future work.

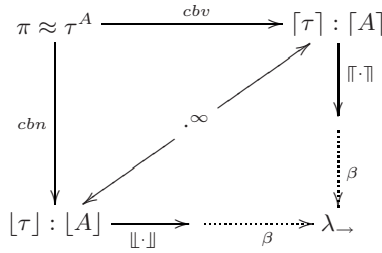


Fig. 1. Outline of the paper

Relation to previous work. Lambek [9] was the first paper to bring Grishin’s work under the attention of a wider public. Lambek’s bilinear systems are both stronger and weaker than what we propose here: they have hard-wired associativity for \otimes, \oplus , which means that control over constituent structure is lost; in addition, only half of the Grishin laws are taken into account ($G1, G3$ in Figure 2), an omission that precludes the account of non-peripheral scope construal presented here. De Groote [10] introduced $\lambda\mu$ calculus and continuations into the linguistic discussion of scope construal; Barker and Shan, in a series of papers ([11,12,13] among others), have been advocating this approach for a variety of semantic phenomena. We discuss the relation of our proposal to theirs in §6. Duality between the call-by-value and call-by-name evaluation strategies has been obtained in [14,4,15], among others. Our starting point is the Curien/Herbelin system because, in contrast to the other cited works, it has implication and difference as primitive operations.

2 The Lambek-Grishin Calculus

The calculus that we will use in this paper is presented in Figure 2. We refer to this system as **LG**. In (2), one finds the extended vocabulary of **LG**: the familiar Lambek operators $\otimes, \backslash, /$ are complemented with a family \oplus, \odot, \ominus . In verbal communication, we pronounce $B \backslash A$ as ‘ B under A ’, A/B as ‘ A over B ’, $B \odot A$

as ‘ B from A ’ and $A \otimes B$ as ‘ A less B ’. As usual under the formulas-as-types perspective, we can view the expressions of (2) as *types* with $\otimes, \backslash, /, \oplus, \odot, \oslash$ as type-forming operations, or as logical *formulas*, with $\otimes, \backslash, /, \oplus, \odot, \oslash$ as connectives.

$A, B ::= p \mid$	atoms: s sentence, np noun phrases, ...
$A \otimes B \mid B \backslash A \mid A/B \mid$	product, left vs right division (TYPES) tensor, left vs right implication (FORMULAS)
$A \oplus B \mid A \odot B \mid B \oslash A$	coproduct, right vs left difference (TYPES) cotensor, left vs right coimplication (FORMULAS)

The **LG** type system exhibits rich symmetries, discussed in full in [16]. In the present paper, two kinds of mirror symmetry will play an important role. The first \bowtie is internal to the \otimes and \oplus families and is order-preserving; the second \cdot^∞ relates the \otimes and \oplus families and is order-reversing. We have $p^{\bowtie} = p = p^\infty$ and the (bidirectional) translation tables of (3).

$$\bowtie \frac{C/D \quad A \otimes B \quad B \oplus A \quad D \odot C}{D \backslash C \quad B \otimes A \quad A \oplus B \quad C \odot D} \quad \infty \frac{C/B \quad A \otimes B \quad A \backslash C}{B \odot C \quad B \oplus A \quad C \otimes A} \quad (3)$$

(PRE-ORDER)	$A \leq A$	$\frac{A \leq B \quad B \leq C}{A \leq C}$
(RESIDUATION)	$A \leq C/B$ iff	$A \otimes B \leq C$ iff $B \leq A \backslash C$
(DUAL RESIDUATION)	$C \odot B \leq A$ iff	$C \leq A \oplus B$ iff $A \odot C \leq B$
(GRISHIN INTERACTION)	$(G1) \quad (A \odot B) \otimes C \leq A \odot (B \otimes C) \quad C \otimes (B \odot A) \leq (C \otimes B) \odot A \quad (G3)$ $(G2) \quad C \otimes (A \odot B) \leq A \odot (C \otimes B) \quad (B \odot A) \otimes C \leq (B \otimes C) \odot A \quad (G4)$	

Fig. 2. **LG**: symmetric Lambek calculus with Grishin interaction principles

Algebraically, the Lambek operators $/, \otimes, \backslash$ form a residuated triple; likewise, the \odot, \oplus, \oslash family forms a dual residuated triple. The *minimal* symmetric categorical grammar consists of just the preorder axioms (reflexivity and transitivity of \leq) together with these (dual) residuation principles.¹

Grishin’s interaction principles. The minimal symmetric system is of limited use if one wants to address the linguistic problems discussed in the introduction. In a system with just the (dual) residuation principles, for every theorem of the (non-associative) Lambek calculus, one also has its image under \cdot^∞ : $A \leq B$ iff

¹ For a comprehensive overview (from a Display Logic perspective) of the substructural space of which **LG** is an inhabitant, the reader can consult [17]. De Groote and Lamarche [18] present sequent calculus and proof nets for a negation-tensor-par formulation of Classical Non-associative Lambek Calculus, of which **LG** is the subsystem given by the polarities of the operators in (2).

$B^\infty \leq A^\infty$. Interaction between the \otimes and the \oplus family, however, is limited to gluing together theorems of the two families via cut. This limited interaction means that a formula from the \oplus family which is trapped in a \otimes context (or vice versa) will be inaccessible for logical manipulation.

The interaction principles proposed in [3] address this situation. Consider first $G1$ and $G2$ in Fig 2. On the lefthand side of the inequality, a coimplication $A \oslash B$ is hidden as the first or second coordinate of a product. The postulates invert the dominance relation between \otimes and \oslash , raising the subformula A to a position where it can be shifted to the righthand side by means of the dual residuation principle. $G3$ and $G4$ are the images of $G1$ and $G2$ under \cdot^\boxtimes . Similarly, a left or right implication trapped within a \oplus context can be liberated by means of the \cdot^∞ images in (4). Combined with transitivity, the Grishin postulates take the form of inference rules ($G1$: from $A \oslash (B \otimes C) \leq D$ conclude $(A \oslash B) \otimes C \leq D$, etc.)

$$\begin{array}{ll} (G3)^\infty & A \setminus (B \oplus C) \leq (A \setminus B) \oplus C \quad (C \oplus B) / A \leq C \oplus (B / A) \quad (G1)^\infty; \\ (G4)^\infty & A \setminus (C \oplus B) \leq C \oplus (A \setminus B) \quad (B \oplus C) / A \leq (B / A) \oplus C \quad (G2)^\infty. \end{array} \quad (4)$$

The Grishin laws manifest themselves in many forms. The key observation for their usefulness in the analysis of scope construal lies in the fact that $(B \oslash C) \oslash A \leq C / (A \setminus B)$ is a theorem of **LG**. This means that a Lambek type $s / (np \setminus s)$ is derivable from a $(s \oslash s) \oslash np$ type; what can be done with the former can also be done with the latter, but the coimplication type also has non-local capabilities thanks to the Grishin interactions.

Apart from the interaction principles $G1$ – $G4$ (and their duals) which will be at the heart of our analysis of scope construal, Grishin considers other options for generalizing Lambek calculus. The reader is referred to [16] for a discussion of these options and their potential linguistic uses. Also in [16] one finds a decision procedure for **LG** based on the monotonicity laws for the type-forming operations, together with the residuation principles and the Grishin principles in rule form.

3 Proofs and Terms

The term language we will use for **LG** derivations is a directional version of Curien/Herbelin's classical $\bar{\lambda}\mu\tilde{\mu}$ terms which takes the \cdot^\boxtimes symmetry into account. The term language distinguishes terms, coterms (contexts) and commands. We give the syntax of the term language in (5). For terms, we use x, M, N ; for coterms (contexts) α, K, L ; commands are cuts $M * L$ between a term M and a coterm L . We overload the notation, writing $x \setminus M$ versus M / x for the left and right abstraction constructs; similarly for coabstraction. As discussed in §1, the familiar lambda abstraction of λ_\rightarrow will be reinstalled as a result of the CPS transformation on the terms of (5).

	$x \in \text{Term}^A$	if $x \in \text{Var}^A$
(L ABSTRACTION)	$x \setminus M \in \text{Term}^{B \setminus A}$	if $x \in \text{Var}^B, M \in \text{Term}^A$
(R ABSTRACTION)	$M/x \in \text{Term}^{A/B}$	if $x \in \text{Var}^B, M \in \text{Term}^A$
(L COAPPLICATION)	$K \prec M \in \text{Term}^{B \odot A}$	if $K \in \text{CoTerm}^B, M \in \text{Term}^A$
(R COAPPLICATION)	$M \succ K \in \text{Term}^{A \odot B}$	if $K \in \text{CoTerm}^B, M \in \text{Term}^A$
(R SHIFT)	$\mu\alpha.(x * K) \in \text{Term}^B$	if $\alpha \in \text{CoVar}^B, x \in \text{Var}^A, K \in \text{CoTerm}^A$
	$\alpha \in \text{CoTerm}^A$	if $\alpha \in \text{CoVar}^A$
(L APPLICATION)	$M \times K \in \text{CoTerm}^{B \setminus A}$	if $K \in \text{CoTerm}^A, M \in \text{Term}^B$
(R APPLICATION)	$K \rtimes M \in \text{CoTerm}^{A/B}$	if $K \in \text{CoTerm}^A, M \in \text{Term}^B$
(L COABSTR)	$\alpha \prec K \in \text{CoTerm}^{B \odot A}$	if $\alpha \in \text{CoVar}^B, K \in \text{CoTerm}^A$
(R COABSTR)	$K \odot \alpha \in \text{CoTerm}^{A \odot B}$	if $\alpha \in \text{CoVar}^B, K \in \text{CoTerm}^A$
(L SHIFT)	$\tilde{\mu}x.(M * \alpha) \in \text{CoTerm}^A$	if $x \in \text{Var}^A, M \in \text{Term}^B, \alpha \in \text{CoVar}^B$

(5)

As in the case of the Lambek calculus, for **LG** we are interested in the resource-sensitive sublanguage. This means that the (co)abstraction and (co)application cases are subject to a linearity condition: the (co)variable bound in a (co)abstraction occurs free exactly once in the body; in the (co)application case the sets of free (co)variables of the term and coterm involved are disjoint. Our use of cut is restricted to patterns $x * K$ ($M * \alpha$) in the shift right (left) construct, where μ ($\tilde{\mu}$) obeys the single-bind restriction.

The dualities we discussed for the type system extend to the term language: (6) and (7). The latter acts on the directional constructs; identity otherwise.

$$\begin{array}{ll}
x^\infty & = \alpha \\
(x \setminus M)^\infty & = M^\infty \odot \alpha \\
(M/x)^\infty & = \alpha \odot M^\infty \\
(M \succ K)^\infty & = K^\infty \times M^\infty \\
(K \prec M)^\infty & = M^\infty \rtimes K^\infty \\
(\mu\beta.(x * K))^\infty & = \tilde{\mu}y.(K^\infty * \alpha)
\end{array}
\qquad
\begin{array}{ll}
\alpha^\infty & = x; \\
(K \odot \alpha)^\infty & = x \setminus K^\infty; \\
(\alpha \odot K)^\infty & = K^\infty / x; \\
(M \times K)^\infty & = K^\infty \succ M^\infty; \\
(K \rtimes M)^\infty & = M^\infty \prec K^\infty; \\
(\tilde{\mu}y.(M * \alpha))^\infty & = \mu\beta.(x * M^\infty).
\end{array}
\tag{6}$$

$$\begin{array}{ll}
(M \succ K)^\boxtimes = K^\boxtimes \prec M^\boxtimes & (M \times K)^\boxtimes = K^\boxtimes \rtimes M^\boxtimes; \\
(x \setminus M)^\boxtimes = M^\boxtimes / x & (K \odot \alpha)^\boxtimes = \alpha \odot K^\boxtimes.
\end{array}
\tag{7}$$

3.1 LG Sequent Calculus

In Lambek calculus, sequents are statements $\Gamma \Rightarrow B$, where Γ is a binary tree with formulas A_1, \dots, A_n at the yield, and B is a single formula. The structure-building operation which puts together the antecedent tree is the counterpart of the \otimes logical operation. In **LG**, sequents $\Gamma \Rightarrow \Delta$ can have structures both in the antecedent and in the succedent. The sequent interpunction (which we write $\cdot \circ \cdot$) is the structural counterpart of \otimes in the antecedent, and of \oplus in the succedent. Notice that in the absence of associativity, \circ is a binary operation.

In the rules below, we decorate **LG** derivations with the terms of (5). We distinguish sequents $\Gamma \xrightarrow{M} \Delta[B]$ and cosequents $\Gamma[A] \xrightarrow{K} \Delta$ with proof term M and

coterm K respectively. A sequent (cosequent) has precisely one active succedent (antecedent) formula. The active formula is unlabeled. The passive antecedent (succedent) formulas are labeled with distinct variables x_i (covariables α_i).

For the axiomatic case, we distinguish two versions, depending on whether the succedent or the antecedent is the active formula. The rules (\Leftarrow) and (\Rightarrow) make it possible to shift the focus from antecedent to succedent or vice versa. These rules are in fact restricted cuts, where one of the premises is axiomatic (Axiom or Co-Axiom).

$$\frac{}{x : A \xrightarrow{x} A} \text{Ax} \quad \frac{}{A \xrightarrow{\alpha} \alpha : A} \text{Co-Ax} . \quad (8)$$

$$\frac{\Gamma[A] \xrightarrow{K} \Delta[\alpha : B]}{\Gamma[x : A] \xrightarrow{\mu\alpha.(x * K)} \Delta[B]} (\Leftarrow) \quad \frac{\Gamma[x : A] \xrightarrow{M} \Delta[B]}{\Gamma[A] \xrightarrow{\tilde{\mu}x.(M * \alpha)} \Delta[\alpha : B]} (\Rightarrow) \quad (9)$$

Let us now consider the sequent left and right rules for the connectives. We restrict attention to the (co)implication fragment, i.e. we only cater for \otimes and \oplus in their ‘structural’ form \circ as antecedent resp. succedent punctuation. The rules of use for the (co)implications are given in (10): these are two-premise rules, introducing an implication (coimplication) in the antecedent (succedent). Notice that we find the \cdot^\bowtie and \cdot^∞ symmetries here at the level of the inference rules, with \cdot^∞ (\cdot^\bowtie) relating pairs of rules in the horizontal (vertical) dimension.

$$\begin{array}{ccc} \frac{B \xrightarrow{K} \Delta \quad \Delta' \xrightarrow{M} \Gamma[A]}{\Delta' \xrightarrow{M \succ K} \Gamma[(A \otimes B) \circ \Delta]} (\otimes R) & & \frac{\Delta \xrightarrow{M} B \quad \Gamma[A] \xrightarrow{K} \Delta'}{\Gamma[\Delta \circ (B \setminus A)] \xrightarrow{M \bowtie K} \Delta'} (\setminus L) \\ & & \\ & & (\otimes R) \leftarrow \cdot^\infty \rightarrow (\setminus L) \\ & & \uparrow \qquad \qquad \uparrow \\ & & \cdot^\bowtie \qquad \qquad \cdot^\bowtie \\ & & \downarrow \qquad \qquad \downarrow \\ & & (\otimes R) \leftarrow \cdot^\infty \rightarrow (/L) \\ & & \\ \frac{B \xrightarrow{K} \Delta \quad \Delta' \xrightarrow{M} \Gamma[A]}{\Delta' \xrightarrow{K \prec M} \Gamma[\Delta \circ (B \otimes A)]} (\otimes R) & & \frac{\Delta \xrightarrow{M} B \quad \Gamma[A] \xrightarrow{K} \Delta'}{\Gamma[(A/B) \circ \Delta] \xrightarrow{K \bowtie M} \Delta'} (/L) \end{array} \quad (10)$$

The rules of proof for the (co)implications are given in (10): these are one-premise rules, introducing an implication (coimplication) in the succedent (antecedent).

$$\begin{array}{ccc}
\frac{x : B \circ \Gamma \xrightarrow{M} \Delta[A]}{\Gamma \xrightarrow{x \setminus M} \Delta[B \setminus A]} (\setminus R) & & \frac{\Gamma[A] \xrightarrow{K} \Delta \circ \alpha : B}{\Gamma[A \odot B] \xrightarrow{K \odot \alpha} \Delta} (\odot L) \\
\\
(\setminus R) \leftarrow \cdot^\infty \rightarrow (\odot L) & & \\
\uparrow \cdot^\infty & & \uparrow \cdot^\infty \\
\downarrow & & \downarrow \\
(/R) \leftarrow \cdot^\infty \rightarrow (\otimes L) & & \\
\\
\frac{\Gamma \circ x : B \xrightarrow{M} \Delta[A]}{\Gamma \xrightarrow{M/x} \Delta[A/B]} (/R) & & \frac{\Gamma[A] \xrightarrow{K} \alpha : B \circ \Delta}{\Gamma[B \odot A] \xrightarrow{\alpha \odot K} \Delta} (\otimes L)
\end{array} \tag{11}$$

Observe that to prove the soundness of the coimplication (implication) rules of proof from the algebraic presentation, one uses the Grishin interaction principles to move the B subformula upwards through the \otimes context (\oplus context), and then shifts it to the succedent (antecedent) part via the residuation principles. The Grishin interaction principles, in other words, are *absorbed* in these rules of proof. We illustrate this in (12) for $(\setminus R)$, writing Γ^\bullet (Δ°) for the formula equivalent of an antecedent (succedent) structure. The vertical dots abbreviate a succession of Grishin interaction steps.

$$\begin{array}{c}
B \otimes \Gamma^\bullet \leq \Delta^\circ[A] \\
\hline
\Gamma^\bullet \leq B \setminus \Delta^\circ[A] \\
\vdots \\
\Gamma^\bullet \leq \Delta^\circ[B \setminus A]
\end{array} \tag{12}$$

As indicated in §2, we will use the formula $(B \odot C) \odot A$ to do the work of the *in situ* binder schema $q(A, B, C)$. (Alternatively, we could have used its \cdot^∞ dual $A \odot (C \odot B)$.) The (qL) and (qR) rules of Fig 3 have the status of derived inference rules. We will use them in §5 to present proofs and terms in a more compact format. In §A we give a worked-out derivation of $(B \odot C) \odot A \Rightarrow C / (A \setminus B)$, together with further abbreviatory conventions. The reader may want to check that a cut of (qR) against (qL) can be rewritten with cuts on the subformulae A, B, C , as required: $\text{cobind}(M^A, K^B, \beta^C) * \text{bind}(x^A, N^B, L^C) \longrightarrow_\beta M * \tilde{\mu}x.(\mu\beta.(N * K) * L)$. One should keep in mind that (qL) and (qR) are short-cuts, i.e. ways of abbreviating a sequence of n inference steps as a one-step inference. For some theorems of **LG**, one cannot take a short-cut: their derivation requires the individual inference rules for the connectives involved. The type transition $(B \odot C) \odot A \Rightarrow ((D \setminus B) \odot (D \setminus C)) \odot A$ is an example. Its derivation is given in [16].

4 Interpretation: Continuation Semantics

We turn now to an interpretation for **LG** derivations in the continuation-passing-style (CPS). In the semantics of programming languages, CPS interpretation has

$$\begin{array}{c}
\frac{C \xrightarrow{L} \Delta \quad \Gamma[x : A] \xrightarrow{N} B}{\Gamma[(B \otimes C) \otimes A] \xrightarrow{\text{bind}(x, N, L)} \Delta} \text{ (qL)} \quad \triangleq \quad \frac{\frac{C \xrightarrow{L} \Delta \quad \Gamma[x : A] \xrightarrow{N} B}{\Gamma[x : A] \xrightarrow{N \succ L} (B \otimes C) \circ \Delta} (\otimes R)}{\Gamma[A] \xrightarrow{\tilde{\mu}x.((N \succ L) * \gamma)} \gamma : (B \otimes C), \Delta} \triangleq \\
\frac{\Gamma[A] \xrightarrow{\tilde{\mu}x.((N \succ L) * \gamma)} \gamma : (B \otimes C), \Delta}{\Gamma[(B \otimes C) \otimes A] \xrightarrow{\gamma \otimes (\tilde{\mu}x.((N \succ L) * \gamma))} \Delta} (\otimes L) \\
\\
\frac{B \xrightarrow{K} \Delta' \circ \beta : C \quad \Gamma \xrightarrow{M} \Delta[A]}{\Gamma \xrightarrow{\text{cobind}(M, K, \beta)} \Delta[\Delta' \circ ((B \otimes C) \otimes A)]} \text{ (qR)} \quad \triangleq \quad \frac{\frac{B \xrightarrow{K} \Delta' \circ \beta : C}{B \otimes C \xrightarrow{K \otimes \beta} \Delta'} (\otimes L) \quad \Gamma \xrightarrow{M} \Delta[A]}{\Gamma \xrightarrow{(K \otimes \beta) \prec M} \Delta[\Delta' \circ ((B \otimes C) \otimes A)]} (\otimes R)
\end{array}$$

Fig. 3. Derived inference rules for $(B \otimes C) \otimes A \approx q(A, B, C)$

been a fruitful strategy to make explicit (and open to manipulation) aspects of computation that remain implicit in a direct interpretation. In the direct interpretation, a function simply returns a value. Under the CPS interpretation, functions are provided with an extra argument for the continuation of the computation. This explicit continuation argument is then passed on when functions combine with each other. Key concepts, then, are “computation”, “continuation” and “value” and they way they relate to each other for different evaluation strategies.

Curien and Herbelin [4] develop the CPS interpretation for a classical system with an implication and a difference operation; call-by-value (cbv) $[\cdot]$ and call-by-name (cbn) $[\cdot]$ regimes are related by the duality between the implication and difference operations. For **LG** we refine the Curien/Herbelin continuation semantics to accommodate the left/right symmetry. We first consider the effect of the CPS interpretation on the level of *types*, comparing a call-by-value (cbv) and a call-by-name (cbn) regime; then we define the CPS interpretation on the level of the *terms* of (5).

Types: call-by-value. The target type language has a distinguished type R of responses, products and functions; all functions have range R . For each type A of the source language, the target language has values $V_A = [A]$, continuations $K_A = R^{V_A}$ (functions from V_A to R) and computations $C_A = R^{K_A}$ (functions from K_A to R).² Notice that given the canonical isomorphism $A \times B \rightarrow C \cong A \rightarrow (B \rightarrow C)$, one can also think of a $V_{A \setminus B}$ as a function from A values to B computations. For p atomic, $[p] = p$. In (13), the $[\cdot]$ translations for the (co)implications are related in the vertical dimension by left/right symmetry \cdot^∞ and in the horizontal dimension by arrow reversal \cdot^∞ : right difference is dual to left division, left difference dual to right division.

² In the schemas (13) and (14) we use exponent notation for function spaces, for comparison with [4]. In the text, we usually shift to the typographically more convenient arrow notation, compare A^B versus $B \rightarrow A$.

$$\begin{aligned}
[A \setminus B] &= R^{[A] \times R^{[B]}} & [B \odot A] &= [B] \times R^{[A]}; \\
[B/A] &= R^{R^{[B]} \times [A]} & [A \odot B] &= R^{[A]} \times [B].
\end{aligned} \tag{13}$$

Types: call-by-name. Under the call-by-name regime, for each type A of the source language, the target language has continuations $K_A = [A]$ and computations $C_A = R^{K_A}$. The call-by-name interpretation $[\cdot]$ is obtained as the composition of the \cdot^∞ duality map and the $[\cdot]$ interpretation: $[A] \triangleq [A^\infty]$. For atoms, $[p] = [p^\infty] = p$. For the (co)implications, compare the cbv interpretation (left) with the cbn interpretation (right) in (14).

$$\begin{aligned}
[A \setminus B] &= R^{[A] \times R^{[B]}} & R^{[A] \times R^{[B]}} &= [B \odot A]; \\
[B \odot A] &= [B] \times R^{[A]} & [B] \times R^{[A]} &= [A \setminus B]; \\
[B/A] &= R^{R^{[B]} \times [A]} & R^{R^{[B]} \times [A]} &= [A \odot B]; \\
[A \odot B] &= R^{[A]} \times [B] & R^{[A]} \times [B] &= [B/A].
\end{aligned} \tag{14}$$

Notice that for the call-by-name regime, the starting point is the level of continuations, not values as under call-by-value. Let's take the definition of $B \odot A$ by means of example. For call-by-value, one starts from $[B \odot A]$ (i.e., $V_{B \odot A}$) that is a pair $V_B \times K_A$; hence its continuation is $K_{B \odot A} = (V_B \times K_A) \rightarrow R$ and its computation is $C_{B \odot A} = ((V_B \times K_A) \rightarrow R) \rightarrow R$. On the other hand, the call-by-name interpretation starts at the level of continuations: $[B \odot A] = (K_A \times C_B) \rightarrow R$ and from this the computation is obtained as usual, viz. $C_{B \odot A} = ((K_A \times C_B) \rightarrow R) \rightarrow R$, hence obtaining a higher order function than the one computed under the call-by-value strategy. This difference will play an important role in the linguistic application of the two strategies.

Terms: cbv versus cbn. Given the different CPS *types* for left and right (co)implications, we can now turn to their interpretation at the *term* level. In (15), we give the cbv interpretation of terms, in (16) of coterms. We repeat the typing information from (5) to assist the reader. The call-by-name regime is the composition of call-by-value and arrow reversal: $[\cdot] \triangleq [\cdot]^\infty$. This CPS interpretation of terms is set up in such a way that for sequents with yield $A_1, \dots, A_n \Rightarrow B$, the cbv interpretation represents the process of obtaining a B *computation* from A_1, \dots, A_n *values*; the cbn interpretation takes A_1, \dots, A_n *computations* to a B computation. See Propositions 8.1 and 8.3 of [4].

A	$[x] = \lambda k. k \ x$	$x : A$	
$B \setminus A$	$[x \setminus M] = \lambda k. (k \ \lambda \langle x, \beta \rangle. [M] \ \beta)$	$x : B, \ M : A$	
A/B	$[M / x] = \lambda k. (k \ \lambda \langle \beta, x \rangle. [M] \ \beta)$	$x : B, \ M : A$	
$B \odot A$	$[M \succ K] = \lambda k. ([M] \ \lambda y. (k \ \langle y, [K] \rangle))$	$M : A, \ K : B$	(15)
$A \odot B$	$[K \prec M] = \lambda k. ([M] \ \lambda y. (k \ \langle [K], y \rangle))$	$M : A, \ K : B$	
B	$[\mu \alpha. (x * K)] = \lambda \alpha. ([K] \ x)$	$\alpha : B, \ x, K : A$	

A	$\lceil \alpha \rceil = \alpha$	$\alpha : A$
$B \otimes A$	$\lceil \alpha \otimes K \rceil = \lambda \langle \alpha, x \rangle. (\lceil K \rceil \ x)$	$\alpha : B, K : A$
$A \otimes B$	$\lceil K \otimes \alpha \rceil = \lambda \langle x, \alpha \rangle. (\lceil K \rceil \ x)$	$\alpha : B, K : A$
$B \setminus A$	$\lceil M \ltimes K \rceil = \lambda k. (\lceil M \rceil \ \lambda x. (k \ \langle x, \lceil K \rceil \rangle))$	$M : B, K : A$
A / B	$\lceil K \rtimes M \rceil = \lambda k. (\lceil M \rceil \ \lambda x. (k \ \langle \lceil K \rceil, x \rangle))$	$M : B, K : A$
A	$\lceil \tilde{\mu}x. (M * \alpha) \rceil = \lambda x. (\lceil M \rceil \ \alpha)$	$x : A, \alpha, M : B$

(16)

5 Application: Scope Construal

In this section we turn to the linguistic application. Our aim is twofold. First we show that a type assignment $(s \otimes s) \otimes np$ for generalized quantifier phrases solves the problems with $s/(np \setminus s)$ mentioned in §1: the type $(s \otimes s) \otimes np$ uniformly appears in positions that can be occupied by ordinary noun phrases, and it gives rise to ambiguities of scope construal (local and non-local) in constructions with multiple GQ and/or multiple choices for the scope domain. Second, we relate the CPS interpretation to the original interpretation for Lambek derivations by defining translations $\llbracket \cdot \rrbracket, \llbracket \cdot \rrbracket$ lifting the lexical constants from the type they have in the original Lambek semantics to the type required by the cbv or cbn level. To realize this second aim, we assume that the result type R is the type of truth values. For the rest our modeltheoretic assumptions are standard. The domain of interpretation for np values is E (the set of individuals), for s values it is $\{0, 1\}$ (the set of truth values). Out of E and $\{0, 1\}$ one then constructs complex domains in terms of function spaces and Cartesian products. In the case of the original Lambek semantics (or Montague grammar) these domains of interpretation are obtained indirectly, via a mapping between syntactic and semantic types where $np' = e$, $s' = t$ and $(A \setminus B) = (B / A) = A' \rightarrow B'$, with $\text{Dom}_e = E$ and $\text{Dom}_t = \{0, 1\}$.

We start with a fully worked-out example of a ‘ S goes to $NP \ VP$ ’ combination, ‘Alice left’. The official sequent derivation is given in (17). Consulting the dictionary of Table 1, we fill in the lexical items *alice* and *left* of type np and $np \setminus s$ respectively for the x and y parameters of the proof term. Call the resulting term M . We now compare the CPS transformation of M under the cbv and cbn execution regimes as defined in (15) and (16).

$$\begin{array}{c}
 \frac{x : np \xrightarrow{x} np \quad s \xrightarrow{\alpha} \alpha : s}{x : np \circ np \setminus s \xrightarrow{x \ltimes \alpha} \alpha : s} (\setminus L) \\
 \frac{x : np \circ np \setminus s \xrightarrow{x \ltimes \alpha} \alpha : s}{x : np \circ y : np \setminus s \xrightarrow{\mu \alpha. (y * (x \ltimes \alpha))} s} (\equiv)
 \end{array} \quad (17)$$

Consider first cbv, on the left in Figure 4. Recall that under the cbv regime a sequent with yield $A_1, \dots, A_n \Rightarrow B$ maps the A_i values to a B computation. A value of type $np \setminus s$ ($V_{np \setminus s}$), as we see in Table 1, is a function taking a pair of an np value and an s continuation to the result type R , i.e. $V_{np} \times K_s \rightarrow R$.

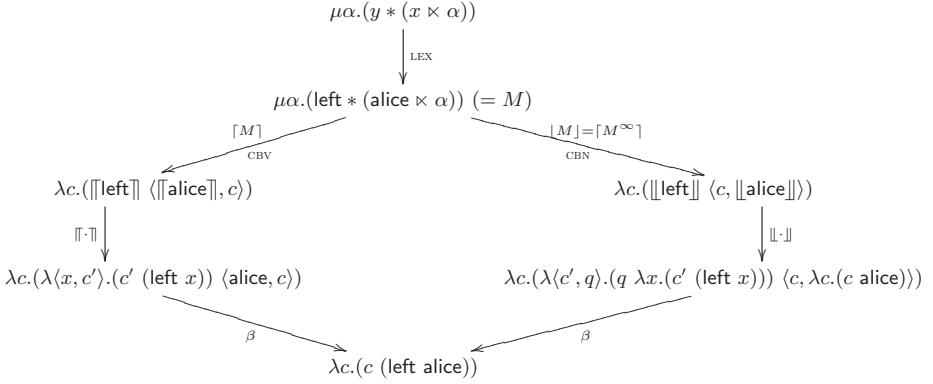


Fig. 4. ‘Alice left’: cbv versus cbn

Combining V_{np} and $V_{np \setminus s}$ we obtain an s computation, i.e. $(s \rightarrow R) \rightarrow R$, by giving the $V_{np \setminus s}$ function the pair it requires, and abstracting over the K_s component. This is what we see in $[M] = \lambda c.([left] \langle [alice], c \rangle)$. The next step is to relate the cbv CPS interpretation to the original semantics of the Lambek calculus. In the original semantics, ‘Alice’ and ‘left’ would be interpreted in terms of constants of type e and $e \rightarrow t$ respectively. The mapping $[[\cdot]]$ in Table 2 produces terms of type V_{np} and $V_{np \setminus s}$ from constants *alice* and *left* of type e and $e \rightarrow t$. Substituting these in $[M]$ (and β conversion) gives $\lambda c.(c \text{ (left alice)})$. Combined with the trivial s continuation (the identity function on $\{0, 1\}$) one obtains (left alice).

On the right in Figure 4 we have the cbn interpretation. Under the cbn regime, a sequent with yield $A_1, \dots, A_n \Rightarrow B$ maps A_i *computations* to a B computation. We obtain the cbn interpretation by duality: $[M] = [M^\infty] = [\tilde{\mu}x.((x \succ \text{alice}) * \text{left})]$, i.e. we read off the cbn interpretation from the mirror image derivation $(np \circ np \setminus s \Rightarrow s)^\infty$, which is $s \Rightarrow s \circ np \circ np$. For the lexical items involved, Table 1 gives the *continuation* types $[A]$ corresponding to the source language types A . To obtain the required types for A *computations*, we take functions from these continuations into R . Specifically, the verb $[[left]]$ in this case is interpreted as a function taking a pair $K_s \times C_{np}$ into R ; the noun phrase argument $[[alice]]$ also is of type C_{np} , i.e. $(V_{np} \rightarrow R) \rightarrow R$. Combining these produces a result of type C_s again by abstracting over the K_s component of the pair given to the verb:

Table 1. Lexical entries. $[A]$ is a value of type A ; $[A]$ is a continuation of type A .

WORD	TYPE	ALIAS	$[\cdot]$ CBV	$[\cdot]$ CBN
alice, lewis	np		$[np]$	$[np]$
left	$np \setminus s$	iv	$R^{[np] \times R^{[s]}}$	$[s] \times R^{[np]}$
teases	$(np \setminus s) / np$	tv	$R^{R^{[iv]} \times [np]}$	$R^{[np]} \times [iv]$
thinks	$(np \setminus s) / s$	tvs	$R^{R^{[iv]} \times [s]}$	$R^{[s]} \times [iv]$
somebody	$s / (np \setminus s)$	su	$R^{R^{[s]} \times [iv]}$	$R^{[iv]} \times [s]$

Table 2. Lifting lexical constants: cbv regime

$\llbracket \text{alice} \rrbracket$	$= \text{alice}$
$\llbracket \text{lewis} \rrbracket$	$= \text{lewis}$
$\llbracket \text{left} \rrbracket$	$= \lambda \langle x, c \rangle. (c \text{ (left } x))$
$\llbracket \text{teases} \rrbracket$	$= \lambda \langle v, y \rangle. (v \lambda \langle x, c \rangle. (c ((\text{teases } y) x)))$
$\llbracket \text{somebody} \rrbracket$	$= \lambda \langle c, v \rangle. (\exists \lambda x. (v \langle x, c \rangle))$

$\llbracket M \rrbracket = \lambda c. (\llbracket \text{left} \rrbracket \langle c, \llbracket \text{alice} \rrbracket \rangle)$. The mapping $\llbracket \cdot \rrbracket$ in Table 3 produces terms of type $[np] \rightarrow R$ and $[np \setminus s] \rightarrow R$ (i.e. computations) from constants `alice` and `left` of type e and $e \rightarrow t$. Substituting these in $\llbracket M \rrbracket$ (and β conversion) gives the same result as what we had under the cbv regime.

Table 3. Lifting lexical constants: cbn regime

$\llbracket \text{alice} \rrbracket$	$= \lambda c. (c \text{ alice})$
$\llbracket \text{lewis} \rrbracket$	$= \lambda c. (c \text{ lewis})$
$\llbracket \text{left} \rrbracket$	$= \lambda \langle c, q \rangle. (q \lambda x. (c \text{ (left } x)))$
$\llbracket \text{teases} \rrbracket$	$= \lambda \langle q, \langle c, q' \rangle \rangle. (q' \lambda x. (q \lambda y. (c ((\text{teases } y) x))))$
$\llbracket \text{somebody} \rrbracket$	$= \lambda \langle v, c \rangle. (\exists \lambda x. (v \langle c, \lambda c'. (c' x) \rangle))$

The reader is invited to go through the same steps for ‘Alice teases Lewis’. The term for the derivation is $\mu \alpha. (\text{teases} * ((\text{alice} \times \alpha) \times \text{lewis})) (= M)$, with the CPS interpretations in (18). The variable v is of type $K_{np \setminus s}$, a verb phrase continuation. Consulting the cbv and cbn dictionaries of Tables 2 and 3, we can substitute the required lambda terms for the lexical constants. After this substitution and β reduction, the cbv and cbn interpretations converge on $\lambda c. (c ((\text{teases lewis}) \text{ alice}))$.

$$\begin{aligned} \llbracket M \rrbracket &= \lambda c. (\llbracket \text{teases} \rrbracket \langle \lambda v. (v \langle \llbracket \text{alice} \rrbracket, c \rangle), \llbracket \text{lewis} \rrbracket \rangle); \\ \llbracket M \rrbracket &= \lambda c. (\llbracket \text{teases} \rrbracket \langle \llbracket \text{lewis} \rrbracket, \langle c, \llbracket \text{alice} \rrbracket \rangle \rangle). \end{aligned} \quad (18)$$

In Fig 5 we highlight the type structure of the CPS interpretations for this sentence, showing that (i) call-by-value produces terms consisting of function applications of values to pairs of values and continuations (left tree), whereas (ii) call-by-name produces terms consisting of the application of computation to pairs of computations and continuation types. The observed difference will be relevant for the interpretation of generalized quantifiers expressions to which we now turn.

Scope construal: simple subject GQ. In Table 1, one finds the CPS image under cbv and cbn of a Lambek-style $s/(np \setminus s)$ type assignment for a GQ expression such as ‘somebody’. The corresponding lexical recipes for the cbv and cbn regimes is given in Tables 2 and 3, respectively. We leave it as an exercise for the reader to work through a derivation with these types/terms and to verify that a type assignment $s/(np \setminus s)$ is restricted to subject position and to local scope, as

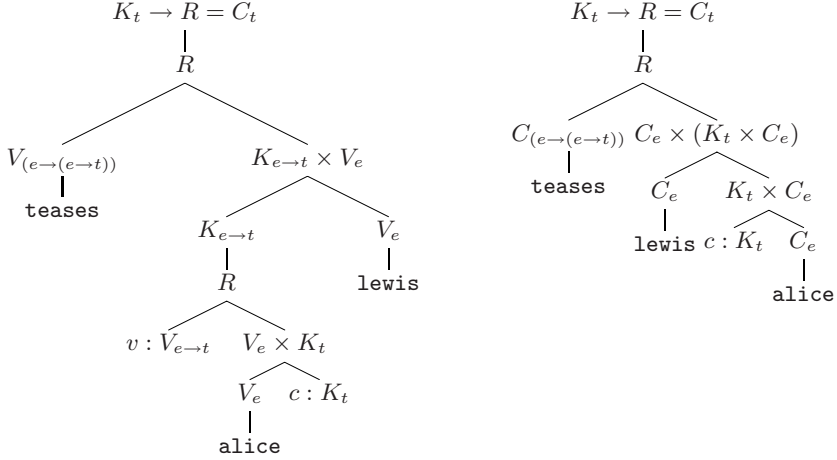


Fig. 5. Type schemas for cbv (left) versus cbn (right)

we saw in §1. Let us turn then to derivations with the type assignment we have proposed: $(s \odot s) \odot np$ (alias: gq) — a type assignment that will accommodate both local and non-local scope construals. In (19) we compute the term for the derivation of ‘somebody left’, using the abbreviatory conventions discussed in the Appendix (e.g., the $\boxed{1}$ in step 2. stands for the proof term computed at step 1.); in (20) its CPS transformation under cbv and cbn. ($z : V_s$, $q : C_{np}$, $y : K_{s \odot s}$)

$$\begin{array}{ll}
 1. \quad \mu\alpha.(\text{left}*(x\alpha)) & \begin{array}{c} s^\circ \\ \swarrow \quad \searrow \\ np \quad (np \setminus s) \\ | \quad | \\ x \quad \text{left} \end{array} \\
 2. \quad \mu\beta.(\text{somebody}*\text{bind}(x, \boxed{1}, \beta)) & \begin{array}{c} s^\circ \\ \swarrow \quad \searrow \\ gq \quad (np \setminus s) \\ | \quad | \\ \text{somebody} \quad \text{left} \end{array}
 \end{array} \quad (19)$$

$$\begin{aligned}
 \boxed{2} &= N & [N] &= \lambda c.((\llbracket \text{left} \rrbracket \langle \pi^2 \llbracket \text{somebody} \rrbracket, \lambda z.(\pi^1 \llbracket \text{somebody} \rrbracket \langle z, c \rangle))) ; \\
 & & [N] &= \lambda c.((\llbracket \text{somebody} \rrbracket \lambda \langle q, y \rangle. (y \langle c, \lambda c'. (\llbracket \text{left} \rrbracket \langle c', q \rangle))))).
 \end{aligned} \quad (20)$$

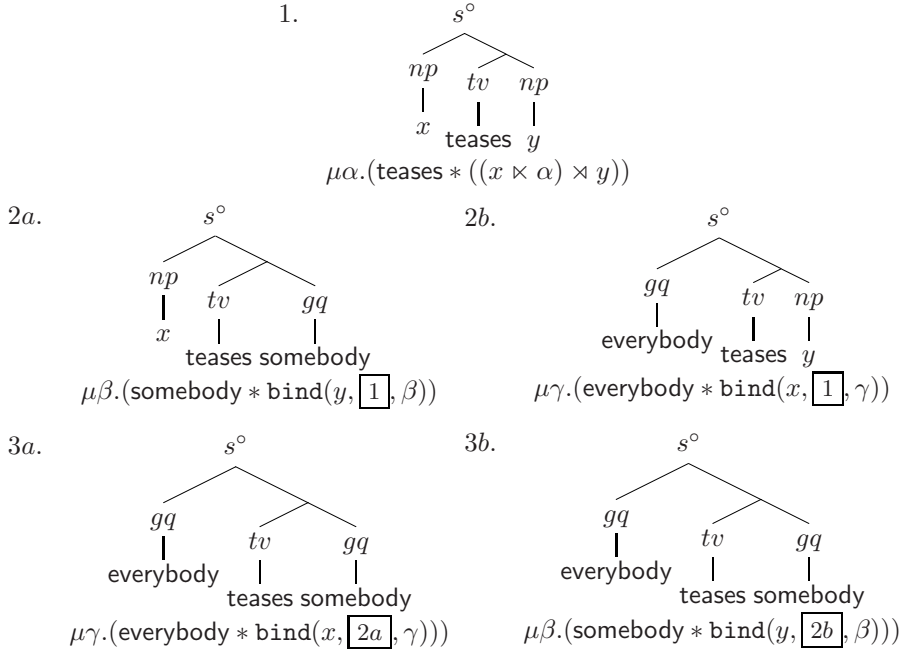
The difference between cbv and cbn regimes observed above with respect to $\llbracket \cdot \rrbracket$, $\llbracket \cdot \rrbracket$ in the term in (18) turns out to be of particular interest here. In (21) we give V_{gq} and K_{gq} for the cbv and cbn interpretations respectively. For cbv, this is a pair of an $s \odot s$ continuation and an np value. This np value would have to be realised as the variable bound by the (logical) constant \exists (of type $(e \rightarrow t) \rightarrow t$) in the $\llbracket \cdot \rrbracket$ translation. Such a binding relation cannot be established from the $K_{s \odot s}$ component of the pair. For cbn, the situation is different: $\llbracket gq \rrbracket$ has the required structure to specify the lifted lexical recipe of (22). \mathcal{Q} is a function taking a pair of a np computation and an $(s \odot s)$ continuation to the result type R . In the body of the term, we apply \mathcal{Q} to the C_{np} term $\lambda k.(k \ x)$, where x is the e type variable bound by \exists , and to the closed term $\lambda \langle c, p \rangle. (p \ c)$ obtained by applying

the C_s variable p to the K_s variable c . In combination with $\llbracket \text{left} \rrbracket$ given above, $\llbracket N \rrbracket$ simplifies to $\lambda c.(\exists \lambda x.(c (\text{left } x)))$ as required. From here on, we stay with the cbn regime.

$$[gq] = R^{[s] \times R^{[s]}} \times [np] \quad [gq] = R^{R^{[np]} \times R^{[s] \times R^{[s]}}}. \quad (21)$$

$$\llbracket \text{somebody} \rrbracket = \lambda \mathcal{Q}.(\exists \lambda x.(\mathcal{Q} \langle \lambda k.(k \ x), \lambda \langle c, p \rangle.(p \ c) \rangle)). \quad (22)$$

Ambiguous scope construal. First, compare ‘Alice teases Lewis’ with ‘everyone teases somebody’ involving two GQ expressions. Starting from $\boxed{1} \mu\alpha.(\text{teases} * ((x \times \alpha) \times y))$, the derivation can be completed in two ways, depending on whether we first bind the object variable y , then the subject variable x , or vice versa. On the left the subject wide scope reading, on the right the object wide scope reading.



By applying the method described with previous examples, one obtains the \forall/\exists reading for (3a) and the \exists/\forall reading for (3b). First, the terms are interpreted by means of the definitions in (15) and (16) obtaining the following results:

$$\begin{aligned}
 [3a] &= \lambda c.(\llbracket \text{evro} \rrbracket \lambda \langle q, y \rangle.(\llbracket \text{smo} \rrbracket \lambda \langle p, z \rangle.(y \langle c, \lambda c'.(z \langle c', \lambda c''.(\llbracket \text{teases} \rrbracket \langle p, \langle c', q \rangle \rangle)))))); \\
 [3b] &= \lambda c.(\llbracket \text{smo} \rrbracket \lambda \langle p, z \rangle.(\llbracket \text{evro} \rrbracket \lambda \langle q, y \rangle.(z \langle c, \lambda c'.(y \langle c', \lambda c''.(\llbracket \text{teases} \rrbracket \langle p, \langle c', q \rangle \rangle)))))). \quad (23)
 \end{aligned}$$

where the variables p, q of type C_{np} are for the object and subject respectively. The y, z variables are $s \oslash s$ continuations, the (primed) c are s continuations. Secondly, the $\llbracket \cdot \rrbracket$ translation is applied, and the readings reduce to $\lambda c.(\forall \lambda x.(\exists \lambda y.(c ((\text{teases } y) \ x))))$ and $\lambda c.(\exists \lambda y.(\forall \lambda x.(c ((\text{teases } y) \ x))))$, respectively.

Local versus non-local scope. Consider the two readings for the sentence ‘Alice thinks somebody left’. The ambiguity arises here from the fact that in this context the GQ can non-deterministically select the embedded or the main clause s as its scope domain. We give the terms for these two derivations (local (a) versus non-local (b) scope) in (24), reusing the components $\boxed{1}$ and $\boxed{2}$ of (19), and the cbn interpretations of these terms in (25). These can be further reduced to (26) via a lexical recipe $\llbracket \text{thinks} \rrbracket = \lambda \langle p, \langle c, q \rangle \rangle. (q \lambda x. (c ((\text{thinks } (p \lambda c. c)) x)))$ expressed in terms of a constant thinks of type $t \rightarrow (e \rightarrow t)$.

$$\begin{aligned} a. & \quad \mu\gamma. (\text{thinks} * ((\text{alice} \times \gamma) \times \boxed{2})); \\ b. & \quad \mu\gamma. (\text{somebody} * \text{bind}(x, \mu\gamma'. (\text{thinks} * ((\text{alice} \times \gamma') \times \boxed{1})), \gamma)). \end{aligned} \quad (24)$$

$$\begin{aligned} [a] &= \lambda c. (\llbracket \text{thinks} \rrbracket \langle \lambda c'. (\llbracket \text{somebody} \rrbracket \lambda \langle q, y \rangle. (y \langle c', \lambda c''. (\llbracket \text{left} \rrbracket \langle c'', q \rangle))), \langle c, \llbracket \text{alice} \rrbracket \rangle)); \\ [b] &= \lambda c. (\llbracket \text{somebody} \rrbracket \lambda \langle q, y \rangle. (y \langle c, \lambda c'. (\llbracket \text{thinks} \rrbracket \langle \lambda c''. (\llbracket \text{left} \rrbracket \langle c'', q \rangle), \langle c', \llbracket \text{alice} \rrbracket \rangle)))). \end{aligned} \quad (25)$$

$$\begin{aligned} [a] &\rightsquigarrow_{\beta} \lambda c. (c ((\text{thinks } (\exists \text{ left})) \text{ alice})); \\ [b] &\rightsquigarrow_{\beta} \lambda c. (\exists \lambda y. (c ((\text{thinks } (\text{left } y)) \text{ alice}))). \end{aligned} \quad (26)$$

6 Conclusions, Further Directions

In this paper we have moved from asymmetric Lambek calculus with a single succedent formula to the symmetric Lambek-Grishin calculus, where both the antecedent and the succedent are formula structures, configured in terms of \otimes and \oplus respectively, and where the \otimes and \oplus environments can interact in a structure-preserving way. This move makes it possible to import into the field of natural language semantics the powerful tools of $\lambda\mu$ -calculus. The main attraction of the proposed continuation semantics, in our view, lies in the fact that **LG** allows us to fully exploit the duality between Lambek’s directional $/, \backslash$ implications and the corresponding directional \oslash, \ominus difference operations, at the level of syntax and at the level of semantics. We thus restore Curry’s original idea of an *isomorphism* between proofs and terms, rather than the weaker homomorphic view of standard Lambek (or Montagovian) semantics.

Our approach differs in a number of respects from the related work cited in §1. Abstracting away from the directionality issue, de Groote’s original application of $\lambda\mu$ calculus to scope construal syntactically types generalized quantifier phrases as np with meaning representation $\mu\alpha^{K_e} (\exists \alpha)$. As a result, a sentence with multiple GQ phrases is associated with a *unique* parse/term; the multiple readings for that term are obtained as a result of the non-confluence of $\lambda\mu$ calculus, which is considered as a feature, not a bug. Our approach in contrast is true to the principle that multiple readings can only arise as the images of distinct proofs, given the Curry-Howard isomorphism between proofs and terms. Barker [19,11] uses a simplified continuation semantics, which lifts types A to $(A \rightarrow R) \rightarrow R$ ‘externally’, without applying the CPS transformation to the internal structure of complex types. This breaks the symmetry which is at the heart of our dual treatment of $/, \backslash$ vs \oslash, \ominus . The structural-rule account of scope flexibility in [12,13] suffers from commutativity problems.

The approach described here, like Hendriks's type-shifting approach, creates all combinatorial possibilities for scope construal. However, it is well known that, depending on the choice of particular lexical items, many of these construals will in fact be unavailable. Bernardi [20] uses the control modalities \Diamond, \Box to calibrate the scopal behaviour of particular classes of GQ expressions. Adding \Diamond, \Box and a pair of dually residuated modalities to **LG** is straightforward. In a follow-up paper, we plan to study the continuation semantics of these operations, relating them to the **shift** and **reset** constructs one finds in the theory of functional programming languages and that have been considered in [11,13].

Finally, the interpretation given here construes scopal ambiguities in a *static* setting. In a recent paper, de Groote [21] develops a continuation-based approach towards *dynamic* interpretation. A natural topic for further research would be to investigate how to incorporate this dynamic perspective in our setting, and how to extend the approach of [21] with the difference operations and the concomitant Grishin interaction principles.

References

1. Lambek, J.: The mathematics of sentence structure. *American Mathematical Monthly* 65, 154–170 (1958)
2. Lambek, J.: On the calculus of syntactic types. In: Jakobson, R. (ed.) *Structure of Language and Its Mathematical Aspects*. American Mathematical Society, 166–178 (1961)
3. Grishin, V.: On a generalization of the Ajdukiewicz-Lambek system. In: *Studies in Nonclassical Logics and Formal Systems*. Nauka, Moscow, pp. 315–334 [English translation in Abrusci and Casadio (eds.) *Proceedings 5th Roma Workshop*, Bulzoni Editore, Roma, 2002] (1983)
4. Curien, P., Herbelin, H.: Duality of computation. In: *International Conference on Functional Programming (ICFP'00)*, pp. 233–243 [2005: corrected version] (2000)
5. Moortgat, M.: Generalized quantifiers and discontinuous type constructors. In: Bunt, H., van Horck, A. (eds.) *Discontinuous Constituency*, pp. 181–207. Walter de Gruyter, Berlin (1996)
6. Hendriks, H.: *Studied flexibility. Categories and types in syntax and semantics*. PhD thesis, ILLC, Amsterdam University (1993)
7. Morrill, G.: Discontinuity in categorial grammar. *Linguistics and Philosophy*. 18, 175–219 (1995)
8. Morrill, G., Fadda, M., Valentin, O.: Nondeterministic discontinuous Lambek calculus. In: *Proceedings of the Seventh International Workshop on Computational Semantics (IWCS7)*, Tilburg (2007)
9. Lambek, J.: From categorial to bilinear logic. In: Schröder-Heister, K.D.P. (ed.) *Substructural Logics*, pp. 207–237. Oxford University Press, Oxford (1993)
10. de Groote, P.: Type raising, continuations, and classical logic. In: van Rooy, R., (ed.) *Proceedings of the Thirteenth Amsterdam Colloquium*, ILLC, Universiteit van Amsterdam, pp. 97–101 (2001)
11. Barker, C.: Continuations in natural language. In: Thielecke, H. (ed.) *CW'04: Proceedings of the 4th ACM SIGPLAN continuations workshop*, Tech. Rep. CSR-04-1, School of Computer Science, University of Birmingham, pp. 1–11 (2004)

12. Barker, C., Shan, C.: Types as graphs: Continuations in type logical grammar. *Language and Information* 15(4), 331–370 (2006)
13. Shan, C.: Linguistic side effects. PhD thesis, Harvard University (2005)
14. Selinger, P.: Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Math. Struct. in Comp. Science* 11, 207–260 (2001)
15. Wadler, P.: Call-by-value is dual to call-by-name. In: *ICFP*, Uppsala, Sweden (August 2003)
16. Moortgat, M.: Symmetries in natural language syntax and semantics: the Lambek-Grishin calculus (this volume). In: Leivant, D., de Queiros, R. (eds.) *WoLLIC'07. Proceedings 14th Workshop on Logic, Language, Information and Computation*. LNCS, vol. 4576, Springer, Heidelberg (2007)
17. Goré, R.: Substructural logics on display. *Logic Journal of IGPL* 6(3), 451–504 (1997)
18. de Groote, P., Lamarche, F.: Classical non-associative Lambek calculus. *Studia Logica* 71, 335–388 (2002)
19. Barker, C.: Continuations and the nature of quantification. *Natural language semantics* 10, 211–242 (2002)
20. Bernardi, R.: Reasoning with Polarity in Categorical Type Logic. PhD thesis, Utrecht Institute of Linguistics OTS (2002)
21. de Groote, P.: Towards a Montagovian account of dynamics. In: *Proceedings SALT 16*, CLC Publications (2006)

A Shorthand Format for Sequent Derivations

As an example of the **LG** term assignment, (27) gives the derivation showing how one obtains a Lambek-style GQ type $C/(A \setminus B)$ from a $(B \otimes C) \otimes A$ source.

$$\begin{array}{c}
 \frac{z : A \xrightarrow{z} A \quad B \xrightarrow{\gamma} \gamma : B}{z : A \circ A \setminus B \xrightarrow{z \times \gamma} \gamma : B} (\setminus L) \\
 \frac{\quad}{z : A \circ A \setminus B \xrightarrow{z \times \gamma} \gamma : B} (\Leftarrow) \\
 \frac{C \xrightarrow{\alpha} \alpha : C \quad z : A \circ y : A \setminus B \xrightarrow{\mu\gamma.(y * (z \times \gamma))} B}{z : A \circ y : A \setminus B \xrightarrow{(\mu\gamma.(y * (z \times \gamma))) \succ \alpha} B \otimes C \circ \alpha : C} (\otimes R) \\
 \frac{\quad}{z : A \circ y : A \setminus B \xrightarrow{(\mu\gamma.(y * (z \times \gamma))) \succ \alpha} B \otimes C \circ \alpha : C} (\Leftarrow) \\
 \frac{A \circ y : A \setminus B \xrightarrow{\tilde{\mu}z.((\mu\gamma.(y * (z \times \gamma))) \succ \alpha) * \beta} \beta : B \otimes C \circ \alpha : C}{(B \otimes C) \otimes A \circ y : A \setminus B \xrightarrow{\beta \otimes (\tilde{\mu}z.((\mu\gamma.(y * (z \times \gamma))) \succ \alpha) * \beta)} \alpha : C} (\otimes L) \\
 \frac{\quad}{(B \otimes C) \otimes A \circ y : A \setminus B \xrightarrow{\beta \otimes (\tilde{\mu}z.((\mu\gamma.(y * (z \times \gamma))) \succ \alpha) * \beta)} \alpha : C} (\Leftarrow) \\
 \frac{x : (B \otimes C) \otimes A \circ y : A \setminus B \xrightarrow{\mu\alpha.(x * (\beta \otimes (\tilde{\mu}z.((\mu\gamma.(y * (z \times \gamma))) \succ \alpha) * \beta)))} C}{x : (B \otimes C) \otimes A \xrightarrow{(\mu\alpha.(x * (\beta \otimes (\tilde{\mu}z.((\mu\gamma.(y * (z \times \gamma))) \succ \alpha) * \beta)))} / y} C/(A \setminus B)} (/R)
 \end{array} \tag{27}$$

This example shows that except for the most simple derivations, the sequent tree format soon becomes unwieldy. Below we introduce a more user-friendly line format, which graphically highlights the tree structure of the antecedent and succedent parts. In the line format, each row has (a) a line number, (b) the (co)term for the currently active formula, and the antecedent (c) and succedent (d) structures in tree format. The cursor singles out the currently active formula. It takes the form \cdot^\bullet in the antecedent, and \cdot° in the succedent. With \boxed{n} we refer to the (co)term at line n . Compare (27) with the derivation for a sentence ‘Somebody left’ in line format.

LINE (CO)TERM	LHS TREE	RHS TREE
1. x	np	np°
2. α	x s^\bullet	s
3. $\boxed{1} \ltimes \boxed{2}$	$np \ (np \setminus s)^\bullet$	α
4. $\mu\alpha.(\text{left} * \boxed{3})$	x $np \ (np \setminus s)$	s
5. β	x left s^\bullet	α
6. $\boxed{4} \succ \boxed{5}$	s^\bullet	s°
7. $\tilde{\mu}x.(\boxed{6} * \gamma)$	$np \ (np \setminus s)$	s
8. $\gamma \oslash \boxed{7}$	x left $np^\bullet \ (np \setminus s)$	$(s \oslash s)^\circ \ s$
9. $\mu\beta.(\text{somebody} * \boxed{8})$	left $((s \oslash s) \oslash np)^\bullet \ (np \setminus s)$	β
	left $((s \oslash s) \oslash np) \ (np \setminus s)$	$(s \oslash s) \ s$
	somebody left	γ β
		s
		β
		s°

We reduce the length of a derivation further using the (qL) and (qR) rules of inference discussed in the main text and *folding*: a sequence of n $(/, \setminus R)$ (respectively $(\oslash, \oslash L)$) one-premise inferences is folded into a one-step one-premise inference; a (\Leftarrow) step (or (\Rightarrow) respectively) followed by a sequence of n $(/, \setminus L)$ (respectively $(\oslash, \oslash R)$) two-premise inferences is folded into a one-step $n + 1$ premise inference; an n premise inference with m non-lexical axiom premises is contracted to an $n - m$ premise inference. Where the succedent (antecedent) tree is just a point, we write the highlighted formula as the root of the antecedent (succedent) tree. The result of applying these conventions for the example sentence ‘somebody left’ is (19) in the main text.

Ehrenfeucht–Fraïssé Games on Linear Orders

Ryan Bissell-Siders

Ryan Bissell-Siders
ryan.bissell-siders@helsinki.fi
<http://www.math.helsinki.fi/>

Abstract. We write strategies for both players in the Ehrenfeucht–Fraïssé game played between two linear orders. We prove that our strategies win whenever possible. The strategy in a game of fixed, finite length focuses the attention of both players on certain finite sets called “total information.” These finite sets are semimodels in the sense that if one player does not have a winning strategy, then the other player can win, even when forced to choose only elements of a model which are indicated by the total information.

These sets are semimodels—finite, nested models, such that a formula is satisfied in a linear order iff the formula is semimodel-satisfied in the corresponding semimodel. The strategy implies the decidability of the theory of linear order, and gives a completion of any formula by one with quantifier rank only 2 larger than the original formula. The strategy generalizes directly to the infinitary theory of linear order.

Keywords: Ehrenfeucht–Fraïssé game, semimodel, decision procedure, linear order, completion.

1 “Total Information” Is Sufficient for the EF Game

For any linear order λ , let λ^+ be the ordered set of cuts in λ , i.e.

Definition 1. $\lambda^+ = (\{(X, Y) : X \cup Y = \lambda, \forall x \in X \forall y \in Y x < y\}; (X, Y) \leq (W, Z) \text{ iff } X \subseteq W)$.

Definition 2. For any linear order λ and element x , let $Th_k^{\text{loc}}(\lambda, x)$ be $\{\text{formulas } \phi \text{ of quantifier rank } k \text{ with the single variable } v \text{ free} : \exists E \forall A, B, C, D (A + E + B + \lambda + C + E + D \models_{[x/v]} \phi(v))\}$ where A, B, C, D, E range over linear orders.

Our goal is to prove theorem 1, i.e., to find in any linear order λ and for any natural number k a finite set $I_k(\lambda) \subseteq \lambda^+$ such that $\lambda \equiv_k \lambda'$ iff $I_k(\lambda) = I_k(\lambda')$ and the same $k - 1$ -quantifier local types $Th_{k-1}^{\text{loc}}(\lambda, x)$ are realized in both λ and λ' at and between the elements of I_k .

Definition 3. Let Seq be the set of decreasing sequences of natural numbers. For k any natural number, let $\text{Seq}(k)$ be the set of descending sequences of natural numbers $\leq k - 2$. Let $<^{\text{seq}}$ be lexicographical ordering on Seq .

Definition 4. For λ any linear order and $(a, b) = ((L_0, L_1), (R_0, R_1))$ any pair of adjacent elements of λ^+ , define $\lambda^{(a,b)} = \{x \in \lambda : (x \in L_1 \wedge \exists z(z \in L_1 \wedge z < x)) \wedge (x \in R_0 \wedge \exists z(z \in R_0 \wedge z > x))\}$.

Definition 5. For λ any linear order, we define $I_\emptyset(\lambda) = \emptyset$. If $s \in \text{Seq}$ and $I_s(\lambda)$ is defined and $s <^{\text{seq}} s'$ and s' is the $<^{\text{seq}}$ -successor of s , let l be the least member of s' , and let $I_{s'}(\lambda)$ contain $I_s(\lambda)$ and for each adjacent pair of elements of $I_s(\lambda)$, elements of λ^+ representing the appearance and disappearance of each local l -quantifier type. Precisely:

For each pair $(a, b), a < b$, of adjacent elements of $I_s(\lambda)$ and for each $t \in \{Th_l^{\text{loc}}(\lambda, z) : z \in \lambda^{(a,b)}\}$, define the cuts $(A_t^{(a,b)}, B_t^{(a,b)}) \in \lambda^+$ and $(C_t^{(a,b)}, D_t^{(a,b)}) \in \lambda^+$ as follows:

- $A_t^{(a,b)} = \{x \in \lambda : \forall z \in \lambda^{(a,b)}(t(z) \rightarrow z > x)\}$,
- $B_t^{(a,b)} =$ the complement of A_t , i.e. $\{x \in \lambda : \exists z \in \lambda^{(a,b)}(t(z) \wedge z \leq x)\}$,
- $C_t^{(a,b)} = \{x \in \lambda : \exists z \in \lambda^{(a,b)}(t(z) \wedge z \geq x)\}$,
- $D_t^{(a,b)} =$ the complement of C_t , i.e. $\{x \in \lambda : \forall z \in \lambda^{(a,b)}(t(z) \rightarrow z < x)\}$.

If the cut $a = (L_0, L_1) \in \lambda^+$ was created as $(C_u^{(a_1, b_1)}, D_u^{(a_1, b_1)})$ (rather than $(A_u^{(a_1, b_1)}, B_u^{(a_1, b_1)})$) and $u = Th_m^{\text{loc}}(\lambda, w)$ and $m > l$ and $\forall x \in C_u \exists y \in C_u (y > x \wedge t(y))$, then we say a covers t . Similarly, if $b = (R_0, R_1)$ was created as $(A_u^{(a_1, b_1)}, B_u^{(a_1, b_1)})$ (rather than $(C_u^{(a_1, b_1)}, D_u^{(a_1, b_1)})$) and u is a local type of higher quantifier rank than t and $\forall x \in B_u \exists y \in B_u (y < x \wedge t(y))$, then we say b covers t .

Let $I_{s', (a,b)} = \{(A_t^{(a,b)}, B_t^{(a,b)}) : (\exists z \in \lambda^{(a,b)}(t = Th_l^{\text{loc}}(\lambda, z))) \wedge a \text{ does not cover } t\} \cup \{(C_t^{(a,b)}, D_t^{(a,b)}) : (\exists z \in \lambda^{(a,b)}(t = Th_l^{\text{loc}}(\lambda, z))) \wedge b \text{ does not cover } t\}$ be the set of first and last appearances of each local type t in the interval $\lambda^{(a,b)}$.

Let $I_{s'} = I_s \cup \cup \{I_{s', (a,b)} : (a, b) \text{ is a pair of adjacent elements of } I_s\}$.

For any $k \geq 2$, let $I_k(\lambda) = \cup I_s(\lambda) : \lambda \in \text{Seq}(k)$. (This choice of index is made because of lemma 1.)

Definition 6. For any linear orders λ and λ' and s a decreasing sequence of natural numbers, we define $\lambda \equiv_s \lambda'$ by induction on s . Say $\lambda \equiv_\emptyset \lambda'$ holds for any linear orders. Let $s \in \text{Seq}$ and s' be the $<^{\text{seq}}$ successor of s and let l be the least number in s' as in definition 4. Then $\lambda \equiv_{s'} \lambda'$ iff $\lambda \equiv_s \lambda'$ and for any pair of adjacent elements of I_s $(a, b) = ((L_0, L_1), (R_0, R_1))$, the cuts of definition 5, for any two types $t, u \in \{Th_l^{\text{loc}}(\lambda, z) : z \in \lambda^{(a,b)}\}$ satisfy each of the following conditions in λ iff they satisfy the same condition in λ' :

1. $(A_t^{(a,b)}, B_t^{(a,b)}) = (L_0, L_1) = a$ or $(C_t^{(a,b)}, D_t^{(a,b)}) = (R_0, R_1) = b$.
2. $B_t^{(a,b)} \cap C_t^{(a,b)}$ is a singleton and $\neg(a \text{ covers } t \text{ and } b \text{ covers } t)$.
3. $B_t^{(a,b)}$ has a least element, and a does not cover t .
4. $C_t^{(a,b)}$ has a greatest element, and b does not cover t .
5. $\exists z \in (B_t^{(a,b)} \setminus B_u^{(a,b)})$ and a does not cover u .
6. $\exists z \in (B_t^{(a,b)} \cap C_u^{(a,b)})$ and $\neg(a \text{ covers } t \text{ and } b \text{ covers } u)$.

7. $\exists z \in A_t^{(a,b)} \cap D_u^{(a,b)}$ and a doesn't cover t and b doesn't cover u .
8. $B_t^{(a,b)} \cap C_t^{(a,b)} = \emptyset$.

Lemma 1. *If $\lambda \equiv_k \lambda'$, then for all $s \in \text{Seq}(k)$, $\lambda \equiv_s \lambda'$.*

Proof: Suppose $\lambda \equiv_s \lambda'$, but $\lambda \not\equiv_{s'} \lambda'$, where s' is the $<^{\text{seq}}$ -successor of s . We show that player I has a winning strategy in $EF_k(\lambda, \lambda')$, hence $\lambda \not\equiv_k \lambda'$.

We first prove, by induction, claim k: If, in the game $EF_k(\lambda, \lambda')$, player I plays the first move in (a_s, b_s) for all $s \in \text{Seq}(k)$, where if s' is the $<^{\text{seq}}$ -successor of s and $s < s' \in \text{Seq}(k)$, then $\{a_{s'}, b_{s'}\} \subseteq I_{s'} \setminus I_s$ and $a_{s'} = (L(a_{s'}), R(a_{s'}))$ and is defined in $I_{s'}$ as $(A_{u(a_{s'})}^{(a_s, b_s)}, B_{u(a_{s'})}^{(a_s, b_s)})$ where $u(a_{s'})$ was not covered by a_s , or $a_{s'} = (C_{u(a_{s'})}^{(a_s, b_s)}, D_{u(a_{s'})}^{(a_s, b_s)})$ where $u(b_{s'})$ was not covered by b_s , then player II must answer in (a_s, b_s) , for all $s \in \text{Seq}(k)$.

Proof: suppose player II answer inside (a_r, b_r) , for all $r \leq s$, but outside $(a_{s'}, b_{s'})$. Without loss of generality, suppose player I plays x_0 above $a_{s'}$, and player II plays y_0 below $a_{s'}$. Writing $a_s = (L(a_{s'}), R(a_{s'}))$, this means that player I played in $R(a_{s'})$, and player II played in $L(a_{s'})$.

Let the least number in s' be l , so that $u(a_{s'}) = Th_l^{\text{loc}}(\lambda, x)$ is a local type of quantifier depth l . If x_0 is the least element of $R(a_{s'})$, or the greatest element of $L(a_{s'})$, then player II has lost, for then x_0 has quantifier-rank- l local type $u(a_{s'})$, and there is no $y \in \lambda'(a_s, b_s)$ such that $Th_k^{\text{loc}}(\lambda', y) = u(a_{s'})$ and $y < a_{s'}$, by the formula defining A and B or C and D in definition 5.

Let the greatest number in s' be $j \leq k - 2$; let j^- be the $<^{\text{seq}}$ -predecessor of (j) , with domain $\{i : i < j\}$. Player I plays x_1 at an element of λ of type $u(a_{(j)})$ near $a_{(j)}$ in (a_{j^-}, b_{j^-}) , where “near” means: if $a_j = (A_{u(a_{(j)})}^{(a_{j^-}, b_{j^-})}, B_{u(a_{(j)})}^{(a_{j^-}, b_{j^-})})$, then player I plays $x_1 \geq a_j$ in λ and x_1 should be the least element of $B_{u(a_{(j)})}^{(a_{j^-}, b_{j^-})}$ if there is one, and otherwise, for each $t \in \{Th_{j-1}^{\text{loc}}(\lambda, z) : z \in \lambda\}$ (a finite set) which is realized for some $z \in \lambda^{(a_{j^-}, b_{j^-})}$, if realizations of t are bounded from below in $B_{u(a_{(j)})}^{(a_{j^-}, b_{j^-})}$, then all of those realizations occur above x_1 , and if t is realized below every $y \in B_{u(a_{(j)})}^{(a_{j^-}, b_{j^-})}$, then t is realized a large number of times (say, $10 + 2^{10+2^{10+k}}$ or more) above x_1 . This is done to assure that $I_{(j) \cup r}(\lambda^{>x_1})$ will contain an $\equiv_{(j)}$ copy of $I_{(j) \cup r}(\lambda^{>a_s})$ whenever r is a descending sequence of numbers $< j$. On the other hand, if $a_j = (C_{u(a_{(j)})}^{(a_{j^-}, b_{j^-})}, D_{u(a_{(j)})}^{(a_{j^-}, b_{j^-})})$, then player I plays $x_1 \leq a_j$ in λ' near the cut $a_{(j)}$, where, again, “near” means that x_1 bounds to the left all quantifier-rank $< j$ local types which can be bound to the left.

If player I plays $x_1 < a_{(j)} = (C, D) < y_0 \in \lambda'$, and player II plays y_1 in the interval (a_{j^-}, b_{j^-}) and $y_1 < a_{(j)} \in I_{(j)}$, then we may consider the game $EF_{k-1}(\lambda^{>y_1}, \lambda'^{>x_1})$, in which player I has played x_0 , and player II has answered with y_0 . We will show that this violates the induction hypothesis. We check, for r any decreasing sequence of numbers $< j$, that the set I_r defined in $\lambda^{>x_1}$ and $\lambda'^{>y_1}$ are $\equiv_{(j)}$ to the sets $I(k \cup r)$ defined in λ and λ' to the right of $a_{(j)}$. In

this case, x_1 was played close to a_s so that the only $t \in \{Th_{j-1}^{\text{loc}}(\lambda, z) : z \in \lambda\}$ which are realized to the right of x_1 are those t which are realized to the right of a_s or which are covered by a_s . A cut c enters $I(k \cup r)$ only if it was not covered by $a_{(j)}$. Therefore, player I was able to play $x_1 < a_{(j)}$ large enough that no elements of type c exist in $(x_1, a_{(j)})$, when $c \in I(k \cup r)$. So, by claim $k - 1$, player II loses. If player I plays $x_1 < a_{(s)} = (C, D) < y_0 \in \lambda'$, and player II plays y_1 outside the interval (a_{j-}, b_{j-}) , then $y_1 < a_{j-}$, then we may consider the game $EF_{k-1}(\lambda^{<a_{(j)}}, \lambda'^{<a_{(j)}})$ in which player I has played x_1 and player II has played y_1 . Player II's move disrespects I_s , for $s \in \text{Seq}(j)$. If player I plays $x_1 > a_{(j)} = (A, B) \in \lambda$, of the type for which $a_{(j)}$ was defined, and player II plays $y_1 < a_{(j)} \in \lambda$, then we know that y_1 is not of the same j -quantifier local type as x_1 . This concludes the proof of claim k .

Now we assume $\lambda \equiv_s \lambda'$ and that player II must answer player I's moves at the same elements or in the same interval between elements of I_s , and we show that $\lambda \not\equiv_{s'} \lambda'$ gives player I a winning strategy. The definition of $\equiv_{s'}$ lists conditions under which it fails, so we show that player I can exploit each of those eventualities and win. Suppose that $\equiv_{s'}$ fails in the interval (a, b) of I_s .

Case 1: Suppose $(A_t^{(a,b)}, B_t^{(a,b)}) = (L_0, L_1)$ in λ , but not in λ' . That is, elements of type t exist below any $z \in \lambda^{(a,b)}$. Then player I plays $x_0 \in \lambda'^{(a,b)}$ below every element of type t in $\lambda'^{(a,b)}$, where t has quantifier rank l , the least element of s' . By Claim k , player II answers with $y_0 \in \lambda'^{(a,b)}$. Now, for each initial segment s_j of s the least element of which is $> l$, player I plays x_j near a_{s_j} in $(a_{(s_j)-}, b_{(s_j)-})$, as described in claim k , and player II answers, by claims $k - 1 \dots k - j$, in the same intervals. After both players have played x_j and y_j close to a_{s_j} , for all initial segments s_j , there is an element z of type t below y_0 and above x_j or y_j (say, x_j) in λ' . There may well be an element z' of type t below x_0 and above y_j in λ , but for any such z' , $I_l(y_j, z') \not\equiv_l I_l(y_j, a_s)$. So player I plays $x_{j+1} = z$, and as the remaining l -many moves would prove that $Th_l(\lambda, y_{j+1}) \neq t$, player II plays some z' of type t below a_{s_j} . Player I proceeds, as in claim l , to prove that $I_l(y_j, z') \not\equiv_l I_l(y_j, a_s)$.

As a result of case 1, we could add the condition – a covers t – to the list of the conditions in definition 6, the definition of $\lambda \equiv_s \lambda'$.

Case 2: Suppose z is the only element of $B_t^{(a,b)} \cap C_t^{(a,b)}$ in λ and that there are two elements $z_0 < z_1$ of $B_t^{(a,b)} \cap C_t^{(a,b)}$ in λ' . If a does not cover t , player I plays $x_0 = z_1$. If b does not cover t , player I plays $x_0 = z_0$. If t has quantifier-rank l , then the next $k - l - 2$ -many moves are spent isolating (a, b) , after which player I plays $x_{k-l-1} = z_0$ after which player I spends the remaining l -many moves defining $I_l(z_s, z_0)$, where $z_s = x_j$ or $z_s = y_j$, z_s is near the cut $(L_0, L_1) = a_s$, and since a_s did not cover t , the interval (z_s, z) has no realization of t .

Case 3: Suppose $B_t^{(a,b)}$ has a least element, $z \in \lambda^{(a,b)}$, but no least element in $\lambda'^{(a,b)}$ and a does not cover t . Player I plays $x_0 = z$, and then proves that (a, y_0) contains an element of type t , as in the previous case.

Case 4: Suppose $z \in (B_t^{(a,b)} \setminus B_u^{(a,b)})$ in λ but $B_t^{(a,b)} = B_u^{(a,b)}$ in λ' , and a does not cover u . Player I plays $x_0 = z$, and then proves that (a, y_0) contains an element of type u , as in case 2.

Case 5: If b doesn't cover u , player I plays $x_0 =$ an element of type t , $x_0 \in (B_t^{(a,b)} \cap C_u^{(a,b)})$ and shows $\exists z \in (x_0, b)$ of type u . If a doesn't cover t , player I plays $x_0 =$ an element of type u , $x_0 \in (B_t^{(a,b)} \cap C_u^{(a,b)})$ and shows $\exists z \in (a, x_0)$ of type t .

Case 6: Player I plays $x_0 = z$ and shows (x_0, b) contains no element of type u and (a, x_0) contains no element of type t .

Case 7: This holds iff $\neg \exists z (z \in \lambda^{(a,b)} \wedge t(z))$. By claim k , player I can play the element of type t in the model in which such an element exists, and then show that player II has replied outside $\lambda^{(a,b)}$. \square

Definition 7. For λ any linear order and k any number, let t be the function with domain $\{\min(B_t) : (A_t, B_t) \in I_k \text{ and } B_t \text{ has a minimal element}\} \cup \{\max(C_t) : (C_t, D_t) \in I_k \text{ and } C_t \text{ has a maximal element}\}$, taking the value $Th_{k-1}^{\text{loc}}(\lambda, x)$ on argument x ; let g be the function with domain I_k^+ taking the value $\{Th_{k-1}^{\text{loc}}(\lambda, x) : x \in \lambda^{(a,b)}\}$ on argument (a, b) . We call I_k, t, g total information for λ, k .

Lemma 2. If $x \in \lambda$, then from $Th_{k-1}^{\text{loc}}(\lambda, x)$ and total information for λ, k we can determine total information for $\{y \in \lambda : y < x\}, k-1$.

Proof: Every element of $I_{k-1}(\{y \in \lambda : y < x\})$ is an element of $I_k(\lambda)$ or an element of $\cap_{A,B} I_{k-1}(A+E+B+\{y \in \lambda : y < x\})$. The value of t on $z \in I_{k-1}(\{y \in \lambda : y < x\}) \cap I_k(\lambda)$ is determined by finding $Th_{k-2}^{\text{loc}}(\lambda, z)$ from $Th_{k-1}^{\text{loc}}(\lambda, z)$; the value of t on $z \in I_{k-1}(\{y \in \lambda : y < x\}) \cap \cap_{A,B} I_{k-1}(A+E+B+\{y \in \lambda : y < x\})$ is the value which the intersection of total information for any large linear order which ends in a copy of $\{y \in \lambda : y < x\}$, i.e. $\cap_{A,B} I_{k-1}(A+E+B+\{y \in \lambda : y < x\})$ assigns to z . We can also say that this is the value which the local theory of x in λ assigns to z . If a, b is an adjacent pair of elements of $I_{k-1}(\{y \in \lambda : y < x\})$, and both $a = (L_0, L_1)$ and $b = (R_0, R_1)$, are in $I_{k-1}(\{y \in \lambda : y < x\}) \cap I_k(\lambda)$, then the value $g((a, b))$ is determined by $I_{(k-2)}(\lambda)$, which names the first and last occurrences of each $k-2$ -quantifier type between a and b ; the $k-2$ -quantifier types realized in (a, b) are, then, those that $I_{(k-2)}(\lambda)$ finds to exist in (a, b) . If both $a = (L_0, L_1)$ and $b = (R_0, R_1)$ are in $I_{k-1}(\{y \in \lambda : y < x\}) \cap I_{k-1}(\{y \in \lambda : y < x\}) \cap I_k(\lambda)$, then the value of g is given by $Th_{k-1}^{\text{loc}}(\lambda, x)$. The value of $g(a, b)$ for b an element of $\cap_{A,B} I_{k-1}(A+E+B+\{y \in \lambda : y < x\})$, is determined by the location of x in $I_k(\lambda)$. \square

For instance, suppose $I_{(k-2)}(\lambda)$ lists the appearance of quantifier-rank- $k-2$ local types $t_0 \dots t_n$ left of x , and then lists quantifier-rank- $k-3$ local types between these, and x appears between the first appearances of two quantifier-rank- $k-3$ local types, r_0 and r_1 , but that r_0 and r_1 both appear between t_{n-1} and t_n . Then, in the gap left of x and all points bound locally to x by lower-rank types,

there appear the quantifier-rank- $k - 2$ local types $t_0 \dots t_{n-1}$. t_n appears left of x , but only after the disappearance of local type r_1 , and not before then (since r_1 disappeared between the last appearances of t_{n-1} and t_n).

Theorem 1. *For any two linear orders, λ, λ' , and for any number k , Player II wins $EF_k(\lambda, \lambda')$ iff the following holds: For all sequences $s <^{\text{seq}} (k - 1)$, there exists an equivalence $\lambda \equiv_s \lambda'$ – bijections between $I_s(\lambda)$ and $I_s(\lambda')$ identifying the cuts $(A_t^{(a,b)}, B_t^{(a,b)})$ and $(C_t^{(a,b)}, D_t^{(a,b)})$ in both models, for all t and all adjacent pairs a, b of elements in $\cup_{r <^{\text{seq}} s} I_r$ such that whenever B_t has a least element or C_t a greatest element, t of that least element is the same in both models, and such that in every interval in I_k , g has the same value in both models.*

Proof (only if): By claim k in lemma 1, player II must answer player I's first move x_0 with some y_0 in the same interval in I_k . But then, Player II can go on to win only if $Th_{k-1}(\lambda, x_0) = Th_{k-1}(\lambda', y_0)$, so $Th_{k-1}^{\text{loc}}(\lambda, x_0) = Th_{k-1}^{\text{loc}}(\lambda', y_0)$.

Proof (if): If $k = 2$, then $Th_0(\lambda, x) = \{x = x\} = \{t^0\} = Th_0^{\text{loc}}(\lambda, x)$, so $I_2 = I_{(0)}$ describes the first and last elements of type $x = x$, and between them determines which subset of $Th_1^{\text{loc}}(\lambda, x) = \{x = x\} = \{A + E + B + \lambda + C + E + D \models \exists y(x > y) \wedge \exists y(y > x)\} = \{t^1\}$ is realized. The reader can check that this accurately describes the various \equiv_2 classes of linear orders. That the theorem holds for $k \geq 3$ is proved by induction on lemma 2. \square

Remark 1. The generalization of I_s to infinitary logic (where s is a descending sequence of ordinals) is clear; the lemma that \equiv_α implies \equiv_s goes through as well, except that I_s is no longer finite, and so we must edit the “covering conditions.” The restriction of the I_s to the class of ordinals reproduces known results about the theory of ordinals and the infinitary theory of ordinals. For instance, in a finite linear order, for each sequence s , $I_{s'}$ identifies the least and greatest element of the central gap in I_s ; the number of nontrivial sequences s of numbers $\leq k - 2$ is $2^{k-1} - 1$, so the k -quantifier theory of a finite linear orders identifies that many elements on each end of the linear order (If there are any elements of λ not named in that process, they are indistinguishable by formulas of quantifier rank k).

By induction, we build the set of local types $\{\exists xt(x) : t = Th_k^{\text{loc}}(\lambda, x) \text{ in some model } \lambda\}$ of quantifier-rank k out of structures I_s^L (where we only consider the cuts of the form $(A_t^{(a,b)}, B_t^{(a,b)})$, and not of the form $(C_t^{(a,b)}, D_t^{(a,b)})$), and in turn the structures I_s are built by identifying the first and last elements of each local type. Thus, defining I_s from local types and local types from simplified I_s , we construct the set of local types of the next quantifier rank. Consistency requires that whenever local types t and s are realized and require an element between them which satisfies ϕ , a local formula of lower quantifier rank, then there is some local type r which extends ϕ . We can prove that if these necessary and propositional formulas are satisfied by a set of local types of uniform quantifier rank, then there is a model realizing exactly those local types, via a Model Existence Game (proving that a model can be constructed element by element,

and that at no stage is there any obstruction), or by constructing the model at once and in its entirety. The resulting models involve many arbitrary choices about whether to realize s left of t , or t left of s . If we consistently decide to realize both, whenever possible, then the models which are constructed contain intervals which are determined by the total information, and intervals in which all possible types are realized densely.

The inherent complexity of deciding the theory of linear order consists only in the difficulty of extending a formula of quantifier rank k to a total information triple of quantifier rank k , because any total information triple is then easily extended to a complete piecewise-dense model in quantifier rank $k + 2$. By comparison, the proof in [1] is surely the most direct proof that the theory of linear order is atomic, but no reasonable bound on the quantifier rank of the completion can be obtained using that argument. In fact, [4] describes the quantifier rank of the completion obtained by that method, and finds an enormous upper bound, involving towers of exponents in the arguments (page 341, middle paragraph). Studying the propositional theory of sets of local types describes the complexity of the set $\{\text{total information triples for } \lambda, k : \lambda \text{ is a linear order and } k \text{ a natural number}\}$. In any case, simply because the theory is propositional, this set is recursively enumerable, thus the theory of linear order is decidable. We have come to that theorem by constructing large finite approximations to a linear order, and I would venture to say that these finite sets are what were described in [2] as the semimodels of a linear order. From our total information triples, we can construct semimodels for the linear orders – the proof of Lemma 1.1 indicates what elements of the model should be played as corresponding to the various elements of I_s .

2 Semimodels and the Complexity of Deciding $Th(LO)$

A semimodel for the vocabulary $\{<\}$ is, in general, a nested sequence of linear orders $M = (M_0 \subseteq M_2 \subseteq \dots M_{k-1}, <^{M_n})$ where we intend to evaluate formulas ϕ of quantifier rank $\leq k$. We write $\phi^{s.m.}$ for the result of replacing the subformula $\exists x\psi$, when it occurs in ϕ within the scope of l_ψ -many quantifiers, by the bounded quantification $\exists x \in M_l\psi$. More precisely,

Definition 8. *A semimodel is a nested sequence $M = M_0 \subseteq M_1 \subseteq \dots M_{k-1}$ of finite sets, with an ordering on M_{k-1} . The rank of M is k .*

We say $M \models^{s.m.} \phi$ if $M \models \phi^{s.m.}$, where $\phi^{s.m.}$ is the relativization of ϕ in which any subformula $\exists x_j\psi(x_0 \dots x_j)$ which appears within the scope of j -many quantifiers is replaced by $\exists x_j \in M_j\psi(x_0 \dots x_j)$, for all $j < k$.

Theorem 2 (Ehrenfeucht [2]). *If U is a class of finite semimodels, and the following hold:*

1. *U is recursively enumerable,*
2. *For any formula ϕ and any semimodel $L \in U$ such that the rank of L as a semimodel is the rank of ϕ as a formula, if $L \models^{s.m.} \phi$, then L extends to a full linear order λ , such that $\lambda \models \phi^{s.m.}$,*

3. For any formula ϕ and any linear order λ such that $\lambda \models \phi$, $\phi^{\text{s.m.}}$ holds in some semimodel $L \in U$,

then the theory of linear order is decidable.

Proof: Enumerate the implications of the theory of linear order, and look for $\neg\phi$. Meanwhile, enumerate elements of U , and look for $L \in U$ of rank equal to the quantifier rank of ϕ , such that $L \models \phi$. By condition 2., some linear order models ϕ , too. By condition 3, if ϕ is consistent, then this procedure terminates in the discovery of a semimodel of ϕ . \square

Definition 9. If M is a semimodel of rank at least k , and λ is a linear order, we say $M \equiv_k^{\text{s.m.}} \lambda$ if for any sentence ϕ of quantifier rank at most k , $\lambda \models \phi$ just in case $M \models^{\text{s.m.}} \phi$.

We write a semimodel of rank k as a string $s_i : i < n$ of numbers $< k$, from which $M_j = \{i : s_i \leq j\}$ recovers the semimodel.

Given a semimodel M , let M^+ be the semimodel formed by shifting the sets M_i ; i.e., $(M^+)_{i+1} = M_i$; $(M^+)_0 = \emptyset$.

Addition of semimodels is the same as addition of orders; if M and N are disjoint and of the same rank, form $(M_{k-1}, <) + (N_{k-1}, <)$, and let $(M + N)_i = M_i \cup N_i$ for $i < k$.

Definition 10. We call M consistent if when $M \models^{\text{s.m.}} \phi$ then from ϕ is not inconsistent with the theory of linear order.

Note that when we determine $M \models^{\text{s.m.}} \phi$, we restrict the quantifiers of ϕ to the ranked sets in M , and when we determine whether ϕ is consistent, we do not restrict its quantifiers.

For instance, 01 is not a consistent semimodel, since $01 \models^{\text{s.m.}} LO \wedge (\forall x \exists y (y > x)) \wedge (\forall x \neg \exists y (y < x))$, where LO is the theory of linear order. Of course, that is not consistent.

Theorem 3. For any model M and sentence ϕ satisfied in M , there is a semimodel N such that $N \models^{\text{s.m.}} \phi$.

Proof: To determine that $M \models \phi$, it is never necessary to play more than finitely many elements of M and M^+ . To be precise about which elements are played was the work of the previous section, where we found that total information is a finite set, sufficient for determining the k -quantifier theory of M . Turning a total information triple into a semi-model can be done automatically: The elements of M which appear there should be simply added to the semi-model; elements of M^+ which appear in the total information triple represent families of types which are realized cofinally often; if that family of types is codified in the semimodel M , then M^+MM is a semimodel in which each of the types in the family is realized left of any other. \square

If that sketch of how to write a string from total information triple seems unsatisfying, I suspect that the problem is mostly that total information triples are

complicated objects, which obviously are not necessary in so simple a theorem. Below, we will re-prove this theorem, by constructing semi-models for the linear orders used in [3].

As a result of this theorem, deciding the theory of linear order can be done by deciding which strings are consistent semimodels for the theory of linear order.

Theorem 4. $(0)^{2^{k-1}}(1)^{2^{k-2}}(2)^{2^{k-3}} \dots (k-1)^{2^0} \models^{\text{s.m.}} Th_k(\omega)$.

Proof: We play the EF game between a semi-model and a model, where in the semi-model the l -th move is restricted to M_l . We answer a large natural number with the last 0. To the right, this leaves the statement of the theorem for $k-1$, so we appeal to induction. To the left, this leaves $0^{2^{k-1}-1}$, which is equivalent to any large, finite linear order, as the reader may check, again by induction. \square

We can also see this proof in terms of total information: For each descending sequence s of numbers $< k-1$, I_s contains an element right of those definable in I_r , where r is the predecessor of s . Any of these could be played on the first move. This accounts for $2^{k-1} - 1$ many elements on the left end of both ω and the semi-model. The next element of ω , corresponding to the last element of M_0 in the semi-model, is an element not definably close to the left end. Its $k-1$ -quantifier type is that it is preceded by a large finite linear order of unknown size, and followed by something $equiv^{k-1}\omega$.

Theorem 5. If $E \equiv_k^{\text{s.m.}} \eta$, the dense linear order without endpoints, then $E^+ + (0) + E^+ \equiv_{k+1}^{\text{s.m.}} \eta$. $101 \equiv_2^{\text{s.m.}} \eta$; $2120212 \equiv_3^{\text{s.m.}} \eta$.

Proof: Like E , η has only one type (of quantifier rank k) $t(x)$, and after playing x to any element of η , the k -quantifier theory of $\eta^{<x}$ or $\eta^{>x}$ is the k -quantifier theory of η . Base case: $\emptyset \equiv_0^{\text{s.m.}} \eta$. \square

Definition 11. In the game $EF_k^{\text{s.m.}}(L, \lambda)$ played on a semi-model and a model, player I plays in L or λ , and player II replies in the other structure. On the $j+1$ -th move, the player who assigns x_j to an element of L must assign x_j to an element of L_j .

Given two semi-models M and N , call $M \times N$ the semi-model obtained by replacing every element $x \in N$ such that $x \in N_j \setminus N_{j-1}$ by a copy of $M^{(+)^j}$.

Note that player II has a winning strategy in $EF_k^{\text{s.m.}}(L, \lambda)$ just in case $L \equiv_k^{\text{s.m.}} \lambda$.

Theorem 6. If $M \equiv_k^{\text{s.m.}} \mu$ and $L \equiv_k^{\text{s.m.}} \lambda$, then $M \times L \equiv_k^{\text{s.m.}} \mu \times \lambda$.

Proof: Suppose player II has a winning strategy in $EF_k^{\text{s.m.}}(L, \lambda)$ and in $EF_k^{\text{s.m.}}(M, \mu)$. On the j -th move of $EF_k^{\text{s.m.}}(M \times L, \mu \times \lambda)$, if player I has moved in the same element of λ or L as in the previous move, player II follows the winning strategy of $EF_k^{\text{s.m.}}(M, \mu)$ in that copy of M . If player I plays in a new element of λ or L , player II responds as in $EF_k^{\text{s.m.}}(L, \lambda)$, and in the corresponding copy of M and μ , begins a new game of $EF_k^{\text{s.m.}}(M, \mu)$. If player I plays in $\lambda \times \mu$, then player II has answered with an element of $(M \times L)_i$. \square

Definition 12. ([3]) M_{LL} is the smallest set of linear orders containing the singleton order, 1 and closed under the operations

- $+$, where $M + N$ forms disjoint copies of M and N , and then orders all the elements of M before any element of N .
- $\times \omega$, where $M \times \omega$ contains countably many copies of M , indexed by ω , and if $n < m \in \omega$, all the elements of M_n occur before any element of M_m .
- $\times \omega^*$, which has the same domain as $M \times \omega$, but if $n < m$, then in $M \times \omega^*$ all elements of M_m precede any element of M_n .
- σ : Let η be the ordering on the rationals, and let χ be any function from η to n such that $\chi^{-1}(i)$ is dense for each $i < n$. The domain of $\sigma\{M_i : i < n\}$ contains a copy M_q of M_i for each rational number q such that $\chi(q) = i$ – and M_q and M_p are disjoint if $p \neq q$ – and if $p < q$ then every element of M_p precedes any element of M_q .

If we understand that M^* is the ordering with the domain of M in which $a <^{M^*} b$ if $a >^M b$, then we could define $M \times \omega^* = (M^* \times \omega)^*$.

Definition 13. Let En be the semimodel such that $En_j = (En_{j-1}^+ + (0)) \times n + En_{j-1}^+$. Let χ map En to n , sending the i -th occurrence of 0 in the construction of En_j to i , for all j . Let $\sigma(\{M_i : i < n\})$ be the semi-model obtained from En by replacing every element $x \in En$ such that $\chi(x) = i$ and $x \in N_j \setminus N_{j-1}$ by a copy of $M_i^{(+)^j}$.

Theorem 7. $En_k \equiv_k^{s.m.} \eta$. $\chi^{-1}(i) \equiv_k^{s.m.} \eta$. Let En, χ , and $\sigma(\{M_i : i < n\})$ be as in the preceding definition. If $M_i \equiv_k^{s.m.} \mu_i$, then the shuffle of the semi-models is $\equiv_k^{s.m.}$ to the shuffle of the linear orders: $\sigma(\{M_i : i < n\}) \equiv_k^{s.m.} \sigma(\{\mu_i : i < n\})$.

Proof: As in the proof of the product, Player II follows $E, \chi \equiv_k^{s.m.} \eta, \chi$, and begins a new game of $EF_k^{s.m.}(M_i, \mu_i)$ whenever whenever player I changes from one element of E or η to another. Player II follows $EF_k^{s.m.}(M_i, \mu_i)$, whenever player I keeps the element of E or η constant. \square

We can formulate semi-models of rank k for elements of M_{LL} as follows:

- $1 \equiv_k^{s.m.} 1$,
- $M + L \equiv_k^{s.m.} \mu + \lambda$ if $M \equiv_k^{s.m.} \mu$ and $L \equiv_k^{s.m.} \lambda$,
- $M \times L \equiv_k^{s.m.} \mu \times \omega$ if $M \equiv_k^{s.m.} \mu$ and $L \equiv_k^{s.m.} \omega$,
- $M \times L \equiv_k^{s.m.} \mu \times \omega^*$ if $M \equiv_k^{s.m.} \mu$ and $L \equiv_k^{s.m.} \omega^*$,
- $\sigma(\{M_i : i < n\}) \equiv_k^{s.m.} \sigma(\{\mu_i : i < n\})$ if $M_i \equiv_k^{s.m.} \mu_i$ for each $i < n$.

Now, the set of semi-models M such that $M \equiv_k \mu$ for some $\mu \in M_{LL}$ is clearly recursively enumerable, and clearly each such M extends to a linear order (namely, μ), so conditions 1 and 2 in Ehrenfeucht’s theorem are clear. The final, third, condition is what was proved in [3].

What is the complexity of determining whether a string is consistent? We can show that this question can be determined quickly in the length of a string, i.e., in linear time. However, the length of a string which represents, as a semi-model, a random k -quantifier equivalence class, has length tower-of-exponential in k .

References

1. Amit, R., Shelah, S.: The complete finitely axiomatized theories of order are dense. *Israel J. Math.* 23, 200–208 (1976)
2. Ehrenfeucht, A.: Decidability of the theory of linear ordering relation. *AMS Notices*, 6.3.38, 550–538 (1959)
3. Läuchli, H., Leonard, J.: On the elementary theory of linear order. *Fund. Math.* LIX, 109–116 (1966)
4. Rosenstein, J.: *Linear Orderings*, pp. 341–342. Academic Press, London (1982)

Hybrid Logical Analyses of the Ambient Calculus

Thomas Bolander¹ and René Rydhof Hansen²

¹ Informatics and Mathematical Modelling, Technical University of Denmark
tb@imm.dtu.dk

² Department of Computer Science, University of Copenhagen
rrhansen@diku.dk

Abstract. In this paper, hybrid logic is used to formulate a rational reconstruction of a previously published control flow analysis for the mobile ambients calculus and we further show how a more precise *flow-sensitive* analysis, that takes the ordering of action sequences into account, can be formulated in a natural way. We show that hybrid logic is very well suited to express the semantic structure of the ambient calculus and how features of hybrid logic can be exploited to reduce the “administrative overhead” of the analysis specification and thus simplify it. Finally, we use *HyLoTab*, a fully automated theorem prover for hybrid logic, both as a convenient platform for a prototype implementation as well as to formally prove the correctness of the analysis.

Keywords: Hybrid logic, static analysis, mobile ambients.

1 Introduction

With the increased, and increasing, focus on making secure and reliable systems also comes an increased need for advanced tools and techniques that can verify important security and reliability properties of a system in all phases of design, modelling, and implementation. In recent years various forms of *static analysis* has gained popularity as basis for verification tools, in part because static analysis can be fully automated and in part because of the ability of static analysis to scale to even the largest systems. There has also been a growing interest in applying static analysis techniques to the verification of more or less formal system models, e.g., models formulated as UML diagrams or process calculi, to verify relevant safety and security properties before starting the actual implementation.

The main goal of this paper is to show how *hybrid logics* can be applied to the specification, validation, and implementation of static analyses. We do this by developing three increasingly precise analyses of the *mobile ambients* process calculus [6] which is used to model distributed systems with a high degree of mobility. The analyses are specified using hybrid logics, in a way that is inspired by the Flow Logic approach [10]; the correctness of the analyses are proved using the *HyLoTab* theorem prover for hybrid logics [11]; and by exploiting the model

generation capabilities of *HyLoTab* we also automatically obtain implementations of the analyses directly from their respective specifications. In addition to the obvious convenience of getting an implementation for free, it also obviates the need for proving the correctness of the implementation.

Furthermore, we argue that hybrid logics are particularly well-suited for specifying analyses for the mobile ambients calculus because the structure of models for hybrid logic formulae is very similar to the structures underlying the semantics of mobile ambients. In addition, the nominals, intrinsic to hybrid logics, provides an easy way to handle the analysis of bound names in the ambient calculus and thereby avoid the additional technical complexity otherwise necessary, cf. [7,8].

Finally, while the intention of this paper is to show some of the advantages of using hybrid logics for static analysis, the developed analyses are not mere “toy”-analyses only useful for demonstration purposes. Indeed the analysis described in Section 6 is comparable to the OCFA analyses of [7,8] and the analysis defined in Section 7 incorporates an element of *flow sensitivity*, by taking the ordering of action sequences into account, that markedly and non-trivially improves precision of the analysis. Although this flow-sensitive analysis is unlikely to be as precise as the “counting analysis” of [7] it does have the advantage of retaining a simple specification, that is straightforward to prove correct, as well as being easily implementable.

Related Work

The mobile ambients calculus has been studied in a number of contexts and several type systems and static analyses have been developed for it, see for example [4,5,7,8,2,1,9]. The analyses in this paper are mainly inspired by the control flow analyses developed in the Flow Logic framework [7,8]. However, in none of these works are logics used so pervasively and essentially to obtain clear analysis specifications, computer assisted correctness proofs, and automated implementation. We are also not aware of any prior use of hybrid logics for static analysis.

2 Mobile Ambients

Process algebras have long been used for modelling concurrent and distributed systems. These models have been invaluable in studying and solving some of the many problems inherent in such systems such as deadlocking, synchronisation, fairness etc.

The *ambient calculus* is a process calculus specifically designed to model *mobility* of processes. This is in contrast to traditional process calculi, such as CSP and the π -calculus, that focus on *communication* between processes. In particular it is possible for an active process and its environment to move between sites. The entities that move in the calculus are called ambients and they may contain other active processes as well as other ambients. This gives rise to a tree structure that changes dynamically as ambients move.

$P, Q ::= (\nu n)P$	restriction
$\mathbf{0}$	inactivity
$P \mid Q$	composition
$!P$	replication
$n[P]$	ambient
$\mathbf{in} \ n.P$	capability to enter n
$\mathbf{out} \ n.P$	capability to exit n
$\mathbf{open} \ n.P$	capability to open n

Fig. 1. Ambient Syntax

It is exactly this tree structure the static analyses discussed in later sections are designed to approximate. The primary goal of the analyses is to ensure that *all possible* concrete tree structures that may occur during the execution of a program are represented in the analysis result. Since this is undecidable in general, the analyses have to *over-approximate* the set of actual tree structures and thus an analysis result may contain tree structures that will never actually occur in a program execution.

In this paper we focus on the core calculus and thus do not take communication into account. As shown in [6] the core calculus is Turing-complete.

Syntax

We assume the existence of a countably infinite set **Nam** of *names*. The metavariables k, l, m, n , and so on, range over names. The syntax of the mobile ambient calculus is defined in Figure 1. The restriction operator $(\nu n)P$ creates a new name n with scope P ; the inactive process, i.e., a process that does nothing, is denoted by $\mathbf{0}$; processes P and Q running in parallel is represented by $P \mid Q$; replication, $!P$, is equivalent to an unbounded number of copies of P running in parallel, thereby providing a recursive operator.

By $n[P]$ we denote the ambient named n that has the process P running inside it. The capabilities **in** n and **out** n are used to move their enclosing ambients whereas **open** n is used to dissolve the boundary of a sibling ambient; this will be made precise when we define the semantics below. We write $\text{fn}(P)$ for the free names of P . Trailing occurrences of the inactive process $\mathbf{0}$ will often be omitted.

Semantics

The semantics of the mobile ambients calculus is defined as a straightforward *reduction semantics* (see Figure 3) using a *congruence relation* (see Figure 2) as is common for process calculi.

The essential idea is that the ambient hierarchy, i.e., the structure determining which ambients are inside which other ambients, can be changed dynamically by ambients entering or exiting other ambients or even by dissolving another ambient. These notions are formalised as the (**in**), (**out**), and (**open**) reduction shown in Figure 3. In later sections these rules will be visualised and explained

$$\begin{array}{ll}
P \equiv P & P \mid Q \equiv Q \mid P \\
P \equiv Q \Rightarrow Q \equiv P & (P \mid Q) \mid R \equiv P \mid (Q \mid R) \\
P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R & !P \equiv P \mid !P \\
\\
P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q & P \equiv Q \Rightarrow \text{in } n.P \equiv \text{in } n.Q \\
P \equiv Q \Rightarrow P \mid R \equiv Q \mid R & P \equiv Q \Rightarrow \text{out } n.P \equiv \text{out } n.Q \\
P \equiv Q \Rightarrow !P \equiv !Q & P \equiv Q \Rightarrow \text{open } n.P \equiv \text{open } n.Q \\
P \equiv Q \Rightarrow n[P] \equiv n[Q] & \\
\\
P \mid \mathbf{0} \equiv P & (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \\
(\nu n)\mathbf{0} \equiv \mathbf{0} & (\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \\
!\mathbf{0} \equiv \mathbf{0} & \text{if } n \notin \text{fn}(P) \\
& (\nu n)(m[P]) \equiv m[(\nu n)P] \\
& \text{if } n \neq m
\end{array}$$

Fig. 2. Structural congruence

$$\begin{array}{ll}
n[\text{in } m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R] & (\text{in}) \\
m[n[\text{out } m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R] & (\text{out}) \\
\text{open } n.P \mid n[Q] \rightarrow P \mid Q & (\text{open}) \\
\\
\frac{P \rightarrow Q}{n[P] \rightarrow n[Q]} \quad (\text{amb}) & \frac{P \rightarrow Q}{(\nu n)P \rightarrow (\nu n)Q} \quad (\nu) \\
\\
\frac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R} \quad (|) & \frac{P' \equiv P \quad P \rightarrow Q \quad Q \equiv Q'}{P' \rightarrow Q'} \quad (\equiv)
\end{array}$$

Fig. 3. Reduction relation

in more detail. The **(amb)** rule allows processes to be active even when moved around in the ambient hierarchy. The (ν) rule defines a scope for the name bound at the ν in much the same way that λ acts as a name binder in the λ -calculus. In the ambient calculus α -equivalent processes, i.e., processes that are identical modulo α -renaming of bound names, are considered to be *identical*. This equivalence often gives rise to technical difficulties in the design of static analyses and heavy technical machinery, like annotating the syntax of processes with labels or markers that remain stable under α -conversion, frequently has to be employed to overcome the inherent difficulties in handling α -equivalence. In a later section we show how such difficulties can be completely avoided by encoding ambient names as nominals in hybrid logic and thereby significantly simplify the technical developments needed for an analysis of the ambient calculus. Processes can be composed in parallel and executed in an interleaved way, as shown by the $(|)$ -rule. Finally, the (\equiv) -rule integrates the congruence relation defined in Figure 2 into the reduction rules. We will not go into further detail with the semantics here but refer the reader to later sections and [6].

Analysing Ambients

As mentioned earlier, the essential and most basic property to analyse in the ambient calculus, is how the ambient hierarchy, i.e., the tree structure determined by which ambients are contained inside which other ambients, may dynamically develop during execution. The analyses described in this paper all compute over-approximations of this dynamically evolving structure. Such analyses are comparable to computing control flow graphs for imperative languages or performing control flow analysis of a functional language. Indeed, in keeping with [7,8], we call such analyses of the ambient calculus *control flow analyses*. In [7,8] the Flow Logic framework is used to define a number of control flow analyses for the ambient calculus. The Flow Logic framework [10] is a specification-oriented framework for defining static analyses where specification of the analysis is clearly separated from the computation of the analysis result.

In the Flow Logic framework analyses are defined by specifying what it means for a proposed analysis result to be correct (rather than how to compute it). In this way an analysis designer can focus on high-level aspects such as what properties to analyse and what correctness means without having to take low-level implementation issues into account. This approach gives a framework that is light-weight and well-suited for rapidly developing a large variety of analyses. The analyses described in later sections are developed in a way that is inspired by the Flow Logic framework.

An analysis specified using Flow Logic is typically implemented by systematically transforming the analysis specification into a constraint generator and then use an appropriate constraint solver to perform the actual (fixed-point) computation of the analysis result. It is often a trivial but tedious task to prove that the constraint generator correctly implements the analysis specification. In this paper we show how the model-generating capabilities of the *HyLoTab* theorem prover for hybrid logics enables us to automatically obtain an implementation for performing the necessary fixed-point computations directly from the high-level analysis specification and thereby obviating the need for proving correctness of the implementation: it is correct by construction.

3 Hybrid Logic

Hybrid logic is an extension of propositional modal logic. Both modal and hybrid logic are thoroughly introduced in [3]. Hybrid logic is obtained from propositional modal logic by adding a second sort of propositional symbols, called *nominals*. We assume that a countably infinite set **Nom** of nominals is given. The metavariables a, b, c , and so on, range over nominals. The semantic difference between ordinary propositional symbols and nominals is that nominals are required to be true at *exactly one* world; that is, a nominal “points to a unique world”. In addition to the nominals we have a countable set of ordinary *propositional symbols*, **Prop**. We use the metavariables p, q, r , and so on, to range over propositional symbols. We assume that the sets **Nom** and **Prop** are disjoint. The hybrid logic

we consider has the following syntax:

$$\phi ::= \top \mid p \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \Diamond\phi \mid \Diamond^-\phi \mid @_a\phi \mid \downarrow a.\phi$$

where $p \in \mathbf{Prop}$ and $a \in \mathbf{Nom}$. This hybrid logic is usually denoted $HL(@, -, \downarrow)$. The dual modal operators \Box and \Box^- , and the propositional connectives not taken as primitive are defined as usual. We now define models.

Definition 1. A model is a tuple (W, R, V) where

1. W is a non-empty set, whose elements are usually called worlds.
2. R is a binary relation on W called the accessibility relation of the model. If $(w, v) \in R$ we say that v is accessible from w .
3. For each propositional symbol p , $V(p)$ is a subset of W . For each nominal a , $V(a)$ is an element of W . $V(a)$ is called the world denoted by a .

Given a model $\mathcal{M} = (W, R, V)$, a world $w \in W$, and a nominal a , we will use \mathcal{M}_w^a to refer to the model which is identical to \mathcal{M} except V maps a to w .

The relation $\mathcal{M}, w \models \phi$ is defined inductively, where $\mathcal{M} = (W, R, V)$ is a model, w is an element of W , and ϕ is a formula of $HL(@, -, \downarrow)$.

$$\begin{aligned} \mathcal{M}, w &\models \top \text{ iff true} \\ \mathcal{M}, w &\models p \text{ iff } w \in V(p) \\ \mathcal{M}, w &\models a \text{ iff } V(a) = w \\ \mathcal{M}, w &\models \neg\phi \text{ iff not } \mathcal{M}, w \models \phi \\ \mathcal{M}, w &\models \phi \wedge \psi \text{ iff } \mathcal{M}, w \models \phi \text{ and } \mathcal{M}, w \models \psi \\ \mathcal{M}, w &\models \Diamond\phi \text{ iff for some } v \in W, (w, v) \in R \text{ and } \mathcal{M}, v \models \phi \\ \mathcal{M}, w &\models \Diamond^-\phi \text{ iff for some } v \in W, (v, w) \in R \text{ and } \mathcal{M}, v \models \phi \\ \mathcal{M}, w &\models @_a\phi \text{ iff } \mathcal{M}, V(a) \models \phi \\ \mathcal{M}, w &\models \downarrow a.\phi \text{ iff } \mathcal{M}_w^a, w \models \phi. \end{aligned}$$

By convention $\mathcal{M} \models \phi$ means $\mathcal{M}, w \models \phi$ for every world $w \in W$. A formula ϕ is *valid* if and only if $\mathcal{M} \models \phi$ for any model \mathcal{M} . In this case we simply write $\models \phi$. A formula ϕ is *satisfiable* if and only if $\neg\phi$ is not valid, that is, if and only if there exists a model \mathcal{M} and a world w such that $\mathcal{M}, w \models \phi$. A formula ϕ is *satisfiable in a model* $\mathcal{M} = (W, R, V)$ if and only if there is a world $w \in W$ such that $\mathcal{M}, w \models \phi$. When ϕ is satisfiable in a model \mathcal{M} we also say that \mathcal{M} *satisfies* ϕ .

If ϕ and ψ are formulae and χ is a subformula of ϕ , we use $\phi[\psi/\chi]$ to denote the formula obtained from ϕ by replacing all occurrences of χ by ψ . Later we will need the following well-known and basic result which we state without proof.

Lemma 1. Let ϕ and ψ be formulae of hybrid logic, and let p be propositional symbol occurring in ϕ . If $\models \phi$ then $\models \phi[\psi/p]$.

In later sections nominals are used to represent the names of the ambient calculus. To simplify matters we take \mathbf{Nam} to be a subset of \mathbf{Nom} thus allowing ambient names to be represented directly as nominals in a hybrid logic formula. This further has the great advantage that α -renaming in the ambient calculus can be directly modelled by α -renaming in hybrid logic, and thereby dispense with the need for more technically involved solutions.

4 Modelling Ambients in Hybrid Logic

Our inspiration for modelling ambients using hybrid logic comes from the way expressions in the ambient calculus are usually visualised. To make things simple, let us start by considering only the following fragment of the ambient calculus:

$$P, Q ::= \mathbf{0} \mid P|Q \mid n[P]. \quad (1)$$

Consider the process P_s in this fragment given by $P_s = n[v[0] \mid k[u[0]]] \mid m[0]$. The standard visualisation of P_s is given in Figure 4.

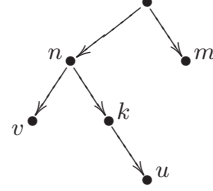
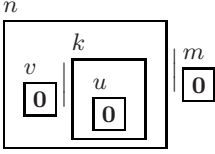


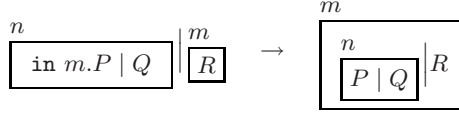
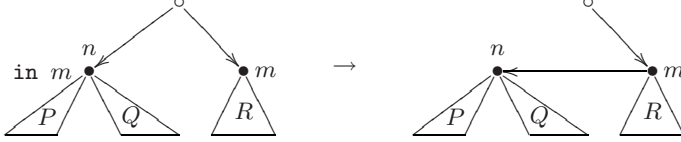
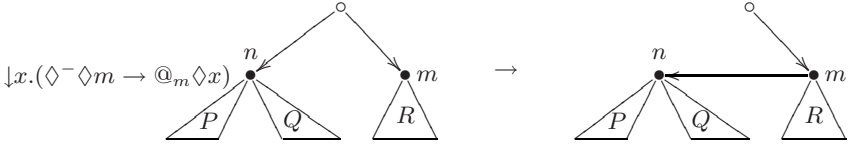
Fig. 4. Box visualisation of the process P_s **Fig. 5.** Tree visualisation of the process P_s

The idea is here to visualise each ambient expression $n[P]$ as a box, using the name n of the ambient as a label on the box, and P as the content of the box. If the process P itself contains ambients, these will be visualised as boxes nested inside the box labelled n . This gives the system of boxes nested within each other a tree-like structure, and it seems obvious to try to visualise this structure as a tree instead of a system of boxes. Ambient names can then be used to label the nodes of the tree, so the process P_s would become represented by the tree given in Figure 5. Such a tree can be interpreted as a hybrid logical model with the ambient names interpreted as nominals. In fact, we can even describe the tree of Figure 5 directly as a hybrid logical formula, $\phi_s = \Diamond(n \wedge \Diamond v \wedge \Diamond(k \wedge \Diamond u)) \wedge \Diamond m$. The formula ϕ_s represents the tree in the sense that any hybrid logical model \mathcal{M} in which ϕ_s is satisfiable will contain the tree of Figure 5 as a subtree (unless some of the nominals are mapped into identical worlds, but if needed this possibility can be excluded by replacing ϕ_s by $\phi_s \wedge \bigwedge_{x \neq y} \neg @xy$). Thus the formula ϕ_s can be considered as a syntactic representation of the tree in Figure 5, which in turn represents the process P_s . Indeed the original process and the corresponding hybrid formula are quite similar:

$$\begin{aligned} P_s &= n[\quad v[0] \mid k[u[0]] \quad] \mid m[0]. \\ \phi_s &= \Diamond(n \wedge \Diamond v \wedge \Diamond(k \wedge \Diamond u)) \wedge \Diamond m. \end{aligned}$$

Inspired by this similarity we can define a direct translation from processes of the fragment (1) into hybrid logical formulae. We define such a translation $(\cdot)^*$ recursively by:

$$\begin{aligned} \mathbf{0}^* &= \top \\ (P \mid Q)^* &= P^* \wedge Q^* \\ (n[P])^* &= \Diamond(n \wedge P^*). \end{aligned}$$

**Fig. 6.** Box visualisation of the (in) axiom**Fig. 7.** Tree visualisation of the (in) axiom**Fig. 8.** Hybrid logical visualisation of the (in) axiom

It is easy to check that P_s^* is logically equivalent to ϕ_s . The point is now that if we take a process P , translate it into the hybrid logical formula P^* , and calculate a model \mathcal{M} of P^* then \mathcal{M} will actually be an analysis of P . This is because the accessibility relation of \mathcal{M} will be encoding the containment relation between the ambients appearing in Q .

Consider now the following extended fragment of the ambient calculus including the three capabilities **in**, **out** and **open**:

$$P, Q ::= \mathbf{0} \mid P|Q \mid n[P] \mid \mathbf{in} \, n.P \mid \mathbf{out} \, n.P \mid \mathbf{open} \, n.P.$$

Capabilities are expressing actions that can be performed on the surrounding ambients. Consider the axiom (in) of Figure 3. A standard visualisation of this axiom is given in Figure 6. What happens in the reduction step of the axiom is that we “execute” the capability **in** $m.P$ which tells the surrounding ambient n to move inside the sibling ambient named m . Using trees instead of boxes, a simple representation of the (in) axiom could be as in Figure 7. In the figure the root is marked by \circ . We will use this as a general convention in the following. From the figure we see that the capability essentially removes one edge and adds another. Since our analyses of the ambient calculus are going to be *over-approximations*, we will concentrate on edges that are added and ignore edges that are removed. Ignoring the removed edge, the capability ‘in m ’ can be seen

as expressing “if m is my sibling, then add an edge from m to me” (compare Figure 7). In hybrid logic this translates into: $\downarrow x.(\Diamond^- \Diamond m \rightarrow @_m \Diamond x)$. The hybrid logical formula expresses: “At the current node x , if it is possible to go one step backwards and then one step forwards to find the node m then there is an edge from m to x ”. Or simply: “If m is a sibling of the current node x then there is an edge from m to x ”. This suggests extending the translation $(\cdot)^*$ defined above to translate **in**-capabilities in the following way:

$$(\text{in } n.P)^* = P^* \wedge \downarrow x.(\Diamond^- \Diamond n \rightarrow @_n \Diamond x).$$

Graphically we can then replace the capability ‘**in** m ’ in Figure 7 by the hybrid logical formula $\downarrow x.(\Diamond^- \Diamond m \rightarrow @_m \Diamond x)$, as is done in Figure 8. The point is now that if we have a hybrid logical model containing the tree on the left hand side of the reduction arrow in Figure 8, then it will also contain the tree on the right hand side. This is because the translated capability $\downarrow x.(\Diamond^- \Diamond m \rightarrow @_m \Diamond x)$ at n forces there to be an edge from m to n . We can make translations of the **out** and **open** capabilities that behave in a similar way. In the end we obtain a translation $(\cdot)^*$ satisfying the following property:

Let P and Q be processes. If a hybrid logical model \mathcal{M} satisfies P^* and if P can be reduced to Q , then \mathcal{M} also satisfies Q^* .

This is a *subject reduction* result, which is a central property of any analysis based in Flow Logic. As we shall see in the following section, it follows directly from this subject reduction result that the translation $(\cdot)^*$ gives rise to a correct analysis of the calculus.

Note that the above translation of ambient calculus processes into hybrid logic formulae merges all occurrences of ambients with the same name, i.e., the translation is unable to distinguish between two different syntactic occurrences of the same name. This results in a slight imprecision (when analysing processes with non-bound names) in the analyses described in later sections. However, this does not affect the correctness of the analyses and it is easily remedied in the rare cases where the extra precision is needed, e.g., by adding more structure to ambient names or even by annotating ambients with labels, cf. [7,8].

5 The Naive Analysis

We now develop the ideas introduced above in detail. Our first analysis is rather simple, and we thus call it the ‘the naive analysis’. The naive analysis is based on a simple operator $(\cdot)_0$ taking a process P and returning a hybrid logical formula P_0 . The operator is defined inductively by the following clauses.

$$\begin{aligned} ((\nu n)P)_0 &= P_0 \\ \mathbf{0}_0 &= \top \\ (P \mid Q)_0 &= P_0 \wedge Q_0 \\ (!P)_0 &= P_0 \\ (n[P])_0 &= \Diamond(n \wedge P_0) \\ (\text{in } n.P)_0 &= P_0 \wedge \downarrow x.(\Diamond^- \Diamond n \rightarrow @_n \Diamond x) \end{aligned}$$

$$\begin{aligned}
(\text{out } n.P)_0 &= P_0 \wedge \downarrow x.(\Diamond^- n \rightarrow @_n \Box^- \Diamond x) \\
(\text{open } n.P)_0 &= P_0 \wedge (\Diamond n \rightarrow n).
\end{aligned}$$

For the translation operator $(\cdot)_0$ we have the following results.

Lemma 2. *For all processes P and Q , if $P \equiv Q$ then $\models P_0 \leftrightarrow Q_0$.*

Proof (Sketch). We need to show that for each of the rules of Figure 2 we get a true statement if we replace each congruence $P \equiv Q$ appearing in the rule by $\models P_0 \leftrightarrow Q_0$. It is easy to check that this is indeed the case using the basic properties of the logical connectives \leftrightarrow and \wedge .

Theorem 1 (Subject Reduction for $(\cdot)_0$). *For all processes P and Q , if $P \rightarrow Q$ then $\models P_0 \rightarrow Q_0$.*

Proof. The proof proceeds by induction on the depth of the proof tree for $P \rightarrow Q$. In the base case we consider proof trees of depth 0 corresponding to the axioms (in), (out) and (open). Consider first the case of the (in) axiom. If $P \rightarrow Q$ is an instance of the (in) axiom then there must be processes P', Q', R' and names n, m such that $P = n[\text{in } m.P' \mid Q'] \mid m[R']$ and $Q = m[n[P' \mid Q'] \mid R']$. This implies

$$\begin{aligned}
P_0 &= \Diamond \left(n \wedge P'_0 \wedge Q'_0 \wedge \downarrow x.(\Diamond^- \Diamond m \rightarrow @_m \Diamond x) \right) \wedge \Diamond \left(m \wedge R'_0 \right) \\
Q_0 &= \Diamond \left(m \wedge R'_0 \wedge \Diamond (n \wedge P'_0 \wedge Q'_0) \right).
\end{aligned}$$

We now need to prove $\models P_0 \rightarrow Q_0$. By Lemma 1, it suffices to prove validity of the formula $\psi = (P_0 \rightarrow Q_0)[p/P'_0, q/Q'_0, r/R'_0]$, where p, q and r are arbitrarily chosen propositional symbols. The theorem prover *HyLoTab* can be used to verify the validity of ψ in the following way. First ψ is negated, and then the negated formula is translated into the syntactic form expected by *HyLoTab* (see [12]):

$$\begin{aligned}
& \neg((\langle \rangle \text{conj}(c0, p0, p1, Dx. (\langle \sim \rangle \langle \rangle c1 \rightarrow @c1 \langle \rangle x)) \\
& \ \& \langle \rangle \text{conj}(c1, p2)) \rightarrow \langle \rangle \text{conj}(c1, p2, \langle \rangle \text{conj}(c0, p0, p1))). \quad (2)
\end{aligned}$$

To conform to the *HyLoTab* syntax, n has been replaced by $c0$, m by $c1$, p by $p0$, q by $p1$ and r by $p2$. Now *HyLoTab* is run with the formula (2) as input, and it is checked that *HyLoTab* terminates with output “not satisfiable”. Since *HyLoTab* thus proves $\neg\psi$ to be non-satisfiable, ψ must be valid, as required. This concludes the case of the (in) axiom. The cases of the (out) and (open) axioms are similar, and have been verified by *HyLoTab* as well. Thus the base cases are completed. We now turn to the induction step. Assume that $P \rightarrow Q$ is the root of a proof tree of depth n , and suppose that for any reduction step $P' \rightarrow Q'$ with a proof tree of depth less than n we have $\models P'_0 \rightarrow Q'_0$. There are now four cases to consider, one for each of the four possible proof rules (amb), (ν) , (\mid) and (\equiv) that could have produced the reduction step $P \rightarrow Q$. Consider first the case of the (amb) rule. If $P \rightarrow Q$ has been produced by (amb) then there must exist processes P', Q' and a name n such that $P = n[P']$, $Q = n[Q']$,

and there is a proof tree of depth $n - 1$ for the reduction step $P' \rightarrow Q'$. By induction hypothesis we then get $\models P'_0 \rightarrow Q'_0$. Furthermore, by the definition of $(\cdot)_0$ we get $P_0 = \Diamond(n \wedge P'_0)$ and $Q_0 = \Diamond(n \wedge Q'_0)$. Since $P'_0 \rightarrow Q'_0$ is valid it is easy to see that $P_0 \rightarrow Q_0$ must be valid as well, as required. This concludes the case of the (amb) rule. The cases of the three remaining rules, (ν) , $(|)$ and (\equiv) , are equally simple. In the case of the (\equiv) rule, Lemma 2 immediately gives the required conclusion.

We will now show how the translation $(\cdot)_0$ gives rise to a correct analysis of the ambient calculus. First we need a few new notions. Let $\mathcal{M} = (W, R, V)$ be a model. The *nominal accessibility relation* of \mathcal{M} is the following relation:

$$\{(m, n) \in \text{Nom}^2 \mid (V(m), V(n)) \in R\}.$$

Thus the nominal accessibility relation contains the set of pairs of nominals (m, n) for which the world denoted by n is accessible from the world denoted by m . It is simply the accessibility relation of \mathcal{M} translated into a relation on nominals. Let P be a process and let m and n be names occurring in P . We say that m *contains* n in P , written $m >_P n$, if there exists a subprocess P' of P satisfying $P' \equiv m[Q \mid n[R]]$. The intuition here is that ‘ m contains n ’ means that the ambient named n runs immediately inside the ambient named m . A *correct analysis* of a process P is a set M of pairs of ambient names satisfying:

For any process Q and any pair of ambient names m and n , if $P \rightarrow^* Q$ and $m >_Q n$ then $(m, n) \in M$.

Thus if M is a correct analysis of a process P and if (m, n) is a pair of names *not* belonging to M , then we know that it is impossible for n to end up running immediately inside m . This is what allows one to prove security properties of e.g. firewalls using correct analyses: If a correct analysis of a process P does not contain a pair (m, n) then we have verified that n can never enter m .

Theorem 2 (Correctness). *Given any process P and any model \mathcal{M} satisfying P_0 , the nominal accessibility relation of \mathcal{M} is a correct analysis of P .*

Proof (Sketch). First it is proved that the following holds:

$$\begin{aligned} &\text{For all processes } Q, \text{ all ambient names } m, n, \text{ and all models } \mathcal{M}, \\ &\text{if } m >_Q n \text{ and } \mathcal{M} = (W, R, V) \text{ satisfies } Q_0 \text{ then } (V(m), V(n)) \in R. \end{aligned} \quad (3)$$

This property is proved by induction on the syntactic structure of Q . We leave out the details, as all cases in the induction proof are easily carried out using the relevant clauses in the definition of $(\cdot)_0$. Now assume P is a process and $\mathcal{M} = (W, R, V)$ is a model satisfying P_0 . What we need to prove is the following:

If Q is a process such that $P \rightarrow^* Q$ and $m >_Q n$ then $(V(m), V(n)) \in R$.

Let thus a process Q be given such that $P \rightarrow^* Q$ and $m >_Q n$. By subject reduction, Theorem 1, we get $\models P_0 \rightarrow Q_0$. Since \mathcal{M} satisfies P_0 it must also satisfy Q_0 . We can now apply (3) to conclude $(V(m), V(n)) \in R$, as required. This concludes the proof.

The theorem above shows that to make an analysis of a process P we simply need to find a model of P_0 . This can be done e.g. by using *HyLoTab*, since *HyLoTab* is a tableau-based theorem prover, and given a satisfiable formula as input it will return a model of that formula. Let us consider an example.

Example 1 (A simple process). Consider the simple process P given by $P = a[\text{open } b.c[0] \mid b[0]]$. We see that b is contained in a in this process. Applying the (open) axiom and the (amb) rule to the process we get $P \rightarrow a[c[0] \mid 0]$. So P reduces to a process in which c is contained in a . Thus a correct analysis of P must contain at least the pairs (a, b) and (a, c) —where the first pair corresponds to the fact that a contains b in P , and the second pair corresponds to the fact that P can be reduced to a process in which a contains c . We will now try to apply the machinery above to construct an analysis of P . By the correctness theorem above, Theorem 2, the nominal accessibility relation of any model of P_0 will be a correct analysis. We can use *HyLoTab* to compute a model of P_0 by simply giving the formula as input to *HyLoTab*. *HyLoTab* then returns the model $\mathcal{M} = (W, R, V)$ given by: $W = \{w_1, w_2, w_3\}$; $R = \{(w_1, w_2), (w_2, w_2), (w_2, w_3)\}$; $V(a) = w_2, V(b) = w_2, V(c) = w_3$. This model is illustrated in Figure 9. From

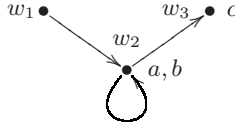
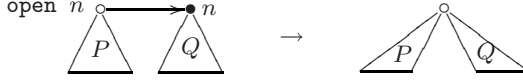
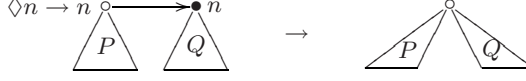
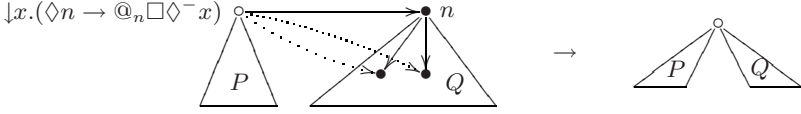


Fig. 9. Model of P_0

the figure we see that the nominal accessibility relation of \mathcal{M} is $\{a, b\} \times \{a, b, c\}$. This set constitutes, by Theorem 2, a correct analysis of the process P . We see that the analysis contains the pairs (a, b) and (a, c) as expected. However, it also contains the pair (b, a) which actually does not correspond to a possible configuration, since it is easily seen that no sequence of reduction steps can reduce P to a process where b contains a . This is because our treatment of the open capability in the translation $(\cdot)_0$ is a bit too simplistic.

Let us take a closer look at how we translate the open capability. A tree visualisation of the open axiom ($\text{open } n.P \mid n[Q] \rightarrow P \mid Q$) is given in Figure 10. Replacing the capability open n in Figure 10 by its translation $\Diamond n \rightarrow n$ we get the hybrid logical visualisation given in Figure 11. Consider the left-hand tree in this figure. Since the formula $\Diamond n \rightarrow n$ holds at the root of this tree, and since the node labelled n is accessible from the root, the nominal n must also hold at the root of the tree. Since nominals are true at unique worlds, the two nodes must be identical. In other words, the two nodes are collapsed into one, corresponding to the right-hand tree in the figure. This shows that the translation of the open capability behaves as expected. However, there is one complication. The collapsed node will obviously be labelled by n , but the root of the right-hand tree is not labelled by n . Since a correct analysis is an *over-approximation*, adding n as label

Fig. 10. Tree visualisation of the **open** axiomFig. 11. Hybrid logical visualisation of the **open** axiom using $(\cdot)_0$ Fig. 12. A better hybrid logical visualisation of the **open** axiom

to some node can never affect correctness, but it can affect the precision of the analysis. What we would like is a more precise translation of the open capability that does not add n as label to the root node. That is, we seek a translation that ‘copies’ the subtree Q at the node labelled n to the root node, but does not also copy the label n . Such a ‘copying’ of Q can be performed by the following process: For every node u in Q , if there is an edge from n to u then add an edge from the root to u . We can also express this process as a hybrid logical formula at the root node: $\downarrow x.(\Diamond n \rightarrow @_n \Box \Diamond^- x)$. If we translate the **open** capability by this formula rather than simply $\Diamond n \rightarrow n$ then we get the hybrid logical visualisation in Figure 12. The dashed edges correspond to the edges that are added by the new translation. In the following section we show that using this more complicated translation of the **open** capability actually yields a more precise analysis. In particular, it will solve the imprecision problem discussed in Example 1.

6 The Not-So-Naive Analysis

We now try to construct a slightly more precise analysis. It is based on a translation $(\cdot)_1$ from processes into hybrid logical formulae given by the following clauses.

$$\begin{aligned}
 ((\nu n)P)_1 &= P_1 \\
 \mathbf{0}_1 &= \top \\
 (P \mid Q)_1 &= P_1 \wedge Q_1 \\
 (!P)_1 &= P_1 \\
 (n[P])_1 &= \Diamond(n \wedge P_1) \\
 (\text{in } n.P)_1 &= P_1 \wedge \Diamond \Box^- \downarrow x.(\Diamond^- \Diamond n \rightarrow @_n \Diamond x) \\
 (\text{out } n.P)_1 &= P_1 \wedge \Diamond \Box^- \downarrow x.(\Diamond^- n \rightarrow @_n \Box^- \Diamond x) \\
 (\text{open } n.P)_1 &= P_1 \wedge \Diamond \Box^- \downarrow x.(\Diamond n \rightarrow @_n \Box \Diamond^- x).
 \end{aligned}$$

Note that the first five clauses in the definition of $(\cdot)_1$ are identical to the corresponding five clauses for $(\cdot)_0$. Furthermore, the clauses for the **in** and **out** capabilities only differ from the corresponding $(\cdot)_0$ clauses by adding $\Diamond\Box^-$ in front of the downarrow binder. The clauses for the **in**, **out** and **open** capabilities are now very similar. The translations of **in** $n.P$, **out** $n.P$ and **open** $n.P$ are all on the form

$$P_1 \wedge \Diamond\Box^- \downarrow x.(An \rightarrow @_n Bx)$$

where A and B are combined modal operators given by:

formula	value of A	value of B
in $n.P$	$\Diamond^- \Diamond$	\Diamond
out $n.P$	\Diamond^-	$\Box^- \Diamond$
open $n.P$	\Diamond	$\Box \Diamond^-$

Note in particular the duality between the translations of **out** $n.P$ and **open** $n.P$.

We can now prove the same results for $(\cdot)_1$ as we did for $(\cdot)_0$.

Lemma 3. *For all processes P and Q , if $P \equiv Q$ then $\models P_1 \leftrightarrow Q_1$.*

Proof. Identical to the proof of Lemma 2 (simply replace 0 by 1 everywhere).

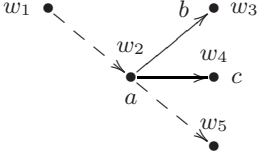
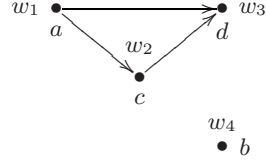
Theorem 3 (Subject Reduction for $(\cdot)_1$). *For all processes P and Q , if $P \rightarrow Q$ then $\models P_1 \rightarrow Q_1$.*

Proof (Sketch). Since the first five clauses are identical for $(\cdot)_0$ and $(\cdot)_1$, we can reuse most of the proof of the subject reduction result for $(\cdot)_0$, Theorem 1. We only need to check the cases where $P \rightarrow Q$ has been produced by one of the rules (**in**), (**out**) or (**open**). In each case we again end up with an implication $P_1 \rightarrow Q_1$ that we need to prove valid, and again we use *HyLoTab* to verify the validity.

Theorem 4 (Correctness). *Given any process P and any model \mathcal{M} satisfying P_1 , the nominal accessibility relation of \mathcal{M} is a correct analysis of P .*

Proof. Identical to the proof of Theorem 2 (simply replace 0 by 1 everywhere).

Example 2 (A simple process). In Example 1 we considered the process P given by $P = a[\text{open } b.c[0] \mid b[0]]$. Let us see how our new analysis based on the translation $(\cdot)_1$ behaves with respect to P . Giving P_1 as input formula to *HyLoTab* it produces the model $\mathcal{M} = (W, R, V)$ given by $W = \{w_1, w_2, w_3, w_4, w_5\}$; $R = \{(w_1, w_2), (w_2, w_3), (w_2, w_4), (w_2, w_5)\}$; $V(a) = w_2, V(b) = w_3, V(c) = w_4$. This model is illustrated in Figure 13. The dashed edges are the edges that either start or end in a node not labelled by any nominal. Such edges do not affect the nominal accessibility relation, and from now on we simply omit them. From the figure we see that the nominal accessibility relation of \mathcal{M} is $\{(a, b), (a, c)\}$. By Theorem 4, this set is a correct analysis of P . Since the set is a proper subset of the analysis we obtained in Example 1, we have now indeed obtained a more precise analysis. Actually, the present analysis of P contains only the two pairs that any correct analysis of P is required to contain, by Example 1.

Fig. 13. Model of P_1 Fig. 14. Model of Q_1

Example 3 (Nested capabilities). Let us consider a slightly more complex example. Let Q be the process given by $Q = a[\text{open } b.\text{open } c.0 \mid c[d[0]]]$. Using Q_1 as input to *HyLoTab* we get the model \mathcal{M} illustrated in Figure 14. Its nominal accessibility relation is $\{(a, c), (a, d), (c, d)\}$. This is the same analysis as provided by the OCFA analysis of [7]. However, the analysis is not completely precise. There are no non-trivial reductions that can be made on Q since there is no ambient named b that the outermost **open** capability can be applied to. Thus we would expect a precise analysis to only give the pairs (a, c) and (c, d) corresponding to the fact that a contains c and c contains d in Q . The reason that the present analysis and the OCFA of [7] do not give this result is that they both ignore the nesting order on the capabilities. In the case of Q this means that the relative ordering of the two capabilities **open** b and **open** c is ignored. Using the hybrid logical machinery we have developed it is fortunately quite easy to improve the analysis and obtain one which takes the ordering of capabilities into account.

Consider the following sequence of capabilities: **in** k .**out** l .**open** m .**0**. Using the present translation we get:

$$(\text{in } k.\text{out } l.\text{open } m.0)_1 = (\text{in } k.0)_1 \wedge (\text{out } l.0)_1 \wedge (\text{open } m.0)_1.$$

This implies that each of the three individual capabilities **in** k , **out** l and **open** m will be ‘executed’ in the same world of the Kripke model induced by the translated formula. This of course means that the translation is flow-*insensitive*: The ordering of the capabilities is not preserved under the translation. However, since we are working with graphs, there is a simple way to solve this problem. Instead of translating a capability like **in** k . P into a formula of the form $P_1 \wedge \psi$ we could translate it into a formula of the form $\Diamond(P_1 \wedge \psi)$. Then each capability will add a new edge to the graph because of the \Diamond in front of the formula, and the body of the capability will be ‘executed’ at the world that this new edge leads to. Thus sequences of capabilities will be translated into paths, and the ordering of them will thus be preserved. In the following section we show how to construct such a flow-sensitive translation.

7 A Better Analysis

We now try to construct an even more precise analysis. It is based on a translation $(\cdot)_2$ from processes into hybrid logical formulae given by the following clauses.

$$\begin{aligned}
((\nu n)P)_2 &= P_2 \\
\mathbf{0}_2 &= \top \\
(P \mid Q)_2 &= P_2 \wedge Q_2 \\
(!P)_2 &= P_2 \\
(n[P])_2 &= \Diamond(n \wedge P_2) \\
(\text{in } n.P)_2 &= \Diamond(P_2 \wedge \downarrow y. \Box^- \downarrow x. (\Diamond^- \Diamond n \rightarrow @_n \Diamond x \wedge @_y \Box \Diamond^- x)) \\
(\text{out } n.P)_2 &= \Diamond(P_2 \wedge \downarrow y. \Box^- \downarrow x. (\Diamond^- n \rightarrow @_n \Box^- \Diamond x \wedge @_y \Box \Diamond^- x)) \\
(\text{open } n.P)_2 &= \Diamond(P_2 \wedge \downarrow y. \Box^- \downarrow x. (\Diamond n \rightarrow @_n \Box \Diamond^- x \wedge @_y \Box \Diamond^- x)).
\end{aligned}$$

Note that the first five clauses in the definition of $(\cdot)_2$ are identical to the corresponding five clauses for $(\cdot)_0$ and $(\cdot)_1$. Furthermore, the clauses for the **in**, **out** and **open** capabilities are quite similar to the corresponding clauses for $(\cdot)_1$. The translation of **in** $n.P$, **out** $n.P$ and **open** $n.P$ are now all on the form

$$\Diamond(P_2 \wedge \downarrow y. \Box^- \downarrow x. (A n \rightarrow @_n B y \wedge @_x \Box \Diamond^- y))$$

where A and B are the same combined modal operators as for $(\cdot)_1$.

Lemma 4. *For all processes P and Q , if $P \equiv Q$ then $\models P_2 \leftrightarrow Q_2$.*

Proof. Identical to the proof of Lemma 2 (simply replace 0 by 2 everywhere).

Theorem 5 (Subject Reduction for $(\cdot)_2$). *For all processes P and Q , if $P \rightarrow Q$ then $\models P_2 \rightarrow Q_2$.*

Proof. Since the first five clauses are identical for $(\cdot)_0$, $(\cdot)_1$ and $(\cdot)_2$, we again only need to consider the rules (**in**), (**out**) and (**open**). These cases are treated similarly to the corresponding cases in the proof of subject reduction for $(\cdot)_1$ and $(\cdot)_0$.

Theorem 6 (Correctness). *Given any process P and any model \mathcal{M} satisfying P_2 , the nominal accessibility relation of \mathcal{M} is a correct analysis of P .*

Proof. Identical to the proof of Lemma 2 (simply replace 0 by 2 everywhere).

Example 4 (Nested capabilities). In Example 3 we considered the process Q given by $Q = a[\text{open } b.\text{open } c.\mathbf{0} \mid c[d[\mathbf{0}]]]$. As mentioned in Example 3, the translation $(\cdot)_1$ gave us an analysis of Q which was not completely precise. Let us see whether the translation $(\cdot)_2$ performs better. Giving Q_2 as input to *HyLoTab* we obtain the model \mathcal{M} presented in Figure 15. This model is exactly as the model of Q_1

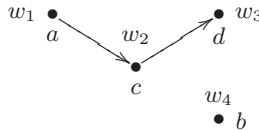


Fig. 15. Model of Q_2

except that the edge from w_1 to w_3 is no longer present. Thus we instead get the following nominal accessibility relation: $\{(a, c), (c, d)\}$. This relation is the analysis we expected and it is seen to be more precise than both our analysis based on $(\cdot)_1$ and the OCFA analysis of [7]. This improved precision is gained by making the analysis of capability sequences flow sensitive (cf. the concluding discussion at the end of Section 6).

8 Complexity Issues and Future Work

So far in this paper we have not dealt with questions concerning computability and complexity. Since the full hybrid logic $HL(@, -, \downarrow)$ is actually undecidable, it is not immediately obvious that the formulae generated by our translations are always computable. Fortunately, the translations are designed in such a way that they translate into a decidable fragment of hybrid logics called the *innocent fragment* [11]. A formula ϕ is called *innocent* if every binder subformula $\downarrow x.\psi$ is only outscoped by \Diamond 's being in the scope of an even number of negation operators (when counting negation operators it is assumed that ϕ uses only the primitive operators, that is, not \rightarrow , \Box or \Box^-). It is easily seen that our translations all fall within the innocent fragment. In [11] van Eijck proves that *HyLoTab* decides the innocent fragment. Thus given *any* process P , *HyLoTab* will necessarily terminate on any translation of P resulting in a model for the translation and thereby an analysis of P .

Furthermore, it appears that the fragment of hybrid logic used in our translations is sufficiently simple that it is possible to obtain a complexity bound on the model generation that is comparable to existing analysis tools for the ambient calculus. However, we leave this investigation for future work. Another line of future work we are interested in is to try to make even better analyses than the one provided by $(\cdot)_2$. We believe that the possibilities of using hybrid logic for precise and yet compact as well as clear analysis of the ambient calculus are yet far from exhausted.

References

1. Amtoft, T., Kfoury, A.J., Pericas-Geertsen, S.M.: What are polymorphically-typed ambients? In: Sands, D. (ed.) ESOP 2001 and ETAPS 2001. LNCS, vol. 2028, pp. 206–220. Springer-Verlag, Heidelberg (2001)
2. Amtoft, T., Makholm, H., Wells, J.B.: Polya: True type polymorphism for mobile ambients. In: Levy, J.-J., Mayr, E.W., Mitchell, J.C. (eds.) TCS 2004. 3rd IFIP International Conference on Theoretical Computer Science, Toulouse, France, August 2004, pp. 591–604. Kluwer Academic Publishers, Dordrecht (2004)
3. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, Cambridge (2001)
4. Cardelli, L., Gordon, A.D.: Types for Mobile Ambients. In: Conference Record of the Annual ACM Symposium on Principles of Programming Languages, POPL'99, pp. 79–92. ACM Press, New York (1999)

5. Cardelli, L., Gordon, A.D.: Anytime, Anywhere: Modal logics for mobile ambients. In: Conference Record of the Annual ACM Symposium on Principles of Programming Languages, POPL'00, pp. 365–377. ACM Press, New York (2000)
6. Cardelli, L., Gordon, A.D.: Mobile ambients. *Theoretical Computer Science* 240, 177–213 (2000)
7. Nielson, F., Hansen, R.R., Nielson, H.R.: Abstract Interpretation of Mobile Ambients. *Science of Computer Programming* 47(2–3), 145–175 (2003)
8. Nielson, F., Hansen, R.R., Nielson, H.R.: Validating Firewalls using Flow Logics. *Theoretical Computer Science* 283(2), 381–418 (2002)
9. Nielson, H.R., Nielson, F.: Shape analysis for mobile ambients. In: Conference Record of the Annual ACM Symposium on Principles of Programming Languages, POPL'00, pp. 142–154. ACM Press, New York (2000)
10. Nielson, H.R., Nielson, F.: Flow Logic: a multi-paradigmatic approach to static analysis. In: Mogensen, T., Schmidt, D.A., Sudborough, I.H. (eds.) *The Essence of Computation*. LNCS, vol. 2566, pp. 223–244. Springer, Heidelberg (2002)
11. van Eijck, J.: Constraint tableaux for hybrid logics. Manuscript, CWI, Amsterdam (2002)
12. van Eijck, J.: Hylotab—tableau-based theorem proving for hybrid logics. Manuscript, CWI, Amsterdam (2002)

Structured Anaphora to Quantifier Domains: A Unified Account of Quantificational and Modal Subordination

Adrian Brasoveanu*

Stanford University and University of California at Santa Cruz

Abstract. The paper proposes an account of the contrast (noticed in [9]) between the interpretations of the following two discourses: *Harvey courts a girl at every convention. {She is very pretty. vs. She always comes to the banquet with him.}*. The initial sentence is ambiguous between two quantifier scopings, but the first discourse as a whole allows only for the wide-scope indefinite reading, while the second allows for both. This cross-sentential interaction between quantifier scope and anaphora is captured by means of a new dynamic system couched in classical type logic, which extends Compositional DRT ([16]) with *plural information states* (modeled, following [24], as sets of variable assignments). Given the underlying type logic, compositionality at sub-clausal level follows automatically and standard techniques from Montague semantics become available. The paper also shows that modal subordination (*A wolf might come in. It would eat Harvey first*) can be analyzed in a parallel way, i.e. the system captures the anaphoric and quantificational parallels between the individual and modal domains argued for in [23]. In the process, we see that modal/individual-level quantifiers enter anaphoric connections as a matter of course, usually functioning simultaneously as both indefinites and pronouns.

1 Introduction: Quantificational Subordination

The present paper proposes an account of the contrast between the interpretations of the discourses in (1) and (2) below from [9] (the superscripts and subscripts indicate the antecedent-anaphor relations).

1. **a.** Harvey courts a^u girl at every convention. **b.** She_u is very pretty.
2. **a.** Harvey courts a_u girl at every convention. **b.** She_u always comes to the banquet with him. **[c.** The_u girl is usually also very pretty.]

* I am grateful to Maria Bittner, Sam Cumming, Hans Kamp, Roger Schwarzschild, Matthew Stone and three WoLLIC 2007 reviewers for their detailed comments and to Daniel Althshuler, Pranav Anand, Donka Farkas, Carlos Fasola, Michael Johnson, Ernie Lepore, Friederike Moltmann, Sarah Murray, Jessica Rett, Ken Shan, Will Starr and the Rutgers Semantics Reading Group (Nov 14, 2006) for discussion. The financial support provided by Stanford University (Humanities Fellows Program) is gratefully acknowledged. The usual disclaimers apply.

Sentence (1a/2a) by itself is ambiguous between two quantifier scopings: it “can mean that, at every convention, there is some girl that Harvey courts or that there is some girl that Harvey courts at every convention. [...] Harvey always courts the same girl [...] [or] it may be a different girl each time” ([9]: 377). The contrast between the continuations in (1b) and (2b) is that the former allows only for the ‘same girl’ reading of sentence (1a/2a), while the latter is also compatible with the ‘possibly different girls’ reading.

Discourse (1) raises the following question: how can we capture the fact that a *singular anaphoric pronoun* in sentence (1b) can interact with and disambiguate *quantifier scopings*¹ in sentence (1a)? That number morphology on the pronoun *she* is crucial is shown by the discourse in (3) below, where the (preferred) relative scoping of *every convention* and *a girl* is the opposite of the one in discourse (1).

3. **a.** Harvey courts a^u girl at every convention. **b.** They_u are very pretty.

Discourse (2) raises the following questions. First, why is it that adding an adverb of quantification, i.e. *always/usually*, makes both readings of sentence (2a) available? Moreover, on the newly available reading of sentence (2a), i.e. the *every convention* >> *a girl* scoping, how can we capture the intuition that the singular pronoun *she* and the adverb *always* in sentence (2b) elaborate on the quantificational dependency between conventions and girls introduced in sentence (2a), i.e. how can we capture the intuition that we seem to have simultaneous anaphora to the two quantifier domains and to the quantificational dependency between them?

The phenomenon instantiated by discourses (1) and (2) is subsumed under the more general label of *quantificational subordination* (see [13]: 139, (2)), which covers a variety of phenomena involving interactions between generalized quantifiers and morphologically singular cross-sentential anaphora. The main goal of this paper is give an account of quantificational subordination couched within a new compositional dynamic system which straightforwardly generalizes to an account of modal subordination, thereby capturing the anaphoric and quantificational parallels between the individual and modal domains argued for in [23], [1], [22], [5] and [7] among others (building on [18] and [19]).

2 Plural Compositional DRT (PCDRT)

This section introduces the semantic framework in which the analysis of discourses (1) and (2) is couched. The main proposal is that (compositionally) assigning natural language expressions finer-grained semantic values (finer grained than the usual meanings assigned in static Montague semantics) enables us to

¹ To see that it is indeed quantifier scopings that are disambiguated, substitute *exactly one^u girl* for *a^u girl* in (1a); this yields two truth-conditionally independent scopings: (i) *exactly one girl* >> *every convention*, which is true in a situation in which Harvey courts more than one girl per convention, but there is exactly one (e.g. Faye Dunaway) that he never fails to court, and (ii) *every convention* >> *exactly one girl*.

capture the interaction between generalized quantifiers, singular pronouns and adverbs of quantification exhibited by the contrast between (1) and (2).

Accounting for *cross-sentential* phenomena in semantic terms (as opposed to purely/primarily pragmatic terms) requires some preliminary justification. First, the same kind of finer-grained semantic values (to be introduced presently) are independently motivated by intra-sentential phenomena (see the account of mixed weak & strong donkey sentences in [2]). Second, the phenomenon instantiated by (1) and (2) is as much intra-sentential as it is cross-sentential. Note that there are four separate components that come together to yield the contrast in interpretation between (1) and (2): (i) the generalized quantifier *every convention*, (ii) the indefinite *a girl*, (iii) the singular number morphology on the pronoun *she* and (iv) the adverb of quantification *always/usually*. To derive the intuitively correct interpretations for (1) and (2), we have to attend to both the cross-sentential connections *a girl–she* and *every convention–always/usually* and the intra-sentential interactions *every convention–a girl* and *always–she*.

I conclude that an account of the contrast between (1) and (2) that involves a revamping of semantic values has sufficient initial plausibility to make its pursuit worthwhile. To this end, I introduce a new dynamic system couched in classical (many-sorted) type logic which extends Compositional DRT (CDRT, [16]) in two ways: (i) with plural information states and (ii) with selective generalized quantification. The resulting system is dubbed Plural CDRT (PCDRT).

2.1 Plural Information States

The main technical innovation relative to CDRT is that, just as in Dynamic Plural Logic ([24]), information states I, J etc. are modeled as *sets* of variable assignments i, j etc.; such *plural* info states can be represented as matrices with assignments (sequences) as rows, as shown below.

Info State I	...	u	u'	...
i_1	...	x_1 (i.e. ui_1)	y_1 (i.e. $u'i_1$)	...
i_2	...	x_2 (i.e. ui_2)	y_2 (i.e. $u'i_2$)	...
i_3	...	x_3 (i.e. ui_3)	y_3 (i.e. $u'i_3$)	...
...
Quantifier domains (sets)		Quantifier dependencies (relations)		
are stored columnwise: $\{x_1, x_2, x_3, \dots\}, \{y_1, y_2, y_3, \dots\}$		are stored rowwise: $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots\}$		

Plural info states enable us to encode discourse reference to both quantifier domains, i.e. *values*, and quantificational dependencies, i.e. *structure*. The values are the sets of objects that are stored in the columns of the matrix, e.g. a discourse referent (dref) u for individuals stores a set of individuals relative to a plural info state given that u is assigned an individual by each assignment/row. The structure is encoded in the rows of the matrix: for each assignment/row in the info state, the individual assigned to a dref u by that assignment is structurally correlated with the individual assigned to some other dref u' by the same assignment.

2.2 The Outline of the Proposed Account

Thus, plural info states enable us to pass information about both quantifier domains and quantificational dependencies across sentential/clausal boundaries, which is exactly what we need to account for the interpretation of discourses (1) and (2). More precisely, we need the following two ingredients.

First, we need a suitable meaning for selective generalized determiners that will store two things in the input plural info state: (i) the restrictor and nuclear scope sets of individuals that are introduced and related by the determiner; (ii) the quantificational dependencies between the individuals in the restrictor/nuclear scope set and any other quantifiers/indefinites in the restrictor/nuclear scope of the quantification, e.g. between *every convention* in (1a/2a) and the indefinite *a girl* in its nuclear scope. Given that plural info states store both sets of individuals and dependencies between them, both kinds of information are available for subsequent anaphoric retrieval; for example, *always* and *she* in (2b) are simultaneously anaphoric to both the sets of conventions and girls and the dependency between these sets introduced in (2a).

The second ingredient is a suitable meaning for singular number morphology on pronouns like *she_u* in (1b) and (2b) above. This meaning has to derive the observed interactions between (i) singular pronouns, (ii) quantifiers and indefinites in the previous discourse, e.g. *every convention* and *a^u girl* in (1a/2a), and (iii) quantifiers in the same sentence, e.g. the adverb *always* in (2b). In particular, I will take singular number morphology on *she_u* to require the set of *u*-individuals stored by the current plural info state to be a singleton. The set of *u*-individuals is introduced by the indefinite *a^u girl* and is available for anaphoric retrieval irrespective of whether the indefinite has wide or narrow scope in sentence (1a/2a). Thus, once again, plural info states are crucial for the analysis: they enable us to store and pass on structured sets of individuals, so that we can constrain their cardinality by subsequent, non-local anaphoric elements.

If the indefinite *a^u girl* has narrow scope relative to *every convention*, the singleton requirement contributed by *she_u* applies to the set of all girls that are courted by Harvey at some convention or other. Requiring this set to be a singleton boils down to removing from consideration all the plural info states that would satisfy the narrow-scope indefinite reading *every convention* >> *a^u girl*, but not the wide-scope reading *a^u girl* >> *every convention*. We therefore derive the intuition that, irrespective of which quantifier scoping we assume for sentence (1a), any plural info state that we obtain after a successful update with sentence (1b) is bound to satisfy the representation in which the indefinite *a^u girl* (or a quantifier like *exactly one^u girl*) takes wide scope.

In discourse (2), however, the adverb of quantification *always* in (2b), which is anaphoric to the nuclear scope set introduced by *every convention* in (2a), can take scope either below or above the singular pronoun *she_u*. If *always* takes scope below *she_u*, we obtain the same reading as in discourse (1). If *always* takes scope above *she_u*, it ‘breaks’ the input plural info state storing all the conventions into smaller sub-states, each storing a particular convention. Consequently, the singleton requirement contributed by *she_u* is enforced locally, relative to each of

these sub-states, and not globally, relative to the whole input info state, so we end up requiring the courted girl to be unique *per convention* and not across the board.

The remainder of this section presents the basics of the compositional dynamic system, while Section 3 introduces the PCDRT meanings for selective generalized determiners, indefinites and singular/plural pronouns.

2.3 DRS's and Conditions in PCDRT

We work with a Dynamic Ty2 logic, i.e. with the Logic of Change in [16] which reformulates dynamic semantics ([8], [12]) in Gallin's Ty2 ([6]). We have three basic types: type t (truth-values), type e (individuals; variables: x, x' etc.) and type s ('variable assignments'; variables: i, j etc.). A suitable set of axioms ensures that the entities of type s do behave as variable assignments².

A dref for individuals u is a function of type se from 'assignments' i_s to individuals x_e (the subscripts on terms indicate their type). Intuitively, the individual $u_{se}i_s$ is the individual that the 'assignment' i assigns to the dref u . Dynamic info states I, J etc. are plural: they are sets of 'variable assignments', i.e. terms of type st . An individual dref u stores a set of individuals with respect to a plural info state I , abbreviated as $uI := \{u_{se}i_s : i_s \in I_{st}\}$, i.e. uI is the image of the set of 'assignments' I under the function u .

A sentence is interpreted as a Discourse Representation Structure (DRS), which is a relation of type $(st)((st)t)$ between an input state I_{st} and an output state J_{st} , as shown in (4) below. A DRS requires: (i) the input info state I to differ from the output state J at most with respect to the **new dref's** and (ii) all the **conditions** to be satisfied relative to the output state J . For example, the DRS $[u_1, u_2][girl\{u_1\}, convention\{u_2\}, courted_at\{u_1, u_2\}]$ abbreviates the term $\lambda I_{st}.\lambda J_{st}. I[u_1, u_2]J \wedge girl\{u_1\}J \wedge convention\{u_2\}J \wedge courted_at\{u_1, u_2\}J$. The definition of dref introduction (a.k.a. random assignment) is given in (5) below³.

4. **[new dref's | conditions]** $:= \lambda I_{st}.\lambda J_{st}. I[\text{new dref's}]J \wedge \text{conditions}J$
5. $[u] := \lambda I_{st}.\lambda J_{st}. \forall i_s \in I(\exists j_s \in J(i[u]j)) \wedge \forall j_s \in J(\exists i_s \in I(i[u]j))$

DRS's of the form **[conditions]** $:= \lambda I_{st}.\lambda J_{st}. I = J \wedge \text{conditions}J$ are *tests*, e.g. $[girl\{u_1\}] := \lambda I_{st}.\lambda J_{st}. I = J \wedge girl\{u_1\}J$ tests that the input state I satisfies the condition $girl\{u_1\}$. Conditions are interpreted *distributively* relative to a plural info state, e.g. $courted_at\{u_1, u_2\}$ is basically the term $\lambda I_{st}. I \neq \emptyset \wedge \forall i_s \in I(courted_at(u_1i, u_2i))$ of type $(st)t$, i.e. it denotes a set of information states; see Subsect. 3.1 below for the general definition of atomic conditions.

2.4 Compositionality

Given the underlying type logic, compositionality at sub-clausal level follows automatically and standard techniques from Montague semantics (e.g. type shifting) become available. In more detail, the compositional aspect of interpretation

² See [16] for more details.

³ See [2] for its justification.

in an extensional Fregean/Montagovian framework is largely determined by the types for the (extensions of the) ‘saturated’ expressions, i.e. names and sentences. Abbreviate them as **e** and **t**. An extensional static logic identifies **e** with *e* and **t** with *t*. The denotation of the noun *girl* is of type **et**, i.e. $et: girl \rightsquigarrow \lambda x_e. girl_{et}(x)$. The generalized determiner *every* is of type **(et)((et)t)**, i.e. $(et)((et)t)$.

PCDRT assigns the following dynamic types to the ‘meta-types’ **e** and **t**: **t** abbreviates $(st)((st)t)$, i.e. a sentence is interpreted as a DRS, and **e** abbreviates *se*, i.e. a name is interpreted as a dref. The denotation of the noun *girl* is still of type **et**, as shown in (6) below. Moreover, the determiner *every* is still of type **(et)((et)t)** – and its definition is provided in the next section.

$$6. \textit{girl} \rightsquigarrow \lambda v_e. [girl_{et}\{v\}], \quad \text{i.e. } \textit{girl} \rightsquigarrow \lambda v_e. \lambda I_{st}. \lambda J_{st}. I = J \wedge girl_{et}\{v\}J$$

3 Generalized Quantification in PCDRT

We turn now to the definition of selective generalized quantification in PCDRT. The definition has to satisfy four desiderata, the first three of which are about anaphoric connections that can be established *internally* (within the generalized quantification), i.e. between antecedents in the restrictor and anaphors in the nuclear scope, and the last of which is about anaphora that can be established *externally*, i.e. between antecedents introduced by/within the quantification and anaphors that are outside the quantification.

Let us begin with internal anaphora. First, we want our definition to be able to account for the fact that anaphoric connections between the restrictor and the nuclear scope of the quantification can in fact be established, i.e. we want to account for donkey anaphora (*Every^u farmer who owns a^{u'} donkey beats it_{u'}*).

Second, we want to account for such anaphoric connections while avoiding the proportion problem that *unselective* quantification (in the sense of [15]) runs into. That is, we need generalized determiners to relate sets of individuals (i.e. sets of objects of type *e*) and not sets of ‘assignments’ (i.e. sets of objects of type *s*). The sentence *Most^u farmers who own a^{u'} donkey beat it_{u'}* provides a typical instance of the proportion problem: intuitively, this sentence is false in a situation in which there are ten farmers, nine have a single donkey each and they do not beat it, while the tenth has twenty donkeys and he is busy beating them all. The unselective interpretation of the *most*-quantification, however, incorrectly predicts that the sentence is true in this situation because more than half of the (*farmer, donkey*) pairs (twenty out of twenty-nine) are such that the farmer beats the donkey.

The third desideratum is that the definition of selective generalized quantification be compatible with both strong and weak donkey readings: we want to allow for the different interpretations associated with the donkey anaphora in (7) (from [13]) and (8) (from [20]) below. Sentence (7) is interpreted as asserting that most slave-owners were such that, for *every* (strong reading) slave they owned, they also owned his offspring. Sentence (8) is interpreted as asserting that every dime-owner puts *some* (weak reading) dime of her/his in the meter.

7. Most^u people that owned a^{u'} slave also owned his_{u'} offspring.
8. Every^u person who has a^{u'} dime will put it_{u'} in the meter.

The fourth desideratum is concerned with external anaphora – and this brings us back to the discourses in (1) and (2). These discourses indicate that we need to make the restrictor and nuclear scope sets of individuals related by generalized determiners available for subsequent anaphora – and we also need to make available for anaphoric take-up the quantificational dependencies between different quantifiers and/or indefinites. In particular, generalized quantification supports anaphora to two sets: (i) the maximal set of individuals satisfying the restrictor DRS, i.e. the *restrictor set*, and (ii) the maximal set of individuals satisfying the restrictor and nuclear scope DRS's, i.e. the *nuclear scope set*⁴. Note that the latter set is the nuclear scope that emerges as a consequence of the conservativity of natural language quantification – and, as [24] (among others) observes, we need to build conservativity into the definition of dynamic quantification to account for the fact that the nuclear scope DRS can contain anaphors dependent on antecedents in the restrictor.

The discourse in (9) below exemplifies anaphora to nuclear scope sets: sentence (9b) is interpreted as asserting that the people that went to the beach are the students that left the party after 5 a.m. (which, in addition, formed a majority of the students at the party). The discourses in (10) and (11) below exemplify anaphora to the restrictor sets contributed by the downward monotonic quantifiers *no*^u *student* and *very few*^u *people* respectively. Consider (10) first: any successful update with a *no*^u quantification ensures that the nuclear scope set is empty and anaphora to it is therefore infelicitous; the only possible anaphora in (10) is restrictor set anaphora. Restrictor set anaphora is the only possible one in (11) also, because nuclear scope anaphora yields a contradictory interpretation for (11b): most of the people with a rich uncle that inherit his fortune don't inherit his fortune.

9. **a.** Most^u students left the party after 5 a.m. **b.** They_u went directly to the beach.
10. **a.** No^u student left the party later than 10 pm. **b.** They_u had classes early in the morning.
11. **a.** Very few^u people with a rich uncle inherit his fortune. **b.** Most of them_u don't.

Thus, a selective generalized determiner receives the translation in (12) below, which is in the spirit – but fairly far from the letter – of [24]⁵.

12. $\text{det}^{u,u'} \sqsubseteq^u \rightsquigarrow$
 $\lambda P_{\text{et}}. \lambda P'_{\text{et}}. \mathbf{max}^u(\langle u \rangle (P(u))); \mathbf{max}^{u' \sqsubseteq^u}(\langle u' \rangle (P'(u'))); [\mathbf{DET}\{u, u'\}]$

⁴ Throughout the paper, I will ignore anaphora to complement sets, i.e. sets obtained by taking the complement of the nuclear scope relative to the restrictor, e.g. *Very few students were paying attention in class. They were hungover.*

⁵ Cf. Definition (4.1), [24]: 149.

As expected, $\mathbf{det}^{u,u'} \sqsubseteq^u$ relates a restrictor dynamic property $P_{\mathbf{et}}$ and a nuclear scope dynamic property $P'_{\mathbf{et}}$. When these dynamic properties are applied to individual dref's, i.e. $P(u)$ and $P'(u')$, we obtain a restrictor DRS $P(u)$ and a nuclear scope DRS $P'(u')$ of type \mathbf{t} . Moreover, a generalized determiner introduces two individual dref's: u stores the restrictor set and u' the nuclear scope set. These two dref's and the two dynamic properties P and P' are the basic building blocks of the three separate updates in (12).

The first update, namely $\mathbf{max}^u(\langle u \rangle(P(u)))$, ensures that the restrictor set u is the maximal set of individuals, i.e. $\mathbf{max}^u(\dots)$, such that, when we take each u -individual separately, i.e. $\langle u \rangle(\dots)$, this individual satisfies the restrictor dynamic property, i.e. $P(u)$. The second update, namely $\mathbf{max}^{u'} \sqsubseteq^u(\langle u' \rangle(P'(u')))$, ensures that the nuclear scope set u' is obtained in much the same way as the restrictor set u , except for the requirement that u' is the maximal structured subset of u , i.e. $\mathbf{max}^{u'} \sqsubseteq^u(\dots)$. Finally, the third update, namely $[\mathbf{DET}\{u, u'\}]$, is a test: we test that the restrictor set u and the nuclear scope set u' stand in the relation denoted by the corresponding static determiner \mathbf{DET} .

The three distinct updates in (12) are conjoined and, as (13) below shows, dynamic conjunction “;” is interpreted as relation composition. Note the difference between dynamic conjunction, which is an abbreviation, and the official, classical, static conjunction “ \wedge ”.

13. $D; D' := \lambda I_{st}. \lambda J_{st}. \exists H_{st} (DIH \wedge D'HJ)$, where D, D' are DRS's (type \mathbf{t}).

To formally spell out the PCDRT meaning for generalized determiners in (12) above and the meanings for indefinites and pronouns, we need: (i) two operators over plural info states, namely a selective maximization operator $\mathbf{max}^u(\dots)$ and a selective distributivity operator $\langle u \rangle(\dots)$ and (ii) a notion of structured inclusion $u' \sqsubseteq u$ that requires the subset to preserve the quantificational dependencies, i.e. the structure, associated with the individuals in the superset.

3.1 Structured Inclusion

Let us start with the notion of structured subset. Recall that plural info states store both values (in the columns of the matrix) and structure (in the rows of the matrix). Requiring one dref u_3 to simply be a value-subset of another dref u_1 relative to an info state I is defined as shown in (14) below; for example, the leftmost u_3 column in the table below satisfies the condition $u_3 \subseteq u_1$ because $u_3I = \{x_1, x_2, x_3\} \subseteq u_1I = \{x_1, x_2, x_3, x_4\}$. Condition (14) requires only *value inclusion* and disregards structure completely. The correlation between the u_1 -individuals and the u_2 -individuals, i.e. the relation $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)\}$, is lost in going from the u_1 -superset to the u_3 -subset: as far as u_3 and u_2 are concerned, x_1 is still correlated with y_1 , but it is now also correlated with y_3 ; moreover, x_2 is now correlated with y_4 (not y_2) and x_3 with y_2 (not y_3).

14. $u_3 \subseteq u_1 := \lambda I_{st}. u_3I \subseteq u_1I$

15. $u_3 \subseteq\subseteq u_1 := \lambda I_{st}. \forall i_s \in I (u_3i = u_1i \vee u_3i = \#)$

Info State I	u_1	u_2	$u_3 (u_3 \subseteq u_1, u_3 \not\subseteq u_1)$	$u_3 (u_3 \subseteq u_1)$
i_1	x_1	y_1	x_1	x_1
i_2	x_2	y_2	x_3	x_2
i_3	x_3	y_3	x_1	#
i_4	x_4	y_4	x_2	x_4

If we use the notion of value-only subset in (14), we make incorrect empirical predictions. Consider, for example, the discourse in (16) below, where u_1 stores the set of conventions⁶ and u_2 stores the set of corresponding girls; furthermore, assume that *every* ^{u_1} *convention* takes scope over *a* ^{u_2} *girl* and that the correlation between u_1 -conventions and courted u_2 -girls is the one represented in the table above. Intuitively, the adverb *usually* in (16b) is anaphoric to the set of conventions introduced in (16a) and (16b) is interpreted as asserting that, at most conventions, the girl courted by Harvey *at that convention* comes to the banquet with him. The leftmost dref u_3 in the table above does store most u_1 -conventions (three out of four), but it does not preserve the correlation between u_1 -conventions and u_2 -girls established in sentence (16a).

We obtain similarly incorrect results for donkey sentences like the one in (17) below: the restrictor of the quantification introduces a dependency between all the donkey-owning u_1 -farmers and the u_2 -donkeys that they own; the nuclear scope set u_3 needs to contain most u_1 -farmers, but in such a way that the correlated u_2 -donkeys remain the same. That is, the nuclear scope set contains a *most*-subset of donkey-owning farmers that beat *their respective donkey(s)*. The notion of value-only inclusion in (14) is, yet again, inadequate.

16. **a.** Harvey courts a ^{u_2} girl at every ^{u_1} convention. **b.** She _{u_2} usually ^{$u_3 \subseteq u_1$} comes to the banquet with him.
17. Most ^{$u_1, u_3 \subseteq u_1$} farmers who own a ^{u_2} donkey beat it _{u_2} .

Thus, to capture the intra- and cross-sentential interaction between anaphora and quantification, we need the notion of *structured inclusion* defined in (15) above, whereby we go from a superset to a subset by discarding rows in the matrix. We are therefore guaranteed that the subset will contain *only* the dependencies associated with the superset (but not necessarily *all* dependencies – see below). To implement this, I follow [24] and introduce a dummy/exception individual # that is used as a tag for the cells in the matrix that should be discarded in order to obtain a structured subset u_3 of a superset u_1 – as shown by the rightmost u_3 column in the table above.

Unlike [24], I do not take the introduction of the dummy individual # to require making the underlying logic partial, i.e. I will not take a lexical relation that has # as one of its arguments, e.g. *girl*(#) or *courted_at*(#, x_1), to be undefined. I will just require the dummy individual # to make any lexical relation

⁶ In the case of a successful *every*-quantification, the restrictor and nuclear scope sets are identical with respect to both value and structure, so we can safely identify them.

false⁷. This allows us to keep the underlying type logic classical while making sure that we do not accidentally introduce $\#$ and inadvertently discard a cell when we evaluate another lexical relation later on. Thus, lexical relations (i.e., atomic conditions) are interpreted *distributively* relative to the non-dummy sub-state of the input plural info state I , as shown in (19) below.

18. $I_{u_1 \neq \#, \dots, u_n \neq \#} := \{i_s \in I : u_1 i \neq \# \wedge \dots \wedge u_n i \neq \#\}$
 19. $R\{u_1, \dots, u_n\} := \lambda I_{st}. I_{u_1 \neq \#, \dots, u_n \neq \#} \neq \emptyset \wedge \forall i_s \in I_{u_1 \neq \#, \dots, u_n \neq \#} (R(u_1 i, \dots, u_n i))$

The notion of structured inclusion \subseteq in (15) above ensures that the subset inherits *only* the superset structure – but we also need it to inherit *all* the superset structure, which we achieve by means of the second conjunct in definition (20) below. This conjunct is needed (among others) to account for the donkey sentence in (7) above, which is interpreted as talking about *every* slave owned by any given person, i.e. the nuclear scope set, which is a *most*-subset of the restrictor set, needs to inherit *all* the superset structure (each slave owner in the nuclear scope set needs to be associated with *every* slave that s/he owned).

20. $u' \sqsubseteq u := \lambda I_{st}. (u' \subseteq u)I \wedge \forall i_s \in I (u i \in u' I_{u' \neq \#} \rightarrow u i = u' i)$

3.2 Maximization and Distributivity

We turn now to the maximization and distributivity operators \mathbf{max}^u and \mathbf{dist}_u , which are defined in the spirit – but not the letter – of the corresponding operators in [24]. Selective maximization and selective distributivity together enable us to dynamize λ -abstraction over both values (individuals, i.e. quantifier domains) and structure (quantificational dependencies); that is, \mathbf{max}^u and \mathbf{dist}_u enable us to extract and store the restrictor and nuclear scope structured sets needed to define dynamic generalized quantification.

Consider the definition of \mathbf{max}^u in (21) below first: the first conjunct introduces u as a new dref, i.e. $[u]$, and makes sure that each individual in uJ satisfies D , i.e. we store *only* individuals that satisfy D . The second conjunct enforces the maximality requirement: any other set uK obtained by a similar procedure, i.e. any other set of individuals that satisfies D , is included in uJ – that is, uJ stores *all* individuals that satisfy D .

21. $\mathbf{max}^u(D) := \lambda I_{st}. \lambda J_{st}. ([u]; D)IJ \wedge \forall K_{st} (([u]; D)IK \rightarrow uK_{u \neq \#} \subseteq uJ_{u \neq \#})$
 22. $\mathbf{max}^{u' \sqsubseteq u}(D) := \mathbf{max}^{u'}([u' \sqsubseteq u]; D)$
 23. $I_{u=x} = \{i_s \in I : u i = x\}$
 24. $\mathbf{dist}_u(D) := \lambda I_{st}. \lambda J_{st}. uI = uJ \wedge \forall x_e \in uI (D I_{u=x} J_{u=x})$ ⁸

⁷ We ensure that any lexical relation R of arity n (i.e. of type $e^n t$, defined recursively as in [16]: 157-158, i.e. as $e^0 t := t$ and $e^{m+1} t := e(e^m t)$) yields falsity whenever $\#$ is one of its arguments by letting $R \subseteq (D_e^{\mathfrak{M}} \setminus \{\#\})^n$.

⁸ In general, $\mathbf{dist}_{u_1, \dots, u_n}(D)$ is defined as:

$$\lambda I_{st}. \lambda J_{st}. \forall x_1 \dots \forall x_n (I_{u_1=x_1, \dots, u_n=x_n} \neq \emptyset \leftrightarrow J_{u_1=x_1, \dots, u_n=x_n} \neq \emptyset) \wedge \forall x_1 \dots \forall x_n (I_{u_1=x_1, \dots, u_n=x_n} \neq \emptyset \rightarrow D I_{u_1=x_1, \dots, u_n=x_n} J_{u_1=x_1, \dots, u_n=x_n}).$$

Definition (24) states that updating an info state I with a DRS D *distributively* over a dref u means: (i) generating the u -partition of I , i.e. $\{I_{u=x} : x \in uI\}$, (ii) updating each cell $I_{u=x}$ in the partition with the DRS D and (iii) taking the union of the resulting output info states. The first conjunct in (24) is required to ensure that there is a bijection between the partition induced by the dref u over the input state I and the one induced over the output state J ; without this requirement, we could introduce arbitrary new values for u in the output state J , i.e. arbitrary new partition cells⁹. The second conjunct is the one that actually defines the distributive update: the DRS D relates every partition cell in the input state I to the corresponding partition cell in the output state J .

3.3 Generalized Quantifiers and Indefinites

The PCDRT meanings for generalized determiners and weak/strong indefinites are provided in (28), (29) and (30) below¹⁰.

25. ${}_u(D) := \lambda I_{st}.\lambda J_{st}. I_{u=\#} = J_{u=\#} \wedge I_{u\neq\#} \neq \emptyset \wedge \mathbf{dist}_u(D)I_{u\neq\#}J_{u\neq\#}$
26. $\langle u \rangle(D) := \lambda I_{st}.\lambda J_{st}. I_{u=\#} = J_{u=\#} \wedge (I_{u\neq\#} = \emptyset \rightarrow I = J) \wedge (I_{u\neq\#} \neq \emptyset \rightarrow \mathbf{dist}_u(D)I_{u\neq\#}J_{u\neq\#})$
27. $\mathbf{DET}\{u, u'\} := \lambda I_{st}. \mathbf{DET}(uI_{u\neq\#}, u'I_{u'\neq\#})$, where \mathbf{DET} is a static det.
28. $\mathbf{det}^{u, u' \sqsubseteq u} \rightsquigarrow \lambda P_{\mathbf{et}}.\lambda P'_{\mathbf{et}}. \mathbf{max}^u(\langle u \rangle(P(u))); \mathbf{max}^{u' \sqsubseteq u}(\langle u' \rangle(P'(u'))); [\mathbf{DET}\{u, u'\}]$
29. $a^{\mathbf{wk}:u} \rightsquigarrow \lambda P_{\mathbf{et}}.\lambda P'_{\mathbf{et}}. [u]; {}_u(P(u)); {}_u(P'(u))$
30. $a^{\mathbf{str}:u} \rightsquigarrow \lambda P_{\mathbf{et}}.\lambda P'_{\mathbf{et}}. \mathbf{max}^u({}_u(P(u)); {}_u(P'(u)))$

The \mathbf{max} -based definition of generalized quantification correctly predicts that anaphora to restrictor/nuclear scope sets is always anaphora to *maximal* sets, i.e. E-type anaphora¹¹. That is, the maximality of anaphora to quantifier sets is an automatic consequence of the fact that we independently need \mathbf{max} -operators to formulate truth-conditionally correct dynamic meanings for quantifiers. This is one of the major results in [24], preserved in PCDRT.

The existential commitment associated with dref introduction is built into (i) the definition of lexical relations in (19) above (i.e. $I_{u_1\neq\#, \dots, u_n\neq\#} \neq \emptyset$) and (ii) the definition of the operator ${}_u(\dots)$ in (25) above (i.e. $I_{u\neq\#} \neq \emptyset$)¹².

There is, however, no such existential commitment in the definition of generalized determiners $\mathbf{det}^{u, u' \sqsubseteq u}$, which employs the distributivity operator $\langle u \rangle(\dots)$

⁹ [17]: 87 was the first to observe that we need to add the first conjunct in (24) to the original definition of distributivity in (18), [24]: 145.

¹⁰ See [2] for the justification of the account of weak/strong donkey ambiguities in terms of weak/strong indefinite articles and see [4] for a detailed investigation of various kinds of indefinites within a related dynamic framework.

¹¹ Recall the Evans examples *Few senators admire Kennedy and they are very junior.* and *Harry bought some sheep. Bill vaccinated them.* and (9), (10) and (11) above.

¹² We need these non-emptiness requirements because the pair $(\emptyset_{st}, \emptyset_{st})$ belongs, on the one hand, to the denotation of $[u]$ for any dref u (see (5) above) and, on the other hand, to the denotation of $\mathbf{dist}_u(D)$ for any dref u and DRS D (see (24) above).

defined in (26) above. The use of $\langle u \rangle(\dots)$ enables us to capture the meaning of both upward and downward monotonic quantifiers by means of the same definition. The problem posed by downward monotonic quantifiers is that their nuclear scope set can or has to be empty; for example, after a successful update with a $\mathbf{no}^{u,u'} \sqsubseteq^u$ quantification, the nuclear scope set u' is necessarily empty, i.e. the dref u' will always store only the dummy individual $\#$ relative to the output info state; this, in turn, entails that no lexical relation in the nuclear scope DRS that has u' as an argument can be satisfied. The second conjunct in the definition of $\langle u \rangle(\dots)$, i.e. $I_{u\neq\#} = \emptyset \rightarrow I = J$, enables us to resolve the conflict between the emptiness requirement enforced by a \mathbf{no} -quantification and the non-emptiness requirement enforced by lexical relations^{13, 14}.

3.4 Singular Number Morphology

Let us turn now to the last component needed for the account of discourses (1) and (2), namely the representation of singular pronouns. Their PCDRT translation, provided in (32) below, has the expected Montagovian form: it is the distributive type-lift of the dref u they are anaphoric to, with the addition of the condition **unique** $\{u\}$. The condition is contributed by singular number morphology and requires uniqueness of the non-dummy value of the dref u relative to the current plural info state I . In contrast, plural pronouns do not require uniqueness, as shown in (33) below. The meanings for singular and plural anaphoric definite articles in (34) and (35) below (we need them to interpret the anaphoric DP *the girl* in (2c) above among others) exhibit the same kind of unique/non-unique contrast as the meanings for singular and plural pronouns.

The uniqueness enforced by the condition **unique** $\{u\}$ is *weak* in the sense that it is relativized to the current plural info state. However, we can require *strong* uniqueness, i.e. uniqueness relative to the entire model, by combining the \mathbf{max}^u operator and the **unique** $\{u\}$ condition, as shown by the Russellian, non-anaphoric meaning for definite descriptions provided in (36) below. This alternative meaning for definite articles, which requires existence and strong uniqueness, is needed to interpret the DP *the banquet* in (2b) above.

31. unique $\{u\} := \lambda I_{st}. I_{u\neq\#} \neq \emptyset \wedge \forall i_s \in I_{u\neq\#} \forall i'_s \in I_{u\neq\#} (ui = ui')$

¹³ Even if definition (28) allows for empty restrictor and nuclear scope sets, we can still capture the fact that subsequent anaphora to such empty sets is infelicitous (e.g. anaphora to the nuclear scope sets in (10) and (11) above) because pronominal meanings are defined in terms of the operator ${}_u(\dots)$ – see, for example, the PCDRT translations for *she* and *they* in (32) and (33) below.

¹⁴ The fact that the second conjunct in (26) requires the identity of the input and output states I and J correctly predicts that anaphora to both empty restrictor/nuclear scope sets and indefinites in restrictor/nuclear scope DRS's associated with such empty sets is infelicitous. For example, the nuclear scope DRS of a successful $\mathbf{no}^{u,u'} \sqsubseteq^u$ -quantification, i.e. $\mathbf{max}^{u' \sqsubseteq^u}(\langle u' \rangle(P'(u')))$, will always be a test; hence, we correctly predict that anaphora to any indefinites in the nuclear scope of a \mathbf{no} -quantification is infelicitous, e.g. *Harvey courts a'' girl at no^{u,u'} convention. #She_{u''} / They_{u''} is/are very pretty.*

32. $she_u \rightsquigarrow \lambda P_{et}. [\mathbf{unique}\{u\}]; {}_u(P(u))$
 33. $they_u \rightsquigarrow \lambda P_{et}. {}_u(P(u))$
 34. $the_sg_u \rightsquigarrow \lambda P_{et}. \lambda P'_{et}. [\mathbf{unique}\{u\}]; {}_u(P(u)); {}_u(P'(u))$
 35. $the_pl_u \rightsquigarrow \lambda P_{et}. \lambda P'_{et}. {}_u(P(u)); {}_u(P'(u))$
 36. $the_sg^u \rightsquigarrow \lambda P_{et}. \lambda P'_{et}. \mathbf{max}^u({}_u(P(u)); [\mathbf{unique}\{u\}]; {}_u(P'(u)))$

The PCDRT translation for proper names and the definitions of dynamic negation and truth are provided in (37), (38) and (39) below. I take the default context of interpretation for all discourses, i.e. the default input info state relative to which a DRS is true/false, to be the singleton info state $\{i_\#\}$, where $i_\#$ is the ‘assignment’ that stores the dummy individual $\#$ relative to all individual dref’s. Finally, the abbreviations in (40) and (41) below and the equivalences in (42) and (43) enable us to simplify – and, therefore, enhance the readability of – some very common PCDRT representations.

37. $Harvey^u \rightsquigarrow \lambda P_{et}. [u|u \in Harvey]; {}_u(P(u))$,
 where $Harvey := \lambda i_s. harvey_e$ (i.e. *Harvey* is a ‘rigid’ individual dref).
 38. $\sim D := \lambda I_{st}. I \neq \emptyset \wedge \forall H_{st} \neq \emptyset (H \subseteq I \rightarrow \neg \exists K_{st} (DHK))$
 39. A DRS D of type **t** is *true* with respect to an input state I_{st} iff $\exists J_{st} (DIJ)$.
 40. ${}_u(C) := \lambda I_{st}. I_{u \neq \#} \neq \emptyset \wedge \forall x_e \in u I_{u \neq \#} (CI_{u=x})$,
 where C is a condition (type $(st)t$).
 41. ${}_u(u_1, \dots, u_n) := \lambda I_{st}. \lambda J_{st}. I_{u=\#} = J_{u=\#} \wedge I_{u \neq \#} [u_1, \dots, u_n] J_{u \neq \#}$,
 where $u \notin \{u_1, \dots, u_n\}$ and $[u_1, \dots, u_n] := [u_1]; \dots; [u_n]$.
 42. ${}_u([C_1, \dots, C_m]) = [{}_u(C_1), \dots, {}_u(C_m)]$
 43. ${}_u([u_1, \dots, u_n | C_1, \dots, C_m]) = [{}_u(u_1, \dots, u_n) | {}_u(C_1), \dots, {}_u(C_m)]$

4 Quantificational Subordination in PCDRT

This section presents the PCDRT analysis of discourses (1) and (2). We start with the two possible quantifier scopings for the discourse-initial sentence (1a/2a). For simplicity, I will assume that the two scopings are due to the two different lexical entries for the ditransitive verb *court_at*, provided in (44) and (45) below¹⁵: *court_at*¹ assigns the indefinite *a girl* wide scope relative to *every convention*, while *court_at*² assigns it narrow scope. I assume that the basic syntactic structure of the sentence is the one given in (46).

44. $court_at^1 \rightsquigarrow \lambda Q'_{(et)t}. \lambda Q''_{(et)t}. \lambda v_e. Q'(\lambda v'_e. Q''(\lambda v''_e. [court_at\{v, v', v''\}]))$
 45. $court_at^2 \rightsquigarrow \lambda Q'_{(et)t}. \lambda Q''_{(et)t}. \lambda v_e. Q''(\lambda v''_e. Q'(\lambda v'_e. [court_at\{v, v', v''\}]))$
 46. $Harvey [[court_at^{1/2} [a\ girl]] [every\ convention]]$

Turning to the meaning of the quantifier *every convention*, note that we can safely identify the restrictor and nuclear scope dref’s u and u' of any *every* ^{$u, u' \sqsubseteq u$} -quantification: the definition in (28) above entails that, if J is an arbitrary output

¹⁵ But it should be clear that PCDRT is compatible with any of the quantifier scoping mechanisms proposed in the literature; for a version of PCDRT that incorporates Quantifying-In / Quantifier Raising, see [2].

state of a successful *every* ^{$u, u' \sqsubseteq u$} -quantification, u and u' have to be identical with respect to both value and structure, i.e. $\forall j_s \in J(uj = u'j)$. We can therefore assume that *every* contributes only one dref, as shown in (47) below. I will also assume that the restrictor set of the *every* ^{u_1} -quantification is non-empty, so we can safely replace the operator $\langle_{u_1}(\dots)$ with the operator $_{u_1}(\dots)$. The PCDRT representations of the two quantifier scopings for sentence (1a/2a) are provided in (50) and (51) below. For simplicity, I take the translation of the proper name *Harvey* to be $\lambda P_{\text{et}}. P(\text{Harvey})$ instead of the more complex one in (37) above¹⁶.

47. *every* ^{u_1} $\rightsquigarrow \lambda P_{\text{et}}. \lambda P'_{\text{et}}. \mathbf{max}^{u_1}([_{u_1}(P(u_1))]; \text{ }_{u_1}(P'(u_1)))$
 48. *every* ^{u_1} *convention* $\rightsquigarrow \lambda P_{\text{et}}. \mathbf{max}^{u_1}([_{\text{convention}}\{u_1\}]; \text{ }_{u_1}(P(u_1)))$
 49. *a* ^{$\text{wk}; u_2$} *girl* $\rightsquigarrow \lambda P_{\text{et}}. [_{u_2} | \text{girl}\{u_2\}]; \text{ }_{u_2}(P(u_2))$
 50. *a* ^{$\text{wk}; u_2$} *girl* $>> \text{every}^{u_1}$ *convention* $\rightsquigarrow [_{u_2} | \text{girl}\{u_2\}];$
 $\text{ }_{u_2}(\mathbf{max}^{u_1}([_{\text{convention}}\{u_1\}]); [_{u_2}(\text{court_at}\{\text{Harvey}, u_2, u_1\})])$
 51. *every* ^{u_1} *convention* $>> \text{a}^{\text{wk}; u_2}$ *girl* $\rightsquigarrow \mathbf{max}^{u_1}([_{\text{convention}}\{u_1\}]);$
 $[_{u_1}(u_2) | \text{ }_{u_1}(\text{girl}\{u_2\}), \text{ }_{u_1}(\text{court_at}\{\text{Harvey}, u_2, u_1\})])$

The representation in (50) updates the default input info state $\{i_{\#}\}$ as follows. First, we introduce some non-empty (i.e. non-dummy) set of individuals relative to the dref u_2 . Then, we test that each u_2 -individual is a girl. Then, relative to each u_2 -individual, we introduce the set of all conventions and store it in the dref u_1 . Finally, we test that, for each u_2 -girl and for each of the corresponding u_1 -conventions (i.e., in this case: for every convention), Harvey courted her at the convention. The output info state obtained after updating with (50) contains a non-empty set of u_2 -girls that where courted by Harvey at every convention and, relative to each u_2 -girl, u_1 stores the set of all conventions.

The representation in (51) updates the default input info state $\{i_{\#}\}$ as follows. First, we introduce the set of all conventions relative to the dref u_1 . Then, for each u_1 -convention, we introduce a u_2 -set of individuals. Finally we test that, for each u_1 -convention, each of the corresponding u_2 -individuals are girls and are such that Harvey courted them at the convention under consideration. The output info state obtained after updating with (51) stores the set of all conventions under the dref u_1 and, relative to each u_1 -convention, the dref u_2 stores a non-empty set of girls (possibly different from convention to convention) that Harvey courted at that particular convention.

We can now see how sentence (1b) – in particular, the singular number morphology on the pronoun *she* _{u_2} – forces the ‘wide-scope indefinite’ reading: the condition **unique** _{$\{u_2\}$} (see (52) and (53) below) effectively conflates the two scopings by requiring the set of u_2 -girls obtained after updating with (50) or (51) to be a singleton. This requirement leaves the truth-conditions derived on the basis of (50) untouched, but makes the truth-conditions associated with (51) strictly stronger.

52. *she* _{u_2} $\rightsquigarrow \lambda P_{\text{et}}. [\mathbf{unique}\{u_2\}]; \text{ }_{u_2}(P(u_2))$
 53. **[unique** _{$\{u_2\}$} , *very-pretty* _{$\{u_2\}$}]

¹⁶ The reader can check that this simplification does not affect the PCDRT truth-conditions for the discourses under consideration.

In contrast, sentence (2b) contains the adverb of quantification *always*_{u₁}, which can take scope above or below the singular pronoun *she*_{u₂}. In the former case, the *u₂*-uniqueness requirement is weakened (i.e., in a sense, neutralized) by being relativized to *u₁*-conventions. As shown in (54) below, I take the meaning of *always*_{u₁} to be a universal quantification over an anaphorically retrieved restrictor, i.e. over the nuclear scope set introduced by the quantifier *every*^{u₁} *convention* in the preceding sentence. Since *always* is basically interpreted as *every* (modulo the anaphorically retrieved restrictor), its translation is parallel to the translation for *every* in (47) above. The general format for the interpretation of quantifiers that anaphorically retrieve their restrictor set is provided in (55).

$$54. \textit{always}_{u_1} \rightsquigarrow \lambda P_{\text{et. } u_1} (P(u_1))$$

$$55. \textit{det}'_u \sqsubseteq^u \rightsquigarrow \lambda P_{\text{et.}} \max^{u' \sqsubseteq^u}_{\langle u' \rangle} (P(u')); [\text{DET}\{u, u'\}]$$

The definite description *the banquet* in (2b) is intuitively a Russellian definite description (see (36) above), which contributes existence and a relativized (i.e. anaphoric) form of uniqueness: we are talking about a *unique* banquet *per convention*. For simplicity, however, I will assume that sentence (2b) contributes a transitive predication of the form *come_with_Harvey_to_the_banquet_of* relating girls and conventions¹⁷, which, as shown in (56) and (57) below, can be translated in two different ways corresponding to the two possible relative scopes of *she*_{u₂} and *always*_{u₁} (that is, the scoping technique is the same as in (44) and (45) above). The translation in (56) gives the pronoun *she*_{u₂} wide scope over the adverb *always*_{u₁}, while the translation in (57) gives the pronoun narrow scope relative to the adverb. The corresponding PCDRT representations, obtained on the basis of the syntactic structure in (58), are provided in (59) and (60) below.

$$56. \textit{come_to_banquet_of}^1 \rightsquigarrow \lambda Q_{(\text{et})t} \cdot \lambda Q'_{(\text{et})t} \cdot Q'(\lambda v'_e \cdot Q(\lambda v_e \cdot [\textit{c.t.b.of}\{v', v\}]))$$

$$57. \textit{come_to_banquet_of}^2 \rightsquigarrow \lambda Q_{(\text{et})t} \cdot \lambda Q'_{(\text{et})t} \cdot Q(\lambda v_e \cdot Q'(\lambda v'_e \cdot [\textit{c.t.b.of}\{v', v\}]))$$

$$58. \textit{she} \text{ } [[\textit{always}] \textit{come_to_banquet_of}^{1/2}]$$

$$59. \textit{she}_{u_2} >> \textit{always}_{u_1} \rightsquigarrow [\text{unique}\{u_2\}, u_2(\textit{c.t.b.of}\{u_2, u_1\})]$$

$$60. \textit{always}_{u_1} >> \textit{she}_{u_2} \rightsquigarrow [u_1(\text{unique}\{u_2\}), u_1(\textit{c.t.b.of}\{u_2, u_1\})]$$

Thus, there are two possible PCDRT representations for sentence (2a) and two possible representations for sentence (2b). Out of the four combinations, three end up requiring the indefinite *a*^{wk:u₂} *girl* to have wide scope relative to *every*^{u₁} *convention*. The fourth combination (51+60), provided in (61) below, encodes the ‘narrow-scope indefinite’ reading that is intuitively available for discourse (2), but not for (1). The PCDRT representation in (61) updates the

¹⁷ The relevant PCDRT translation for the definite article is *the*^{u₃}_{u₁} $\rightsquigarrow \lambda P_{\text{et.}} \lambda P'_{\text{et. } u_1} (\max^{u_3}_{u_3} (P(u_3))); [\text{unique}\{u_3\}]; u_3(P'(u_3))$, which, together with a relational interpretation of the noun *banquet* as *banquet of it*_{u₁} $\rightsquigarrow \lambda v_e \cdot [\textit{banquet}\{v\}, \textit{of}\{v, u_1\}]$, yields the following PCDRT translation for our Russellian definite description with relativized uniqueness: *the*^{u₃}_{u₁} *banquet of it*_{u₁} $\rightsquigarrow \lambda P_{\text{et. } u_1} (\max^{u_3} ([\textit{banquet}\{u_3\}, \textit{of}\{u_3, u_1\}]); [\text{unique}\{u_3\}]; u_3(P(u_3)))$.

default input info state $\{i_\#\}$ as follows: first, we introduce the set of all conventions relative to the dref u_1 , followed by the introduction of a non-empty set of u_2 -individuals relative to each u_1 -convention; the remainder of the representation tests that, for each u_1 -convention, the corresponding u_2 -set is a singleton set whose sole member is a girl that is courted by Harvey at the u_1 -convention under consideration and that comes with him to the banquet of that convention.

61. $\mathbf{max}^{u_1}([convention\{u_1\}]); [_{u_1}(u_2) \mid _{u_1}(girl\{u_2\}),$
 $_{u_1}(court_at\{Harvey, u_2, u_1\}), _{u_1}(\mathbf{unique}\{u_2\}), _{u_1}(c.t.b.of\{u_2, u_1\})]$

Summarizing, PCDRT enables us to formulate a compositional dynamic account of the intra- and cross-sentential interaction between generalized quantifiers, anaphora and number morphology exhibited by the quantificational subordination discourses in (1) and (2) above. The main proposal is that plural info states together with a suitable dynamic reformulation of the independently motivated denotations for generalized determiners and number morphology in static Montague semantics enable us to account for quantificational subordination in terms of structured anaphora to quantifier domains.

5 A Parallel Account of Modal Subordination

In this section, I will briefly indicate how PCDRT can be extended to give a compositional account of modal subordination discourse like the one in (62) below (based on [21]).

62. **a.** a^u wolf might come in. **b.** It_u would attack Harvey first.

Under its most salient interpretation, (62) asserts that, for all the speaker knows, it is possible that a wolf comes in. Moreover, in *any* such epistemic possibility, the wolf attacks Harvey first. Discourse (62) is parallel to discourse (2) above: the interaction between the indefinite a^u *wolf* and the modal *might* on the one hand and the singular pronoun it_u and the modal *would* on the other hand is parallel to the interaction between a^u *girl-every convention* and *she_u-always*.

The addition of another basic type **w** for possible worlds together with dref's p, p' etc. of type sw is almost everything that is needed to account for discourse (62). In the resulting Intensional PCDRT (IP-CDRT) system, the dref's p, p' etc. store sets of possible worlds, i.e. *propositions*, relative to a plural info state, e.g. $pI := \{p_{sw}i_s : i_s \in I_{st}\}$, i.e. pI is the image of the set of 'assignments' I under the function p . The basic IP-CDRT system is very much parallel to the PCDRT system introduced in the previous sections, so I provide only some of the relevant definitions. In particular, the definition of structured inclusion for sets of worlds in (65) below employs a dummy/exception world $\#_w$ ¹⁸ which makes every lexical relation false just as the dummy/exception individual $\#_e$ does.

¹⁸ We can take the dummy world $\#_w$ to be the world where no individual whatsoever exists, hence all the lexical relations are false because a relation between certain individuals obtains at a particular world w only if those individuals exist in w .

63. $R_p\{u_1, \dots, u_n\} :=$
 $\lambda I_{st}. I_{p \neq \#, u_1 \neq \#, \dots, u_n \neq \#} \neq \emptyset \wedge \forall i_s \in I_{p \neq \#, u_1 \neq \#, \dots, u_n \neq \#} (R_{pi}(u_1 i, \dots, u_n i))^{19}$
 64. $[p] := \lambda I_{st}. \lambda J_{st}. \forall i_s \in I(\exists j_s \in J(i[p]j)) \wedge \forall j_s \in J(\exists i_s \in I(i[p]j))$
 65. $p' \sqsubseteq p := \lambda I_{st}. (p' \sqsubseteq p)I \wedge \forall i_s \in I(pi \in p'I_{p' \neq \#} \rightarrow pi = p'i),$
 where $p' \sqsubseteq p := \lambda I_{st}. \forall i_s \in I(p'i = pi \vee p'i = \#).$

In an intensional Fregean/Montagovian framework, the compositional aspect of interpretation is largely determined by the types for the extensions of the ‘saturated’ expressions, i.e. names and sentences, plus the type that enables us to build intensions out of these extensions. Let us abbreviate them as **e**, **t** and **s** respectively. We preserve the dynamic types that PCDRT assigns to the ‘meta-types’ **e** and **t**, i.e. $\mathbf{t} := (st)((st)t)$ and $\mathbf{e} := se$; predictably, IP-CDRT uses possible-word dref’s to build intensions, i.e. $\mathbf{s} := \mathbf{sw}$. Just as generalized determiners in PCDRT relate dynamic properties P, P' etc. of type **et** (see (28) above), modal verbs relate dynamic propositions \mathbb{P}, \mathbb{P}' etc. of type **st**, as shown in (66) below. Moreover, just as a pronoun anaphorically retrieves an individual dref and makes sure that a dynamic property holds of that dref (see the meaning for *she* in (32) above), the indicative verbal mood anaphorically retrieves p^* , which is the designated dref for the actual world, and makes sure that a dynamic proposition holds of p^* , as shown in (67) below.

Finally, just as the quantifier *always* in (2b) is anaphoric to the nuclear scope set introduced by *every convention* in (2a), the modal quantifier *would* in (62b) is anaphoric to the nuclear scope set introduced by *might* in (62a). The general format for the translation of anaphoric modal quantifiers is provided in (68) below (cf. the translation of anaphoric determiners in (55) above).

66. $if^p + modal_{\mu, \omega}^{p' \sqsubseteq p} \rightsquigarrow \lambda \mathbb{P}_{st}. \lambda \mathbb{P}'_{st}. \lambda q_s.$
 $\mathbf{max}^p(\langle p \rangle(\mathbb{P}(p))); \mathbf{max}^{p' \sqsubseteq p}(\langle p' \rangle(\mathbb{P}'(p'))); [\mathbf{MODAL}_{q, \mu, \omega}\{p, p'\}]$
 67. $indicative_{p^*} \rightsquigarrow \lambda \mathbb{P}_{st}. [\mathbf{unique}\{p^*\}]; p^*(\mathbb{P}(p^*))$
 68. $modal_{\mu, \omega}^{p' \sqsubseteq p} \rightsquigarrow \lambda \mathbb{P}_{st}. \lambda q_s. \mathbf{max}^{p' \sqsubseteq p}(\langle p' \rangle(\mathbb{P}'(p'))); [\mathbf{MODAL}_{q, \mu, \omega}\{p, p'\}]$

This concludes our brief survey of IP-CDRT. It is hopefully clear by now that IP-CDRT enables us to provide an analysis of the modal subordination discourse in (62) above that is parallel to the analysis of the quantificational subordination discourse in (2); see [2] for a detailed account showing that the parallels between anaphora and quantification in the individual and modal domains are systematically captured in IP-CDRT.

6 Comparison with Previous Approaches

PCDRT differs from most previous dynamic approaches in at least three respects. The first difference is conceptual: PCDRT captures the idea that reference to structure is as important as reference to value and that the two should be treated

¹⁹ The definition of atomic conditions in (63) assumes static lexical relations $R_w(x_1, \dots, x_n)$ of the expected intensional type $e^n(\mathbf{wt})$, where $e^n\tau$ (for any type τ) is defined as: $e^0\tau := \tau$ and $e^{m+1}\tau := e(e^m\tau)$.

in parallel. This is primarily encoded in the definition of new dref introduction in (5) above, which differs from the corresponding definitions in [24], [11] and [17] (among others) with respect to the treatment of discourse reference to structure.

The second difference is empirical: the motivation for plural information states is provided by several distinct kinds of phenomena, including singular intra- and cross-sentential individual-level anaphora and modal anaphora and subordination, in contrast to the previous literature (e.g. [24], [11] and [17]), which relies mostly on plural individual-level anaphora (but see [25] for an analysis of questions and modal subordination in a related dynamic system). Consequently, the empirical coverage of (Intensional) PCDRT is correspondingly broader.

Finally, from a formal point of view, PCDRT accomplishes two non-trivial goals for the first time. On the one hand, it is not obvious how to recast van den Berg's Dynamic Plural Logic in classical type logic, given that the former logic is partial and conflates discourse-level plurality (i.e. the use of plural information states) and domain-level plurality (i.e. non-atomic individuals)²⁰.

On the other hand, Intensional PCDRT – which builds on and unifies [14]/[10], [16], [24] and [23] – is, to my knowledge, the first dynamic framework that systematically and explicitly captures the anaphoric and quantificational parallels between the individual and modal domains while, at the same time, keeping the underlying logic classical and preserving the Montagovian approach to compositionality²¹.

References

1. Bittner, M.: Topical Referents for Individuals and Possibilities. In: Hastings, R. et al. (eds.) *Proceedings of SALT 11, CLC*, Cornell University, pp. 36-55 (2001)
2. Brasoveanu, A.: *Structured Nominal and Modal Reference*. PhD dissertation, Rutgers University (2007a)
3. Brasoveanu, A.: Donkey Pluralities: Plural Information States vs. Non-atomic Individuals. In: *The Proceedings of Sinn und Bedeutung 11* (to appear)
4. Farkas, D.F.: Varieties of Indefinites. In: Jackson, B. (ed.) *The Proceedings of SALT 12, CLC*, Cornell University, pp. 59-84 (2002)
5. Frank, A.: *Context Dependence in Modal Constructions*. PhD dissertation, Stuttgart University (1996)
6. Gallin, D.: *Intensional and Higher-Order Modal Logic with Applications to Montague Semantics*. North-Holland Mathematics Studies, Amsterdam (1975)
7. Geurts, B.: *Presuppositions and Pronouns*. Elsevier, Amsterdam (1999)
8. Kamp, H.: A theory of truth and semantic representation. In: Groenendijk, J., Janssen, T., Stokhof, M. (eds.) *Formal Methods in the Study of Language*. Part 1, pp. 277-322. Mathematical Center, Amsterdam (1981)
9. Karttunen, L.: Discourse Referents. In: McCawley, J.D. (ed.) *Syntax and Semantics* Vol. 7, pp. 363-385. Academic Press, New York (1976)
10. Kratzer, A.: The Notional Category of Modality. In: Eikmeyer, H.J., Rieser, H. (eds.) *Words, Worlds, and Contexts*, pp. 38-74. Walter de Gruyter, Berlin (1981)

²⁰ See [3] for more discussion of discourse-level vs. domain-level plurality.

²¹ See [2] for a more detailed comparison with alternative accounts, e.g. Skolem function based approaches and dynamic approaches that employ discourse relations (e.g. [26]).

11. Krifka, M.: Parametric Sum Individuals for Plural Anaphora. *Linguistics and Philosophy* 19, 555–598 (1996)
12. Heim, I.: The Semantics of Definite and Indefinite Noun Phrases. PhD dissertation, UMass Amherst (1982)
13. Heim, I.: E-Type Pronouns and Donkey Anaphora. *Linguistics and Philosophy* 13, 137–177 (1990)
14. Lewis, D.: Counterfactuals. Harvard University Press (1973)
15. Lewis, D.: Adverbs of Quantification. In: Keenan, E. (ed.) *Formal Semantics of Natural Language*, pp. 3–15. Cambridge University Press, Cambridge (1975)
16. Muskens, R.: Combining Montague Semantics and Discourse Representation. *Linguistics and Philosophy* 19, 143–186 (1996)
17. Nouwen, R.: Plural Pronominal Anaphora in Context. PhD Dissertation, University of Utrecht (2003)
18. Partee, B.: Some Structural Analogies between Tenses and Pronouns in English. *Journal of Philosophy* 70, 601–609 (1973)
19. Partee, B.: Nominal and Temporal Anaphora. *Linguistics and Philosophy* 7, 243–286 (1984)
20. Pelletier, F.J., Schubert, L.K.: Generically Speaking or Using Discourse Representation Theory to Interpret Generics. In: Chierchia, G., Partee, B.H., Turner, R. (eds.) *Properties, Types, and Meanings*, Vol. 2, pp. 193–268. Kluwer, Dordrecht (1989)
21. Roberts, C.: Modal Subordination, Anaphora and Distributivity. PhD dissertation, UMass Amherst (1987)
22. Schlenker, P.: Ontological Symmetry in Language: A Brief Manifesto. In: *Mind & Language* (to appear)
23. Stone, M.: Reference to Possible Worlds. RuCCS Report 49, Rutgers University, New Brunswick (1999)
24. van den Berg, M.: Some Aspects of the Internal Structure of Discourse. PhD dissertation, University of Amsterdam (1996)
25. van Rooy, R.: Modal subordination in Questions. In: Hulstijn, J., Nijholt, A. (eds.) *The Proceedings of Twendial 1998*, pp. 237–248 (1998)
26. Wang, L., McCready, E., Asher, N.: Information Dependency in Quantificational Subordination. In: Turner, K., von Heusinger, K. (eds.) *Where Semantics Meets Pragmatics*, Elsevier, Amsterdam (2006)

On Principal Types of BCK- λ -Terms

Sabine Broda and Luís Damas

DCC-FCUP & LIACC
Universidade do Porto, Portugal

Abstract. Condensed BCK-logic, i.e. the set of BCK-theorems provable by the condensed detachment rule of Carew Meredith, has been shown to be exactly the set of principal types of BCK- λ -terms. In 1993 Sachio Hirokawa gave a characterization of the set of principal types of BCK- λ -terms in β -normal form based on a relevance relation that he defined between the type variables in a type. We define a symmetric notion of this and call it dependence relation. Then, using the notion of β_S -reduction introduced by de Groote, we obtain a characterization of the complete set of principal types of BCK- λ -terms.

Keywords: Typed lambda-calculus; principal types; condensed BCK-logic.

1 Introduction

A λ -term M is a BCK- λ -term when in every subterm of M each variable occurs free at most once. Every BCK-term is typable, cf. [4], and the type-schemes of BCK-terms form the implicational fragment of BCK-logic. Both the BCK- λ -calculus as well as BCK-logic have been a major topic of study over the years. Since BCK-terms normalize in linear time, they are of interest to several areas of computation, such as functional programming languages or compilation. In [6] one can find a summary of the main properties and an extensive list of references to work performed in this area.

Condensed BCK-logic, i.e. the set of BCK-theorems provable by the condensed detachment rule of Carew Meredith, cf. [5], is identical to the set of principal types of closed BCK- λ -terms. Sachio Hirokawa presented in [7] a method of deciding if a type σ is a principal type for a given BCK-term M in β -normal form. An algorithm that decides if there exists a BCK-term M in β -normal form with principal type σ and that in the affirmative case actually constructs M , can be found in [1]. A very neat characterization of the set of principal types of closed BCK- λ -terms in β -normal form has been given in [8], also by Hirokawa. Actually, his result solves, as a corollary for the particular case in which type variables occur exactly twice, the corresponding problem for the BCI- λ -calculus. Furthermore, the subject-expansion property of the BCI-calculus has as a consequence that the principal types of BCI- λ -terms in β -normal form are exactly the principal types of all BCI- λ -terms. Thus Hirokawa's characterization, restricted to types in which type variables occur twice, is also a characterization of the

complete set of principal types of BCI-terms. The subject-expansion property does not hold in the BCK- λ -calculus, which makes the general problem, i.e. not restricted to β -normal forms, substantially more complicated, contrary to the BCI-fragment where both cases are identical. In the present paper we give a characterization of the complete set of principal types of closed BCK- λ -terms, and consequently of the set of theorems of condensed BCK-logic.

We recall some results by Hirokawa and de Groote in Sect.2 that we will need in the remainder of the paper. In Sect.3 we identify a restricted class of well structured BCK- λ -terms whose principal types are still the principal types of all BCK- λ -terms. A principal type algorithm for the terms in this class is given in Sect.4 and the simplicity of the algorithm enables us then to identify several properties of BCK-principal types. In Sect.5 we relate the structure of the terms in the restricted class with the structure of the relevance graph of their types. The relevance relation defined between the type variables in a type is the one given by Hirokawa in [8]. In Sect.6 we define a somehow symmetric notion of this and call it dependence relation. This relation captures the constraints on initial abstraction sequences in the terms and allows us finally to give a characterization of the principal types of closed BCK- λ -terms in Theorem 2. A link to a document containing the proofs of the presented results, together with some additional lemmata that are necessary, can be found at the homepage of the first author in <http://www.ncc.up.pt/~sbb/publications.html>. Note that the proof of Theorem 2 is constructive in the sense that it provides an algorithm for constructing a closed BCK- λ -term with principal type σ whenever σ matches the conditions in the theorem.

Although this work has been guided by the intuition given by the formula-tree method, cf. [2], we have chosen to present it in an autonomous way in order to simplify the relation with Hirokawa, cf. [8].

2 Background

In this paper we consider, at some times, the simply typed λ -calculus enriched with a distinguished constant \mathbf{K} with principal type $a \rightarrow b \rightarrow a$. A λ -term M is a BCK- λ -term when in every subterm of M each variable occurs free at most once. Note that every BCK-term in the enriched calculus corresponds directly to a BCK-term in the simply typed λ -calculus with the same principal type, obtained by substituting each occurrence of \mathbf{K} by $\lambda xy.x$. On the other hand, every BCK-term in the simply typed λ -calculus is also a BCK-term in the enriched calculus. Consequently, the set of principal types of closed BCK-terms is identical to the set of principal types of closed BCK-terms with the additional constant \mathbf{K} .

Throughout this paper we will use repeatedly three terms M_1 , M_2 and M_3 given in Example 1 to illustrate definitions and results.

Example 1. The terms $M_1 = \lambda xyuv.xy(\lambda z.uv)$, $M_2 = \lambda xyz.\mathbf{K}x(\lambda w.z(w(\lambda t.yt)))$ and $M_3 = \lambda xyr.x(\lambda z.z)(\lambda uv.\mathbf{K}v(\lambda w.u(\lambda s.y(\lambda t.\mathbf{K}w(\lambda o.tro))))$ are all BCK-terms, respectively with principal type

$$\sigma_{M_1} = (a_y \rightarrow (a_z \rightarrow a_u) \rightarrow a_x) \rightarrow a_y \rightarrow (a_v \rightarrow a_u) \rightarrow a_v \rightarrow a_x,$$

$$\sigma_{M_2} = a_x \rightarrow (a_t \rightarrow a_y) \rightarrow (a_w \rightarrow a_z) \rightarrow a_x$$

and $\sigma_{M_3} = ((a_z \rightarrow a_z) \rightarrow (((a_s \rightarrow a_y) \rightarrow a_u) \rightarrow a_v \rightarrow a_v) \rightarrow a_x) \rightarrow (((a_r \rightarrow a_o \rightarrow a_t) \rightarrow a_w) \rightarrow a_y) \rightarrow a_r \rightarrow a_x$.

2.1 The Relevance Relation

In this subsection we recall some notation and results from [8], where Hirokawa provides a characterization of the set of principal type-schemes of closed BCK- λ -terms in β -normal form.

Definition 1. *The occurrence of a type σ in itself is a positive occurrence. If an occurrence of γ is positive (resp. negative) in β , then the occurrence of γ in $\sigma \rightarrow \beta$ is positive (resp. negative) in $\sigma \rightarrow \beta$. If an occurrence of γ is positive (resp. negative) in σ , then the occurrence of γ in $\sigma \rightarrow \beta$ is negative (resp. positive) in $\sigma \rightarrow \beta$. The core of a type σ is the rightmost type variable in σ . We denote it by $\text{core}(\sigma)$.*

Definition 2. *A type is balanced if and only if every type variable occurs at most twice in it and if twice, then with opposite signs. A type variable with exactly one occurrence in a balanced type is called positive if this occurrence is positive and negative otherwise. A type variable with exactly one positive and one negative occurrence in a balanced type is called neutral.*

Example 2. The core of σ_{M_1} , of σ_{M_2} as well as of σ_{M_3} is a_x and all three types are balanced. When we adorn the occurrences of type variables in σ_{M_1} and σ_{M_2} with their polarities, we obtain

$$\sigma_{M_1} = (a_y^+ \rightarrow (a_z^- \rightarrow a_u^+) \rightarrow a_x^-) \rightarrow a_y^- \rightarrow (a_v^+ \rightarrow a_u^-) \rightarrow a_v^- \rightarrow a_x^+$$

and

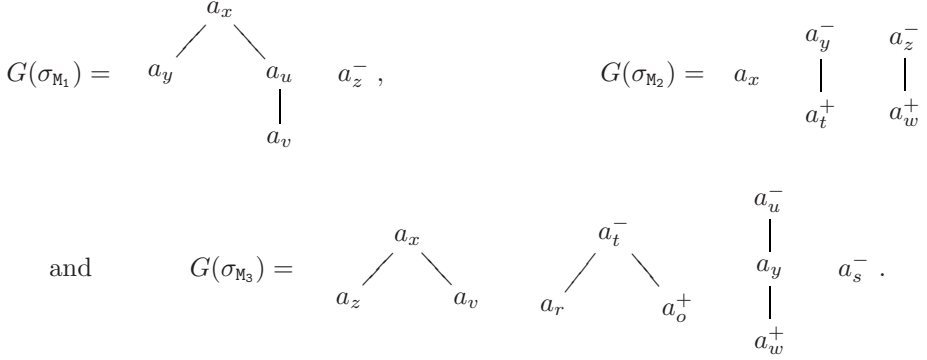
$$\sigma_{M_2} = a_x^- \rightarrow (a_t^+ \rightarrow a_y^-) \rightarrow (a_w^+ \rightarrow a_z^-) \rightarrow a_x^+.$$

To characterize the set of principal types of BCK- λ -terms in normal form, Hirokawa defined a *relevance relation* between type variables in a type.

Definition 3. *Let b and c be type variables in a type σ . Then, $b \succ_\sigma c$ if and only if σ contains a negative occurrence of a subtype of the form $\sigma_1 \rightarrow \dots \rightarrow (\gamma \rightarrow \dots \rightarrow \gamma_k \rightarrow b^+) \rightarrow \dots \rightarrow \sigma_l \rightarrow c^-$. The relevance relation \succ_σ^* is defined as the transitive and reflexive closure of \succ_σ . When $b \succ_\sigma^* c$, one says that b is relevant to c in σ .*

The relevance graph of σ , denoted by $G(\sigma)$, is the directed graph whose nodes are type variables in σ and whose edges are ordered pairs (b, c) of types-variables in σ such that $b \succ_\sigma^ c$.*

Example 3. The relevance graphs of σ_{M_1} , σ_{M_2} and σ_{M_3} are respectively



Note that we draw the graphs upside-down, unlike Hirokawa, and that we marked type variables that are positive (resp. negative) in σ with a positive (resp. negative) sign. Neutral variables are not marked. Note also that a type variable b is relevant to c in σ if and only if $b = c$ or b is a descendant of c in $G(\sigma)$.

The characterization of the set of principal types of BCK- λ -terms in normal form given in [8] is equivalent to the following.

Theorem 1. *A type σ is a principal type of a closed BCK- λ -term in normal form if and only if σ satisfies the following conditions.*

- (a) σ is balanced and there are no positive variables in σ .
- (b)
 - (i) every neutral type variable in σ is relevant to $\text{core}(\sigma)$;
 - (ii) if $b \succ_\sigma^* c$ and $c \succ_\sigma^* b$, then $b = c$;
 - (iii) if $d \succ_\sigma b$ and $d \succ_\sigma c$, then $b = c$;
 - (iv) if b is a negative variable in σ , then there is no variable c in σ such that $c \succ_\sigma b$.
- (c) if b is a neutral variable in σ with an occurrence (positive or negative) in a subtype γ , then $b \succ_\sigma^* \text{core}(\gamma)$.

Example 4. Clearly, the only type in Example 1 satisfying these conditions is σ_{M_1} .

2.2 β_S -Normal Forms

In [3] de Groote defines an I-redex as being a β -redex $(\lambda x.M)N$ such that $x \in FV(M)$ and calls it otherwise, i.e. if $x \notin FV(M)$, a K-redex. He also defines β_I - and β_K -reduction respectively by the contraction rules $(\lambda x.M)N \rightarrow_{\beta_I} M[N/x]$ and $(\lambda x.M)N \rightarrow_{\beta_K} M$. Finally, he defines the structural notion of β_S -reduction by the contraction rule

$$(\lambda x.M)PN \rightarrow_{\beta_S} (\lambda x.MN)P \quad \text{if } x \notin FV(M),$$

and proves the postponement-property of β_K -contractions: “Let M, N, O be λ -terms such that $M \rightarrow_{\beta_K} N$ and $N \rightarrow_{\beta_I \beta_S} O$. Then, there exists a λ -term P

such that $M \rightarrow_{\beta_I\beta_S} P$ and $P \rightarrow_K O$.” De Groote also proved the Church-Rosser property of $\beta_I\beta_S$ -reduction. As a consequence every BCK- λ -term has a unique $\beta_I\beta_S$ -normal form.

3 Standardized $(\beta_I\beta_S + \mathbf{K})$ -Normal Forms

We begin our study by identifying a restricted class of well structured BCK- λ -terms in the enriched calculus, whose principal types are exactly the principal types of all BCK- λ -terms. The following result states that the subject-expansion property holds for $\beta_I\beta_S$ -reduction of BCK- λ -terms.

Lemma 1. *Let M and N be BCK- λ -terms and let (Γ, τ) be a pair such that $M \rightarrow_{\beta_I\beta_S} N$ and $\Gamma \vdash N : \tau$. Then, $\Gamma \vdash M : \tau$.*

The previous result has as a consequence, that the principal types of BCK- λ -terms are exactly the principal types of BCK- λ -terms in $\beta_I\beta_S$ -normal form. Next we argue that for every $\beta_I\beta_S$ -normal form in the simply typed λ -calculus, there is a term, in a canonical form, with the same principal type. Note that the only redexes in a $\beta_I\beta_S$ -normal form are K-redexes of the form $(\lambda k.M)N$, where $k \notin FV(M)$ and such that $(\lambda k.M)$ has no further argument other than N . On the other hand, for typable terms M and N and such that $k \notin FV(M)$, the terms $(\lambda k.M)N$ and $\mathbf{K}MN$ have the same principal type. We call a $\beta_I\beta_S$ -normal form, where every redex $(\lambda k.M)N$ has been substituted by $\mathbf{K}MN$ a $(\beta_I\beta_S + \mathbf{K})$ -normal form. It is easy to show that every BCK- λ -term M in the simply typed λ -calculus has a unique $(\beta_I\beta_S + \mathbf{K})$ -normal form N in the enriched calculus and that every occurrence of \mathbf{K} in N has exactly two arguments.

Now, we introduce the notion of β_\star -reduction defined by the contraction rules

$$\mathbf{K}(\lambda x.M)N \rightarrow_{\beta_\star} \lambda x.\mathbf{K}MN \quad \text{and} \quad \mathbf{K}(\mathbf{K}MN)L \rightarrow_{\beta_\star} \mathbf{K}(\mathbf{K}NL).$$

Note that every β_\star -redex admits exactly the same types as its contractum. Furthermore, given a $(\beta_I\beta_S + \mathbf{K})$ -normal form M , there is no infinite β_\star -reduction sequence starting in M ¹. Thus, every $(\beta_I\beta_S + \mathbf{K})$ -normal form M admits (at least) one β_\star -normal form and any β_\star -normal form of M admits the same types as M . We conclude that the set of principal types of BCK- λ -terms equals the set of principal types of β_\star -normal forms obtained from BCK- λ -terms in $(\beta_I\beta_S + \mathbf{K})$ -normal form. We will call these terms *standardized $(\beta_I\beta_S + \mathbf{K})$ -normal forms*, or abbreviated standardized $(\beta_I\beta_S + \mathbf{K})$ -nfs, and analyze their principal types in the rest of this paper.

¹ Every application of the first rule moves an occurrence of \mathbf{K} inside an abstraction and an application of the second rule does not change the number of abstractions over the occurrences of \mathbf{K} . Since there is only a finite number of abstractions in M , the first rule can only be applied a finite number of times. Thus, every infinite β_\star -reduction sequence starting in M would contain an infinite sequence of applications of the second rule. This is impossible, since every application of this rule moves parenthesis to the right, which clearly can only be performed a finite number of times.

For M of the form $\lambda x_1 \dots x_n. x P_1 \dots P_k$ or $\lambda x_1 \dots x_n. \mathbf{K}(x P_1 \dots P_k) D$, with $k \geq 0$,
 let $\text{PT}(M) = \text{type}(x_1, M) \rightarrow \dots \rightarrow \text{type}(x_n, M) \rightarrow a_x^+$,
 where

$$\text{type}(y, P) = \begin{cases} \text{PT}(N_1) \rightarrow \dots \rightarrow \text{PT}(N_m) \rightarrow a_y^- & \text{if } y N_1 \dots N_m \text{ is a subterm of } P \\ & \text{for some } m \geq 0 \text{ such that} \\ & y \text{ has exactly } m \text{ arguments;} \\ a_y^- & \text{if } y \text{ has no occurrence in } P. \end{cases}$$

Fig. 1. The PT - Algorithm

The following characterization of standardized $(\beta_I \beta_S + \mathbf{K})$ -normal forms results directly from the definition of $\beta_I \beta_S$ - and β_* -reduction.

Corollary 1. *Every standardized $(\beta_I \beta_S + \mathbf{K})$ -normal form M is of either one of the following two forms*

$$M \equiv \lambda x_1 \dots x_n. x P_1 \dots P_k \quad \text{or} \quad M \equiv \lambda x_1 \dots x_n. \mathbf{K}(x P_1 \dots P_k) D,$$

where $n, k \geq 0$ and such that P_1, \dots, P_k and D are standardized $(\beta_I \beta_S + \mathbf{K})$ -normal forms.

Definition 4. *Let M be a standardized $(\beta_I \beta_S + \mathbf{K})$ -nf as in the previous corollary. The complete subterms of M are M itself and all complete subterms of P_1, \dots, P_k and D . We call x the head-variable of M , x_1, \dots, x_n its initial abstraction sequence and P_1, \dots, P_k the arguments of x .*

4 Computing the Principal Types

In this section we define a principal type algorithm for standardized $(\beta_I \beta_S + \mathbf{K})$ -nfs and use it to identify several properties of BCK-principal types. In the following definition we use only type variables with names of the form a_x , where x is a term-variable, as we have already done for the principal types given in the examples. Also, all occurrences of type variables will be marked with positive (resp. negative) signs according as the corresponding occurrences in the type that is being computed are positive (resp. negative) occurrences.

Definition 5. *Let M be a standardized $(\beta_I \beta_S + \mathbf{K})$ -nf. Then, $\text{PT}(M)$ is defined as in Fig.1.*

Proposition 1. *Let M be a closed standardized $(\beta_I \beta_S + \mathbf{K})$ -nf. Then $\text{PT}(M)$ is a principal type for M .*

Example 5. Naturally, $\text{PT}(M_i) = \sigma_{M_i}$, for $i = 1, 2, 3$.

By inspection of the PT-algorithm, it is easy to verify that every negative (resp. positive) occurrence of a type variable a_x in $\text{PT}(M)$ is due to an occurrences of x in M that is (resp. is not) in an abstraction sequence. As such,

we call occurrences of term-variables in M *negative* if and only if they are in abstraction sequences and *positive* otherwise.

Using the PT-algorithm we obtain several properties of BCK-principal types.

Proposition 2. *Every principal type of a standardized $(\beta_I\beta_S + \mathbf{K})$ -nf is balanced.*

Definition 6. *A balanced type is called acyclic if and only if its relevance graph is acyclic.*

Proposition 3. *Every principal type of a standardized $(\beta_I\beta_S + \mathbf{K})$ -nf is acyclic.*

Proposition 4. *Consider a balanced acyclic type σ and its relevance graph $G(\sigma)$. Then, the following hold.*

1. *If there are $n \geq 0$ negative type variables b_1, \dots, b_n in σ , then $G(\sigma)$ consists of exactly $n + 1$ connected components, all of which are trees, denoted by $\mathcal{T}_0, \dots, \mathcal{T}_n$.*
2. *The roots of $\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_n$ are respectively $\text{core}(\sigma), b_1, \dots, b_n$.*
3. *All internal nodes of $\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_n$ are neutral type variables.*

In the sequel we assume that the \mathcal{T}_i are labelled in such a way that the root of \mathcal{T}_0 is always $\text{core}(\sigma)$ and that the root of \mathcal{T}_i is b_i , for $1 \leq i \leq n$. We conclude the following from propositions 2, 3 and 4.

Corollary 2. *Every BCK-principal type satisfies the conditions of the previous proposition.*

Let σ be a balanced type σ whose relevant graph consists of trees $\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_n$. From now on, and as we have already done in the examples, we will consider $\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_n$ as ordered trees such that

- if b and c are both children of a node a in some \mathcal{T}_i , and the (corresponding) positive occurrence of b is to the left of the positive occurrence of c in σ , then node b is to the left of node c in \mathcal{T}_i .

5 Discards and Discarding Variables

In this section we identify special subterms of standardized $(\beta_I\beta_S + \mathbf{K})$ -nfs, that correspond directly to the trees $\mathcal{T}_0, \dots, \mathcal{T}_n$ in the relevance graph of $\text{PT}(M)$, cf. proposition 4.

Definition 7. *Consider a standardized $(\beta_I\beta_S + \mathbf{K})$ -normal form M . The set of discarding variables in M is inductively defined as follows.*

- *The variables in the initial abstraction sequence of D are discarding, for every subterm of M of the form $\lambda x_1 \dots x_n. \mathbf{K}(xP_1 \dots P_k)D$.*
- *All variables in the initial abstraction sequence of an argument of another discarding variable in M are discarding.*

$$\begin{aligned}
& \text{Strip}(\lambda x_1 \dots x_n. \mathbf{K}(x P_1 \dots P_k) D, M) = \text{Strip}(\lambda x_1 \dots x_n. x P_1 \dots P_k, M) \\
& \text{Strip}(\lambda x_1 \dots x_n. x P_1 \dots P_k, M) = \\
& \quad \begin{cases} \lambda x_1 \dots x_n. x & \text{if } x \text{ is a discarding variable in } M; \\ \lambda x_1 \dots x_n. x \text{ Strip}(P_1, M) \dots \text{Strip}(P_k, M) & \text{otherwise.} \end{cases}
\end{aligned}$$

Fig. 2. The Strip Algorithm

A term D is called a discard in M iff

- the head-variable of D is not discarding and
- M has a subterm of the form $\lambda x_1 \dots x_n. \mathbf{K}(x P_1 \dots P_k) D$ or D is the argument of a discarding variable in M .

Example 6. All three terms M_1 , M_2 as well as M_3 are standardized $(\beta_I \beta_S + \mathbf{K})$ -normal forms. Term M_1 , from [8], is in fact even a β -normal form. As such, M_1 has neither discards nor discarding variables. The term M_2 has discarding variables w and t and discards in M_2 are $(\lambda w. z(w(\lambda t. yt)))$ and $\lambda t. yt$. Finally, M_3 has two discards $\lambda w. u(\lambda s. y(\lambda t. \mathbf{K} w(\lambda o. tro)))$ and $\lambda o. tro$. The variables w and o are discarding in M_3 and all other variables are non-discarding.

Definition 8. Given a closed standardized $(\beta_I \beta_S + \mathbf{K})$ -nf M and a complete subterm N of M , we define the term $\text{Strip}(N, M)$ as in Fig. 2.

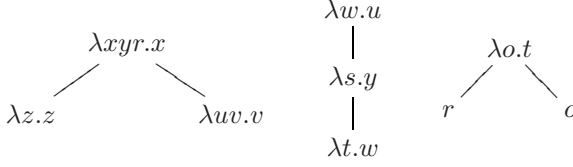
Example 7. Consider again the terms M_2 and M_3 . For $M_2 = \lambda xyz. \mathbf{K} x(\lambda w. z(w(\lambda t. yt)))$ with discards $D_1 = \lambda w. z(w(\lambda t. yt))$ and $D_2 = \lambda t. yt$, there is $\text{Strip}(M_2, M_2) = \lambda xyz. x$, $\text{Strip}(D_1, M_2) = \lambda w. zw$ and $\text{Strip}(D_2, M_2) = \lambda t. yt$, respectively with Böhm trees

$$\begin{array}{ccc}
\lambda xyz. x & \lambda w. z & \lambda t. y \\
& \downarrow & \downarrow \\
& w & t
\end{array}$$

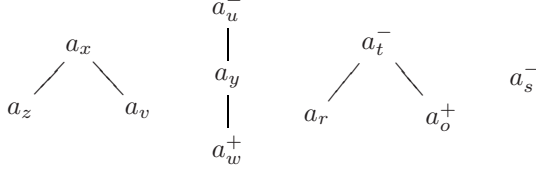
Compare their structure and head-variables labelling the nodes with the structure and type variables in the relevance graph of its principal type $G(\sigma_{M_2}) =$

$$\begin{array}{ccc}
a_x & a_z^- & a_y^- \\
& \downarrow & \downarrow \\
& a_w^+ & a_t^+
\end{array}$$

The term $M_3 = \lambda xyr. x(\lambda z. z)(\lambda uv. \mathbf{K} v(\lambda w. u(\lambda s. y(\lambda t. \mathbf{K} w(\lambda o. tro))))$ has two discards $D_1 = \lambda w. u(\lambda s. y(\lambda t. \mathbf{K} w(\lambda o. tro)))$ and $D_2 = \lambda o. tro$. We have, $\text{Strip}(M_3, M_3) = \lambda xyr. x(\lambda z. z)(\lambda uv. v)$, $\text{Strip}(D_1, M_3) = \lambda w. u(\lambda s. y(\lambda t. w))$ and $\text{Strip}(D_2, M_3) = \lambda o. tro$, respectively with Böhm trees



and relevance graph $G(\sigma_{M_3}) =$



Example 7 suggests the following result.

Proposition 5. *Consider a standardized $(\beta_I\beta_S + \mathbf{K})$ -normal form M and its principal type $\text{PT}(M)$ with relevance graph \mathcal{G} . Let $\{D_1, \dots, D_k\}$ be the set of discards in M and $\{z_{k+1}, \dots, z_n\}$ the set of all non-discarding variables with no positive occurrence in M , where $0 \leq k \leq n$. Finally let $\text{BT}_0, \text{BT}_1, \dots, \text{BT}_k$ be the Böhm trees of $\text{Strip}(M, M), \text{Strip}(D_1, M), \dots, \text{Strip}(D_k, M)$ respectively. Then, \mathcal{G} consists of n trees*

$$\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_k, \mathcal{T}_{k+1}, \dots, \mathcal{T}_n$$

such that

- $\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_k$ have respectively the structure of $\text{BT}_0, \text{BT}_1, \dots, \text{BT}_k$ and nodes are labelled according to the head-variables in the Böhm trees;
- for $i \in \{k+1, \dots, n\}$, tree \mathcal{T}_i consists of just one node a_{z_i} .

6 The Dependence Relation

In the previous section we established a direct correspondence between the structure of the relevance graph of the principal type σ of a standardized $(\beta_I\beta_S + \mathbf{K})$ -normal form M and the terms obtained from M and its discards by algorithm **Strip**. We now introduce a somehow symmetric relation on the type variables in σ that captures the constraints on initial abstraction sequences in M . This will finally allow us to give a characterization of principal BCK- λ -terms in Theorem 2.

Definition 9. *Consider a balanced type σ and two type variables b and c in σ . We write $b \triangleright_\sigma c$ if and only if σ contains a positive occurrence of a subtype of the form*

$$\sigma_1 \rightarrow \dots \rightarrow (\gamma_1 \rightarrow \dots \rightarrow \gamma_k \rightarrow b^-) \rightarrow \dots \rightarrow \sigma_l \rightarrow c^+.$$

Whenever $b \triangleright_\sigma c$, then we say that b depends on c .

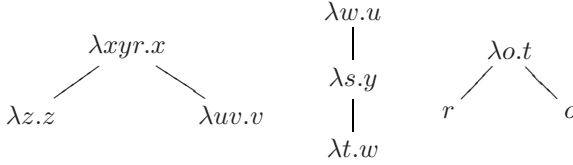
Lemma 2. *Let M , \mathcal{T}_i , BT_i , etc. be as in proposition 5. If a_x and a_y are respectively vertices in trees \mathcal{T}_i and \mathcal{T}_j , for some $0 \leq i, j \leq n$, then the following hold.*

1. $a_x \triangleright_\sigma a_y$ if and only if there is a node of the form $\lambda \dots x \dots .y$ in BT_j and this node is not the root of one of $\text{BT}_1, \dots, \text{BT}_k$.
2. if $i = j$, then $a_x \triangleright_\sigma a_y \Rightarrow a_x \succ_\sigma^* a_y$;

Example 8. Consider again M_3 and its principal type σ_3 and compare the dependence relation for this type, given by

$$\begin{array}{cccc} a_x \triangleright_{\sigma_3} a_x & a_y \triangleright_{\sigma_3} a_x & a_r \triangleright_{\sigma_3} a_x & a_z \triangleright_{\sigma_3} a_z \\ a_u \triangleright_{\sigma_3} a_v & a_v \triangleright_{\sigma_3} a_v & a_s \triangleright_{\sigma_3} a_y & a_t \triangleright_{\sigma_3} a_w \end{array}$$

with the Böhm trees corresponding to M_3 .



The characterization of BCK-principal types is given in the following theorem.

Theorem 2. *Consider a balanced acyclic type σ , with relevance graph \mathcal{G} consisting of $n \geq 0$ trees $\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_n$. Then, σ is the principal type of a closed BCK- λ -term if and only if*

1. *There are no positive variables in \mathcal{T}_0 .*
2. *If a_x and a_y are vertices in the same tree and $a_x \triangleright_\sigma a_y$, then $a_x \succ_\sigma^* a_y$.*
3. *There is a tree \mathbb{T} with nodes $\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_n$ such that, for every two nodes \mathcal{T}_i and \mathcal{T}_j and vertices a_x and a_y , respectively in \mathcal{T}_i and \mathcal{T}_j , the following hold.*
 - (a) *If $a_x \triangleright_\sigma a_y$ and $i \neq j$, then \mathcal{T}_i is a descendent of \mathcal{T}_j in \mathbb{T} .*
 - (b) *If \mathcal{T}_i and \mathcal{T}_j occur in a subtree of a node \mathcal{T}_k in \mathbb{T} and if there are vertices a_z and a_t in \mathcal{T}_k such that $a_x \triangleright_\sigma a_z$ and $a_y \triangleright_\sigma a_t$, then $a_z \succ_\sigma^* a_t$ or $a_t \succ_\sigma^* a_z$.*

For BCK-principal types of β -normal forms we obtain the following simple characterization, equivalent to Theorem 1.

Corollary 3. *Consider a balanced acyclic type σ , with relevance graph \mathcal{G} consisting of $n \geq 0$ trees $\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_n$. Then, σ is the principal type of a closed BCK- λ -term in β -normal form if and only if*

1. *There are no positive variables in \mathcal{T}_0 ;*
2. *if $a, b \in \mathcal{T}_0$, then $a \triangleright_\sigma b$ implies $a \succ_\sigma^* b$;*
3. *$\mathcal{T}_1, \dots, \mathcal{T}_n$ consist of isolated nodes b_1, \dots, b_n .*

Finally the characterization of BCI-principal types reads as follows.

Corollary 4. *A type σ is a principal type of a BCI- λ -term if and only if*

1. *all type variables in σ have exactly one positive and one negative occurrence;*
2. *if $a, b \in \sigma$, then $a \triangleright_\sigma b$ implies $a \succ_\sigma^* b$.*

Acknowledgments. The work presented in this paper has been partially supported by funds granted to *LIACC* through *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *Programa POCTI*.

References

1. Broda, S., Damas, L.: On principal types of combinators. *Theoretical Computer Science* 247, 277–290 (2000)
2. Broda, S., Damas, L.: On Long Normal Inhabitants of a Type. *Journal of Logic and Computation* 15(3), 353–390 (2005)
3. de Groote, P.: The Conservation Theorem revisited. In: Bezem, M., Groote, J.F. (eds.) *TLCA 1993*. LNCS, vol. 664, pp. 163–178. Springer, Heidelberg (1993)
4. Hindley, J.R.: BCK-combinators and linear λ -terms have types. *Theoretical Computer Science* 64, 97–105 (1989)
5. Hindley, J.R., Meredith, D.: Principal type-schemes and condensed detachment. *Journal of Symbolic Logic*, 55, 901–105 (1990)
6. Hindley, J.R.: BCK and BCI logics, condensed detachment and the 2-property. *Notre Dame J. Formal Logic* 34(2), 231–250 (1993)
7. Hirokawa, S.: Principal types of BCK-lambda-terms. *Theoretical Computer Science* 107, 253–276 (1993)
8. Hirokawa, S.: The Relevance Graph of a BCK-Formula. *Journal of Logic and Computation* 3(3), 269–285 (1993)

A Finite-State Functional Grammar Architecture

Alexander Dikovsky

LINA FRE-CNRS 2729, Université de Nantes, France

`alexandre.dikovsky@univ-nantes.fr`

Abstract. We describe a simple and efficiently implementable grammatical architecture for generating dependency trees from meaning structures. It is represented by finite state tree top-down transducers applied to feature trees to convert them into sequences of typed forms (typed sentences). The correctness of the generated types is checked using a simple and efficient dependency calculus. The corresponding dependency structure is extracted from the correctness proof. Using an English example, it is demonstrated that this transduction can be carried out incrementally in the course of composition of the meaning structure.

1 Introduction

The issue under study in this paper is how to design the grammar so that *the surface representation of sentences were incrementally generated in the course of planning their meaning*. So stated, this issue does not fit the standard generativistic frame, nor even in its modern form [4,15]. The closest suitable frame is that of functional grammar, more specifically, the Prague Functional Generative Description [21], where a similar issue was already studied [22] and the Lexical Functional Grammar [19], especially the Optimal Syntax (OS) proposal [3]. As with OS, the *input meaning representation* we consider is underspecified with respect to the logical form. This representation, called *discourse plan* (DP), inspired by DRT [17] and Frame Semantics [13] was introduced in [8] and developed in [11]. The DP are a kind of feature trees which combine argument thematic roles, lexical types, topic/focus/background partitioning, distributive/non-distributive collectivity, co-reference, scope hierarchy etc., and are interpreted using typed lambda-terms. The DP underspecification is completely different from the universally adopted “scope-precedence-abstraction” underspecification common to many proposals (cf. [20,2,5,14]). Its distinguishing feature is that, theoretically, DP should uniquely encode argument scopes through their *communicative ranks* in *semantic diatheses*, a concept close to the *argument perspective* in Frame Semantics. This fundamental difference has far-reaching semantic consequences. E.g., an adequate DP of the sentence *Every linguist knows two languages* represents only its $\forall\exists$ reading (the dual one needs a different perspective expressed through a different wording). Another consequence is that ideally, in contrast to OS, the correspondence mapping between DP and surface structures should be *functional*. In our approach to generation, we make two important choices: we choose *dependency structures* (DS) as the surface structure representation and

we use Categorical Dependency Grammars (CDG) [9,6] as the basic grammatical formalism. CDG are lexicalized type grammars defined using a simple dependency type calculus. They are significantly more expressive than CF-grammars and are parsed in polynomial time. In contrast to OS and similar parallel correspondence syntactic theories originating from LFG, in the generation architecture described in this paper, the functional correspondence mapping converts DP into sequences of typed surface forms. The DP is feasible if the generated typing is proved correct in the dependency calculus. The corresponding surface DS is extracted from the proof. So, at least theoretically, this architecture can be seen as a “generate and check” process. The well-typing proof may or may not be a part of the conversion. In the first case, the conversion may also be seen as incremental “generation-while-planning”. One can expect that the generation process is substantially simpler than the corresponding parsing process. Indeed, such hard problems as lexical ambiguity, anaphora resolution, ellipsis etc. do not arise while generation. This is why we choose deterministic finite state tree transducers (FST) as a formal model of the functional correspondence mapping.

In what follows, we sketch out the DP and the CDG dependency calculus, next we define FST over general feature trees and finally we describe their subclass converting DP into typed sequences (*DP-DS-converters*). The architecture is illustrated by a fragment of a modular DP-DS-converter for the English. This example shows that at least the core part of this architecture allows generation-while-planning.

2 Discourse Plans

Our choice of DP in the role of input meaning representation is principally explained by the way it represents the (verb) argument structure. In DP, as in Frame Semantics, one and the same structure (*situation* in DP terms) may have different arguments when seen from different perspectives, and the arguments are identified with *thematic roles* and provided with *lexical types*. In the DP below, are used the most general proto-roles. E.g., the role SBJ corresponds to the semantic subject/agent, OBJ corresponds to the semantic patient/object/theme, INSTR corresponds to the semantic instrument/mediator. We don’t choose between different theories of thematic roles (cf. [16]) and proceed from the following assumptions. Firstly, the roles are labels for classes of semanteme arguments which uniquely identify the arguments for every semanteme of a given lexical type. Secondly, the thematic roles form a *semantic prominence* hierarchy with the effect that if a role R_1 is more prominent than a role R_2 ($R_1 > R_2$, e.g., SBJ > OBJ) co-occurring in a situation S , then the R_1 -argument *overscopes* the R_2 -argument of S . This assumption is perfectly conformable to the main approaches to the prominence order: the logical approach [12], where some variables in the entailed characteristic properties of “less prominent” role arguments are bound with values instantiated in the properties of “more prominent” role arguments, and the approach of lexical semantics (cf. [16]) in which “less prominent” means “deeper in the event structure defining the situation’s lexical meaning”. The

lexical types used in DP form a hierarchy. Above four most general types: **s** (*sententials*), **n** (*nominators*), **q** (*qualifiers*), **c** (*circumstantials*) there are more specific ones, e.g. **s** \prec **s_{eff}** (*caused effect situations*), **n** \prec **n_a** (*animated nominators*), **q** \prec **q_{prop}** (*property qualifiers*), **c** \prec **c_{mann}** (*manner circumstantials*).

Identification of arguments with thematic roles is standard in subcategorization frames, but DP go further and, following the proposal of the Prague school ([21]), introduce into the frames elements of communicative structure to distinguish between different perspectives (*situation profiles* in DP terms). E.g., in the DP in Fig. 1 is used situation $\langle\langle\text{open}\rangle\rangle$ in its canonical profile:

$$\langle\langle\text{open}(\text{SBJ}^{\text{n}_a}, \text{OBJ}^{\text{n}}, \text{INSTR}^{\text{n}})^{\text{s}_{\text{eff}}}\rangle\rangle,$$

where it has value type **s_{eff}** and three nominator arguments with the roles SBJ, OBJ, INSTR. All other possible situation profiles are derived from the canonical one using *semantic diatheses*, i.e. explicit transformations of the canonical profile depending on the speaker's view of the salience and the intended type of each semanteme's *actant* (its proper argument identified by a thematic role, as opposed to other optional circumstantial arguments identified by other attributes). So every situation in every its profile has a particular first order type with primitive argument subtypes. The views are expressed using several *communicative ranks* assigned to the actants: T(topic), O(topic to be elided in the surface structure), \odot (focus), \oplus (background), \ominus (periphery). The topicality order $T/O > \odot > \oplus$ corresponds to the prominence order of the actants in the canonical profile. The diatheses assign to the actants new ranks, roles and/or types and eventually change the situation value type. In this way (rather close to [18]) are defined the diatheses of passivization, nominalization, object alternation, etc. E.g., the DP in Fig. 2 makes use of the diathesis *decaus-ins* of $\langle\langle\text{open}\rangle\rangle$

$$\langle\langle\text{dth}_{\text{decaus-ins}}(\text{key}_{\oplus}, \emptyset \leftrightarrow \text{SBJ}_{\ominus}, \text{SBJ} \leftrightarrow \text{INSTR}_{\text{T}}, \text{OBJ} \leftrightarrow \text{OBJ}_{\odot})^{\text{s}_{\text{eff}}}\rangle\rangle$$

deriving the profile in which the peripheral SBJ-actant is eliminated, the topicalized INSTR-actant is promoted to SBJ and the OBJ-actant is left intact.

Example 1. *John easily opened the door with the new key*

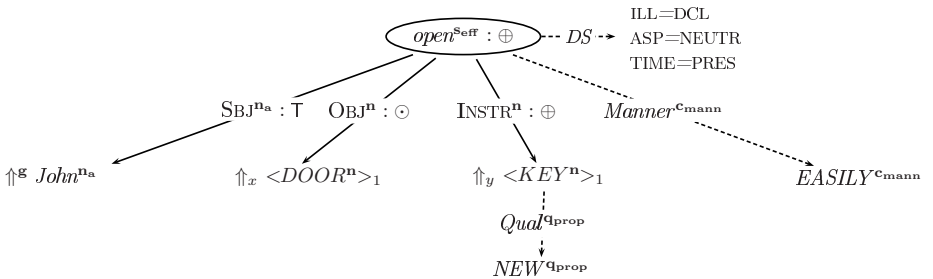


Fig. 1. A DP using the canonical profile of $\langle\langle\text{open}\rangle\rangle$

(Representing DP, we use the following graphical notation. The ellipses signify situations. The boxes represent the diatheses, in particular, the assignment of

communicative ranks to the actants. Solid lines link the actants to their semantemes and are labeled with roles. Dashed lines link circumstantials to their governor semantemes and are labeled with names of attributes. Several non-optional attributes represent the so called *discourse status* and state the situation's illocutive status, the aspect and the semantic time). Diamonds represent embedded functions: e.g. the ι -operator. $\uparrow^{\mathbf{g}}$ stands for global context references and \downarrow_x, \uparrow_x stand for anaphoric references introduction and access. The superscripts correspond to lexical types.)

Example 2. *The new key easily opened the door*

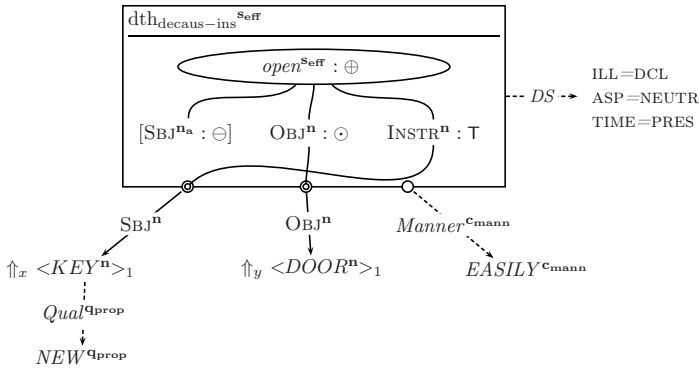


Fig. 2. DP DP_{decaus} using a semantic derivative of $\langle\langle open \rangle\rangle$

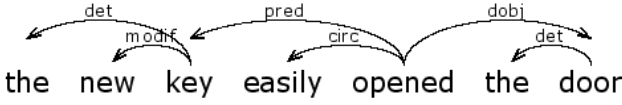
From the semantic perspective, the DP may be seen as first order typed terms serving for compositional definition of semantic expressions. From the syntactic perspective, they are a special kind of feature trees.

Definition 1. *The set of feature trees (f-trees) over functors $F = \{f/n \mid n \geq 0\}$ and sorts Σ is the least set $\mathcal{T}(F, \Sigma)$ which (i) contains all constants $f/0 \in F$, and (ii) contains all expressions $f(b_1 : t_1, \dots, b_n : t_n)$ for all $f/n \in F$, all pairwise different sorts $b_1, \dots, b_n \in \Sigma$ and all f-trees $t_1, \dots, t_n \in \mathcal{T}(F, \Sigma)$.*

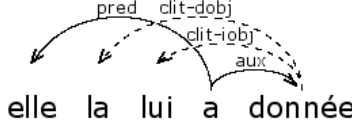
In particular, a DP using as its main situation a situation K in derived profile $dth_d(K)$, can be seen as the f-tree $dth_d(K)(s_1 : \pi_1, \dots, s_m : \pi_m)$, where the sorts s_i are either the roles of shifted actants with non-null ranks or the attributes identifying present circumstantials, and π_i are the f-trees representing the corresponding argument sub-plans. For instance, in the DP in Fig. 1 using $\langle\langle open \rangle\rangle$ in canonical profile, $s_1 = SBJ^{na}$, $s_2 = OBJ^n$, $s_3 = INSTR^n$, $s_4 = MANNER^{cmann}$. On the other hand, in the f-tree representing the DP in Fig. 2, which uses the derivative $\langle\langle dth_{decaus-ins}(open) \rangle\rangle$, $s_1 = SBJ^n$, $s_2 = OBJ^n$ and $s_3 = MANNER^{cmann}$.

3 Generalized Categorical Dependency Grammars

The *generalized categorical dependency grammars* (gCDG) recently introduced in [9] and studied in [6,7] are categorical grammars generating sentences together with their *dependency structures* (DS). The DS may be trees (DT) as it is the case in Fig. 3 or not (cf. Fig. 4). Whatever is the case, they are linearly ordered and this order may be *projective*, as in Fig. 3(a), or not, as in Fig. 3(b). The non-projective order is due to *discontinuous dependencies* in which the governor g is separated from the subordinate s by a word not dominated by g (as it is the case of the auxiliary verb a in Fig. 3(b) separating the participle *donnée* from its pronominalized objects).



(a) A projective DT



(b) A non-projective DT (French: *she_{FEM} to-him has given)

Fig. 3.

Like the classical categorical grammars, gCDG are lexicalized, i.e. are defined by a *lexicon*: a function λ assigning to each word a finite set of its dependency types. A type has the form $[l_m \setminus \dots \setminus l_1 \ v / r_n \dots / r_1]^P$, where v is the (continuous) dependency on the governor, l_m, \dots, l_1 and r_n, \dots, r_1 are the (continuous) dependencies of left and right subordinates in the reverse order, and P is a sequence of *polarized valencies* (the *potential*) determining discontinuous dependencies. Types with empty potentials determine projective DT. E.g., the DT in Fig. 3(a) is determined by the type assignments:

$$\begin{aligned} \text{the} &\mapsto \text{det} & \text{key} &\mapsto [\text{modif} * \setminus \text{det} \setminus \text{pred}] & \text{new} &\mapsto \text{modif} \\ \text{easily} &\mapsto \text{circ} & \text{opened} &\mapsto [\text{circ} * \setminus \text{pred} \setminus S / \text{dobj}] & \text{door} &\mapsto [\text{det} \setminus \text{dobj}] \end{aligned}$$

where the types *modif* and *circ* are *iterated* and S is the sentence type. The polarized valencies are of four kinds: $\swarrow d$ (negative: that of the distant right governor through dependency d), $\searrow d$ (same on the left), $\nearrow d$ (positive: that of a distant left subordinate through dependency d), $\nwarrow d$ (same on the right). A pair of *dual* valencies $\swarrow d$ and $\nwarrow d$ ($\nearrow d$ and $\searrow d$) define a discontinuous dependency d . A negative valency can be anchored on a host word using anchor

types $\#(\swarrow d)$, $\#(\searrow d)$. E.g., the lexicon below uniquely determines the DT in Fig. 3(b) ($\swarrow \textit{clit} - \textit{dobj}$, $\swarrow \textit{clit} - \textit{iobj}$ are anchored on the auxiliary a):

$$\begin{array}{lll} \text{elle} \mapsto \text{pred} & \text{la} \mapsto [\#(\surd \text{clit} - \text{dobj})] \surd \text{clit} - \text{dobj} & \text{lui} \mapsto [\#(\surd \text{clit} - \text{iobj})] \surd \text{clit} - \text{iobj} \\ \text{a} \mapsto [\#(\surd \text{clit} - \text{iobj}) \setminus \#(\surd \text{clit} - \text{dobj}) \setminus \text{pred} \setminus S / \text{aux}] & & \text{donnée} \mapsto [\text{aux}] \setminus \text{clit} - \text{dobj} \setminus \text{clit} - \text{iobj} \end{array}$$

Such dependency types are formalized with the following calculus ¹

$$\mathbf{L}^1. C^{P_1} [C \setminus \beta]^{P_2} \vdash [\beta]^{P_1 P_2}$$
$$\mathbf{I}^1. C^{P_1}[C^* \setminus \beta]^{P_2} \vdash [C^* \setminus \beta]^{P_1 P_2}$$
$$\Omega^1. [C^* \setminus \beta]^{\dot{P}} \vdash [\beta]^{\dot{P}}$$

D¹. $\alpha^{P_1(\swarrow C)P(\nwarrow \hat{C})P_2} \vdash \alpha^{P_1 P P_2}$, if $(\swarrow C)P(\nwarrow C)$ satisfies the pairing rule

FA (first available): P has no occurrences of $\swarrow C, \nwarrow C$.

\mathbf{I}^1 is the classical elimination rule. Eliminating the argument subtype $C \neq \#(\alpha)$ it constructs the (projective) dependency C and concatenates the potentials. $C = \#(\alpha)$ creates no dependency. \mathbf{I}^1 derives $k > 0$ instances of C . Ω^1 serves for the case $k = 0$. \mathbf{D}^1 derives discontinuous dependencies. It pairs and eliminates dual valencies satisfying the rule \mathbf{FA} to create the discontinuous dependency C .

For a DS D and a string x , let $G(D, x)$ denote the relation: D is constructed in a proof $\Gamma \vdash S$ for some $\Gamma \in \lambda(x)$. Then the *language* generated by G is the set $L(G)_{=df} \{w \mid \exists D \ G(D, w)\}$.

The gCDG are very expressive. Evidently, they generate all CF-languages. They can also generate non-CF languages.

Example 3. The g CDG

$$G_{abc} : a \mapsto A \swarrow^A, [A \setminus A] \swarrow^A, b \mapsto [B/C] \nwarrow^A, [A \setminus S/C] \nwarrow^A, c \mapsto C, [B \setminus C]$$

generates the language $\{a^n b^n c^n \mid n > 0\}$ [9]. For instance, $G_{abc}(D^{(3)}, a^3 b^3 c^3)$ holds for the dependency structure (DS) in Fig. 4 and the string $a^3 b^3 c^3$ due to the proof in Fig. 5.

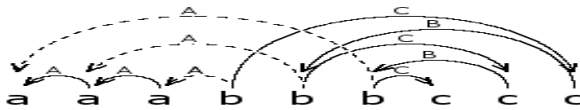


Fig. 4. Dependency structure for $a^3b^3c^3$

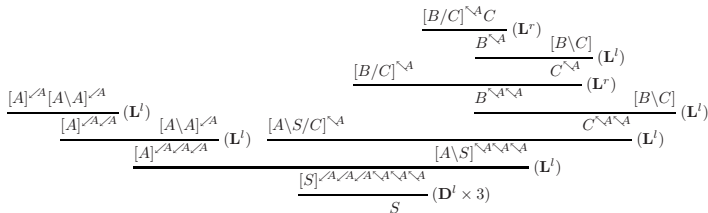


Fig. 5. Dependency structure correctness proof

¹ We show left-oriented rules. The right-oriented are symmetric.

Recently, it was shown that the family of gCDG-languages $\mathcal{L}(gCDG)$ is an AFL [7]. Seemingly, it is incomparable with the mildly CS languages [24], generated by multi-component TAG, linear CF rewrite systems, minimalist grammars [23]. $\mathcal{L}(gCDG)$ contains non-TAG languages, e.g. $L^{(m)} = \{a_1^n a_2^n \dots a_m^n \mid n \geq 1\}$ for all $m > 0$. In particular, it contains the language $MIX = \{w \in \{a, b, c\}^+ \mid |w|_a = |w|_b = |w|_c\}$, for which E. Bach conjectures that it is not mildly CS. On the other hand, we conjecture [6] that this family does not contain the copy language $L_{copy} = \{xx \mid x \in \{a, b\}^*\}$, which is TAG². gCDG have an efficient polynomial time parsing algorithm [6,7] which rests upon a property of independence of basic categories and polarized valencies in the proofs in the dependency calculus. This property is expressed in terms of *projections* of categories and “*well-bracketing*” criteria for potentials.

For a sequence of categories γ , its *local projection* $\|\gamma\|_l$ and *de valency projection* $\|\gamma\|_v$ are defined as follows:

1. $\|\varepsilon\|_l = \|\varepsilon\|_v = \varepsilon$; $\|\alpha\gamma\|_l = \|\alpha\|_l \|\gamma\|_l$ and $\|\alpha\gamma\|_v = \|\alpha\|_v \|\gamma\|_v$ for a category α .
2. $\|C^P\|_l = C$ et $\|C^P\|_v = P$ for every category C^P .

To speak about “well-bracketing” of potentials, we interpret $\swarrow d$ and $\nearrow d$ as *left brackets* and $\searrow d$ and $\nwarrow d$ as *right brackets*. For a left bracket valency α , the corresponding (*dual*) right bracket valency is denoted $\check{\alpha}$. The set of left-bracket valencies is denoted V^l . That of right-bracket valencies is denoted V^r . $V \stackrel{\text{def}}{=} V^l \cup V^r$. A potential is *balanced* if it is well bracketed in the usual sense.

For a potential P and dual valencies $\alpha \in V^l$, $\check{\alpha} \in V^r$, the values

$$\begin{aligned} \Delta_\alpha(P) &= \max\{|P'|_\alpha - |P'|_{\check{\alpha}} \mid P' \text{ is a suffix of } P\}, \\ \Delta_{\check{\alpha}}(P) &= \max\{|P'|_{\check{\alpha}} - |P'|_\alpha \mid P' \text{ is a prefix of } P\} \end{aligned}$$

(where $|\gamma|_\alpha$ is the number of occurrences of α in γ) express the deficits of right and left α -brackets in P (i.e. the maximal number of right and left bracket α -valencies which need to be added to P on the right (left) so that it became balanced. It is not difficult to prove that a potential P is balanced iff $\sum_{\alpha \in V} \Delta_\alpha(P) = 0$.

Let \mathbf{c} be the projective core of the dependency calculus, consisting of the rules **L**, **I** and **Ω** and $\vdash_{\mathbf{c}}$ denote the provability relation in this sub-calculus. One of the key results of [6,7] is the following property of *projections independence* providing the abovementioned polynomial time parsing algorithm.

Theorem 1. *For a gCDG G with dictionary λ and a string x , $x \in L(G)$ iff there is $\Gamma \in \lambda(x)$ such that $\|\Gamma\|_l \vdash_{\mathbf{c}} S$ and $\|\Gamma\|_v$ is balanced.*

By the way, from this theorem it follows that the correctness of an assignment of categories to words in a sentence is recognized in real time.

Corollary 1. *For every sequence of types γ with distinguished constituents (i.e. without elimination ambiguity), the problem of provability $\gamma \vdash S$ in the dependency calculus is resolved in real time $|\gamma|$ using one stack and v counters, where v is the number of polarized valency names.*

² It doesn't prevent that gCDG explain the cross-serial dependencies [10].

4 Finite State Converters of F-Trees into D-Structures

First of all, we explain how the general f-trees are converted into DS. Next, we will adapt this conversion to the DP.

Definition 2. *Deterministic FS transducer (DFS transducer) is an automaton on f-trees $\Gamma = (F, \Sigma, W, Q, q_0, \gamma, \delta)$, where $\mathcal{T}(F, \Sigma)$ is the set of f-trees over F and Σ, W is a set of words, Q is a finite set of states, q_0 being the initial state, γ is a lexical interpretation, i.e. a function $F \times Q \rightarrow W$ and δ is a transition function defined as follows.*

Let $B = \{b_1, \dots, b_k\} \subseteq \Sigma$ be a set of sorts and $f/k \in F$ be a functor, $k \geq 0$. Then $\Pi(f, B, Q)$ denotes the set of all strings x over $(F \times Q) \cup (B \times Q)$ such that x has at most one occurrence (f, q) for some $q \in Q$ and at most one occurrence of (b, q) for some $q \in Q$ for every sort $b \in B$ ³. In these terms, δ is a function $\{\delta : (f(b_1, \dots, b_k), q) \mapsto \beta \mid \beta \in \Pi(f, \{b_1, \dots, b_k\}, Q)\}$. Elementary associations $((f(b_1, \dots, b_k), q) \mapsto \beta) \in \delta$ are called transitions.

For an f-tree π and a state $q \in Q$, $\Gamma((\pi, q))$ is defined by:

$$\Gamma((\pi_n, q_n) \dots (\pi_1, q_1)) = \Gamma((\pi_1, q_1)) \dots \Gamma((\pi_n, q_n)).$$

$$\Gamma((f, q)) = \gamma(f, q), \text{ for all } f \in F.$$

$$\Gamma((f(b_1 : \pi_1, \dots, b_k : \pi_k), q)) = \Gamma(\sigma(\delta(f(b_1, \dots, b_k), q))),$$

where σ is the natural morphism $\sigma((b_i, q)) = (\pi_i, q)$, $\sigma(\alpha) = \alpha$ for $\alpha \notin \{b_1, \dots, b_k\}$.

The transduction $\Gamma : \mathcal{T}(F, \Sigma) \rightarrow W^+$ is defined by $\Gamma(\pi) = \Gamma((\pi, q_0))$.

Definition 3. FT-DS converter $T = (\Gamma, G)$ is a DFS transducer Γ whose lexical interpretation γ_T is coordinated with the lexicon λ of a gCDG G through a typing function $\text{type} : Q \rightarrow \bigcup_{w \in W} \lambda(w)$ as follows: $\gamma_T(f, q) = w : \text{type}(q)$ and $\text{type}(q) \in \lambda(w)$. T uniquely determines the following relation between the f-trees, the strings and the D-structures: $T(\pi, x, D)$ iff $\Gamma(\pi) = w_1 : C_1 \dots w_m : C_m$, where $x = w_1 \dots w_m$, $C_1 \dots C_m \vdash S$ and D is constructed in this proof.

Language of T : $L(T) = \{x \in W^+ \mid \exists \pi, D(T(\pi, x, D))\}$.

Example 4. It is not difficult to see that the following FT-DS converter T_{abc} generates the language $L(T_{abc}) = \{a^n b^n c^n \mid n > 0\}$.

$$T_{abc} = \begin{cases} (h(1, 2), q_0) \rightarrow (1, -)(2, S) & (f(1, 2), S) \rightarrow (f, S)(1, +)(2, \setminus) \\ (g(1), -) \rightarrow (1, -)(g, -) & (f, S) \rightarrow b : [A \setminus S/C]^{\setminus A} \\ (g, -) \rightarrow a : [A \setminus A]^{\setminus A} & (f(1, 2), +) \rightarrow (f, +)(1, +)(2, \setminus) \\ (g_0, -) \rightarrow a : A^{\setminus A} & (f, +) \rightarrow b : [B/C]^{\setminus A} \\ (e, \setminus) \rightarrow c : [B \setminus C] & (f_1(1), +) \rightarrow (f, +)(1, C) \\ (e, C) \rightarrow c : C \end{cases}$$

For instance, T_{abc} converts the f-tree $\pi^{(3)}$ in Fig. 6 into the typed string $(a : A^{\setminus A})(a : [A \setminus A]^{\setminus A})(a : [A \setminus A]^{\setminus A})(b : [A \setminus S/C]^{\setminus A})(b : [B/C]^{\setminus A})(b : [B/C]^{\setminus A})(c : C)(c : [B \setminus C])(c : [B \setminus C])$. So $T_{abc}(\pi^{(3)}, a^3 b^3 c^3, D^{(3)})$, where $D^{(3)}$ is the DS in Fig. 4.

³ This is a resource sensitive, "no copy" constraint.

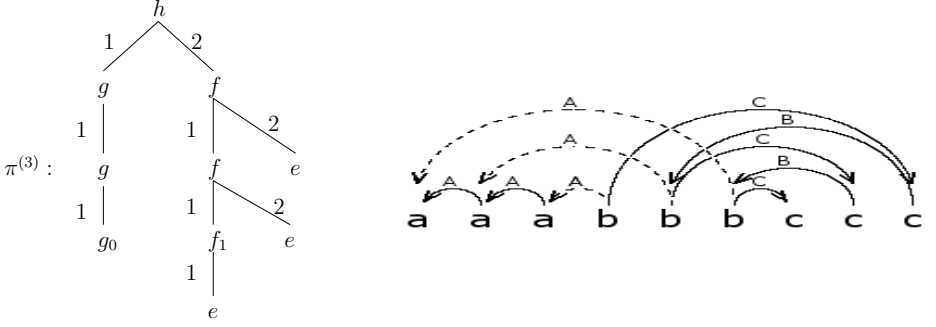


Fig. 6.

The FT-DS converters have the same weak generative power as gCDG in the following sense.

Theorem 2. *The family $\mathcal{L}(FT-DS(RTL))$ of languages $T(L)$, where L is a regular f-tree language and T is an FT-DS converter, coincides with $\mathcal{L}(gCDG)$.*

Proof scheme. 1. The inclusion $\mathcal{L}(gCDG) \subseteq \mathcal{L}(FT-DS(RTL))$ is evident. 2. The inverse inclusion follows from the established in [1] weak equivalence of gCDG to dependency structure grammars (DSG), a class of rule based generating dependency grammars. Due to the no copy constraint, for every given signature of functors F and sorts Σ , every FS-DT T applied to $\mathcal{T}(F, \Sigma)$ can be converted into a DSG I_T generating the language $T(\mathcal{T}(F, \Sigma))$. It remains to remark that for every FS-DT T and every finite automaton A on f-trees there is an FS-DT T_A such that $T(\Delta(A)) = T_A(\mathcal{T}(F, \Sigma))$, where $\Delta(A)$ is the set of f-trees accepted by A . The latter is proved using the classical construction of a finite automaton for the intersection of languages accepted by finite automata.

5 DP-DS Converters

DP-DS converters are a special kind of the FT-DS converters, where the DP are used in the place of arbitrary f-trees. It is significant that the set of all DP is a regular f-tree language.

In order to arrive at a syntax of DP-DS converters usable in practice, we code the DP and the converter transitions using HPSG-like feature structures. In particular, we use states with two substructures, a semantic one: DS[ILL, ASP, TIME], standing for the discourse status of the converted DP and a morpho-syntactic one: GS[LEFT, RIGHT, VALUE, PTN, GRCL (grammatical class), TNS (tense), NUM (number), PERS (person)]⁴, standing for the features of the corresponding surface construction (see Fig. 7-17). Four selected features VALUE, LEFT, RIGHT and PTN in GS serve to define the dependency type $type(q)$. Namely, $type(q) = [\alpha_1 \setminus \dots \setminus \alpha_l \setminus \gamma / \beta_1 / \dots / \beta_r]^P$ if $q.GS.LEFT = \alpha_1 \setminus \dots \setminus \alpha_l$, $q.GS.RIGHT = \beta_1 / \dots / \beta_r$, $q.GS.VALUE = \gamma$ and $q.GS.PTN = P$.

⁴ Both lists are open.

Like in HPSG, we make use of partially defined (i.e. generalized) structures and of their specializations (denoted $t_{gener} \preceq t_{spec}$). In particular, $t_1 = t_2 \preceq t$ means that t is a specialization of the most general unifier of t_1 and t_2 . The unification is constrained by the hierarchy of primitive types. We also use default values. E.g., the default value $PTN = \varepsilon$ stands for the projective dependency types. Another example: for the *attributes* (sorts of *optional* arguments) listed in the situation argument frames under the name OPTARGS (see Fig. 7), the default value $A = \varepsilon$ stands for the absent arguments A . $OPTARGS = \emptyset$ means that all optional arguments are absent.

The transitions are also represented by partial feature structures and so can be consecutively specialized. The right hand side of a transition is represented as the value of the sort TO. The values of the sort PO stand for right hand side orderings.

A fragment of DP-DS transitions for English

Below, we present several transition schemes sufficient to generate the sentences in examples 1,2 from the corresponding DP. Fig. 7 presents the necessary argument frames.

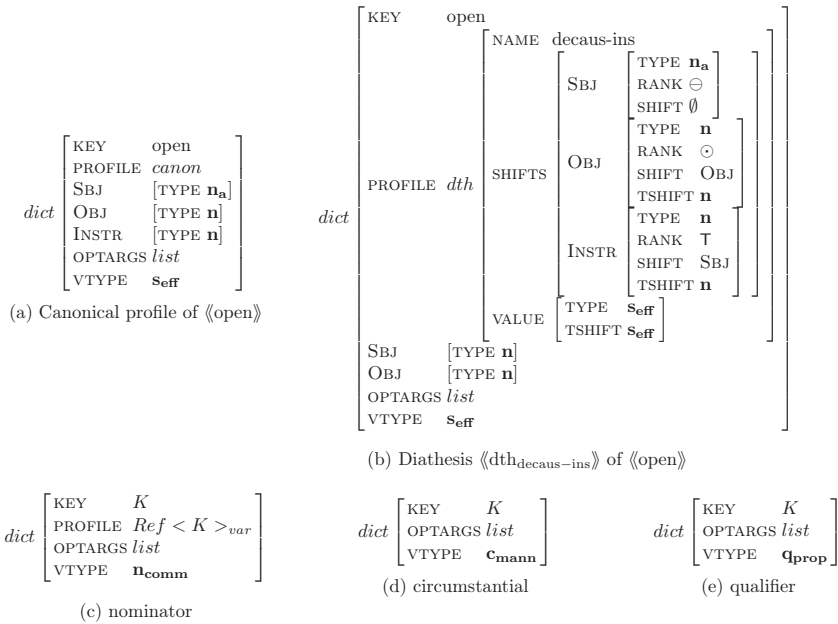


Fig. 7. Argument frames

$$t_{canon} [SEM \ dict [PROFILE \ canon]]$$

$$t_{decaus-ins} [SEM \ dict [PROFILE \ dth [NAME \ decaus - ins]]]$$

Fig. 8. Transition schemes for profile selection

Two transition schemes in Fig. 8 serve to select the argument frame corresponding to the situation profile. Fig. 9 presents a transition scheme applied to the main situations with OBJ-argument in declarative DP. So it applies to both profiles of «open» in Fig. 7(a),(b). Its specific cases assign the axiom value type S to the main transitive verb (e.g. «open») in the surface form.

$$t_{(dcl,obj)} \left[\begin{array}{l} \text{SEM} \quad \text{dict} \left[\begin{array}{l} \text{OBJ} \quad x_{obj} \\ \text{VTYPE} \quad \mathbf{s} \end{array} \right] \\ \text{STATE} \left[\text{DS} \left[\text{ILL DCL} \right] \right] \\ \text{TO} \quad \left[\text{HEAD} \left[\text{STATE} \left[\text{GS} \left[\text{VALUE } S \right] \right] \right] \right] \end{array} \right]$$

Fig. 9. Transition scheme for declarative OBJ-argument situations

The scheme in Fig. 10 applies to the situations in semantical past time and neutral aspect corresponding in English to the surface simple past tense of main verbs. The transitions fitting this scheme assign to the situation's surface verb form $form(K, f)$ its left and right subordinates' subtypes, propagates the corresponding value types to SBJ-sort and OBJ-sort subplans and, at the same time, positions these subplans with respect to the verb. A similar scheme for semantical past time and perfect aspect is found in Fig. 11.

$$t_{PN} \left[\begin{array}{l} \text{SEM} \quad \text{dict} \left[\text{KEY } K \right] \\ \text{STATE} \left[\text{DS} \left[\begin{array}{l} \text{ASP} \quad \text{NEUTR} \\ \text{TIME} \quad \text{PAST} \end{array} \right] \right] \\ \text{TO} \quad \left[\begin{array}{l} \text{SBJ} \quad \left[\text{STATE} \left[\text{GS} \left[\text{VALUE } pred \right] \right] \right] \\ \text{HEAD} \quad \left[\text{STATE} \left[\text{GS } f \left[\begin{array}{l} \text{LEFT} \quad circ * \backslash pred \backslash \\ \text{RIGHT} \quad / circ * / dobj \\ \text{TNS} \quad \text{PASTSIMPLE} \\ \text{GrCl} \quad \text{VT} \end{array} \right] \right] \right] \\ \text{LEX} \quad form(K, f) \\ \text{OBJ} \quad \left[\text{STATE} \left[\text{GS} \left[\text{VALUE } dobj \right] \right] \right] \\ \text{PO} \quad \{ \text{SBJ} < \text{HEAD} < \text{OBJ} \} \end{array} \right] \end{array} \right]$$

Fig. 10. Transition scheme for situations in neutral aspect past time

The transition schemes in Fig. 12 and 13 apply to the situations in neutral (respectively, perfect) aspect and semantical past time if there is a MANNER-circumstantial. E.g., the transitions fitting the neutral aspect scheme place this argument between the SBJ-sort argument and the verb representing the situation if the circumstantial is an adverbial. Otherwise, it places this argument on the right of the OBJ-sort argument. Fig. 14 presents an example of a terminal transition scheme for adverbials without (optional) arguments. The cases of this scheme select the corresponding form and finalize the dependency type assigning $type(q) = q.GS.VALUE$ (i.e. $q.GS.PTN = q.GS.LEFT = q.GS.RIGHT = \varepsilon$). A similar scheme for adjective modifiers is found in Fig. 17.

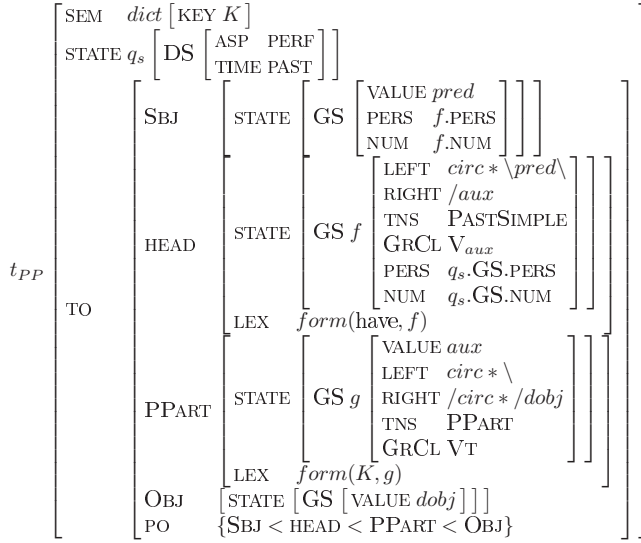


Fig. 11. Transition scheme for perfect aspect situations in past time

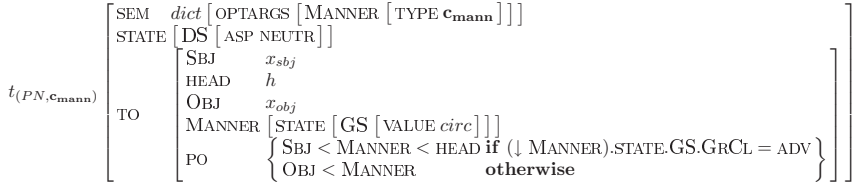


Fig. 12. Transition scheme for manner circumstantials in neutral aspect past time

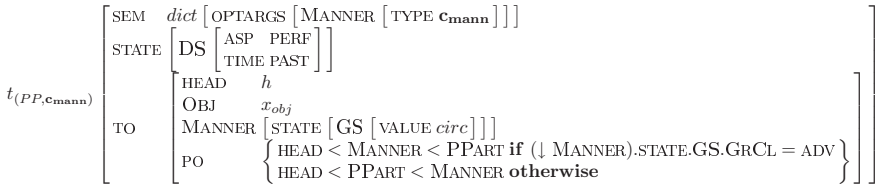


Fig. 13. Transition scheme for manner circumstantials in past perfect

Then follow special cases of transition schemes for nominator type subplans in the positions of SBJ and OBJ-arguments in examples 1,2. In particular, that in Fig. 15 places the definite article on the left of a referenced nominator semanteme, whereas, that in Fig. 16 places between them an optional qualifier, if any.

$$t_{(c, noargs)} \left[\begin{array}{l} \text{SEM } dict \left[\begin{array}{l} \text{KEY } K \\ \text{OPTARGS } \emptyset \\ \text{VTYPE } c \\ \text{VALUE } t \end{array} \right] \\ \text{STATE } [GS] \\ \text{TO } \left[\begin{array}{l} \text{HEAD } \left[\begin{array}{l} \text{STATE } [GS] \\ \text{LEX } form(K, q_c) \end{array} \right] \end{array} \right] \end{array} \right]$$

Fig. 14. Transition scheme for adverbials

$$t_{(n, def)} \left[\begin{array}{l} \text{SEM } dict \left[\begin{array}{l} \text{PROFILE } \uparrow_{ref} < K >_1 \\ \text{VTYPE } n \end{array} \right] \\ \text{TO } \left[\begin{array}{l} \text{DET } \left[\begin{array}{l} \text{STATE } [GS] \\ \text{LEX } form(ART, q_d) \end{array} \right] \\ \text{HEAD } \left[\begin{array}{l} \text{STATE } [GS] \\ \text{LEX } form(K, q_N) \end{array} \right] \\ \text{PO } \{DET < HEAD\} \end{array} \right] \end{array} \right]$$

Fig. 15. Transition scheme for determined nominators

$$t_{(n, Qual)} \left[\begin{array}{l} \text{SEM } dict \left[\begin{array}{l} \text{OPTARGS } [QUAL [TYPE q_{prop}]] \\ \text{HEAD } h \\ \text{DET } d \\ \text{QUAL } [STATE [GS [VALUE modify]]] \\ \text{PO } \{DET < QUAL < HEAD\} \end{array} \right] \end{array} \right]$$

Fig. 16. Transition scheme for qualifiers of determined nominators

$$t_{(q, noargs)} \left[\begin{array}{l} \text{SEM } dict \left[\begin{array}{l} \text{KEY } K \\ \text{OPTARGS } \emptyset \\ \text{VTYPE } q \\ \text{VALUE } t \end{array} \right] \\ \text{STATE } [GS] \\ \text{TO } \left[\begin{array}{l} \text{HEAD } \left[\begin{array}{l} \text{STATE } [GS] \\ \text{LEX } form(K, q_{Adj}) \end{array} \right] \end{array} \right] \end{array} \right]$$

Fig. 17. Transition scheme for adjective modifiers

In Fig. 18 is shown a computation of a DP-DS-converter whose transitions are determined by the transition schemes in Fig. 8-17. In the course of this computation, the sentence *The new key easily opened the door* is generated from the DP DP_{decaus} in Fig. 2 using five transitions:

$$\begin{aligned} t_{decaus-ins} &= t_{(dcl,obj)} = t_{PN} = t_{(PN,c_{mann})} \preceq t_1, \\ t_{(n,Qual)} &= t_{(n,def,K=KEY)} \preceq t_2, & t_{(q,noargs,K=NEW)} &= \preceq t_3, \\ t_{(c,noargs,K=EASILY)} &= \preceq t_4, & t_{(n,def,K=DOOR,OPTARGS=\emptyset)} &= \preceq t_5. \end{aligned}$$

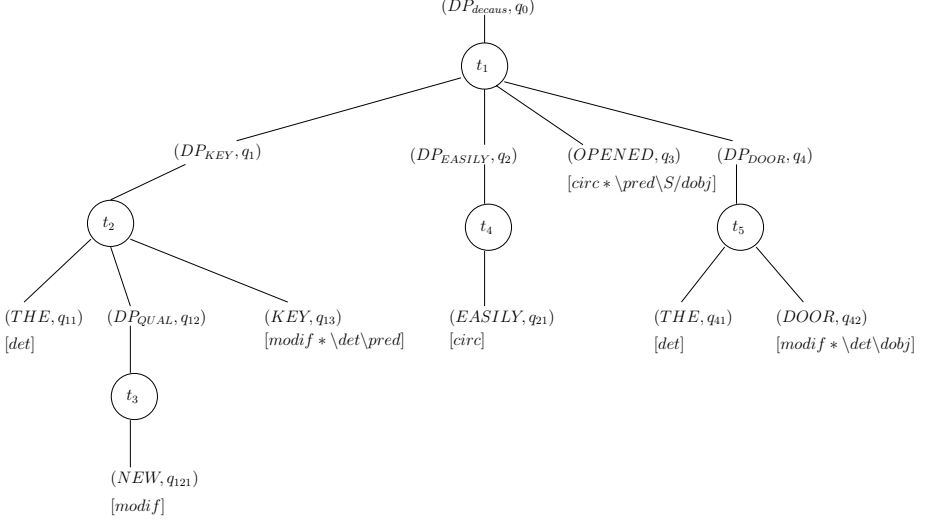


Fig. 18. Generation of *The new key easily opened the door*

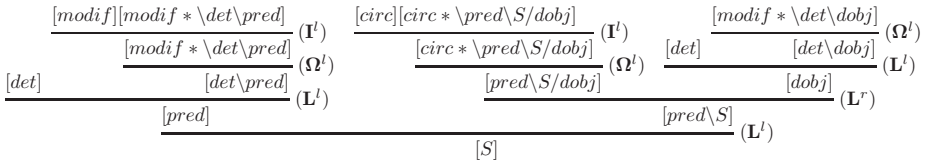


Fig. 19. A correctness proof

A proof of correctness of this computation is shown in Fig. 19. It is easy to see that in this proof is constructed the D-tree in Fig. 3(a).

Properties of DP-DS-conversion architecture. Our English example illustrates the following important properties of the DP-DS-conversion.

1. The transition schemes are either terminal (lexical or referential), or one argument, or they decompose a situation profile into constituents which fix the elimination order of local dependencies' subtypes.

2. The specific transition schemes are not guessed, but uniquely determined (through unification) by the substructures to which they apply.
3. The transitions do not copy or move a material. Unbounded discontinuous dependencies are established through dual valencies in correctness proofs.
4. In the place of a general proof search is used a real time type checking proof (see Corollary 1) witnessing for or against the feasibility of a DP when applied to the type sequence generated from it.

One can see that due to these properties the finite-state conversion and the type checking can be carried out in parallel in the course of DP composition.

6 Conclusion

This work is theoretical. It presents a new compositional generativistic architecture in which the surface syntax is defined in terms of dependency types, the meaning structures uniquely encode the arguments' scopes, surface forms and precedence, and where the meaning structure is converted into the corresponding surface structure using a deterministic fs-tree transducer. An experimental work is needed to make sure that this deterministic architecture can be realized through a system of thematic roles, semantic diatheses, lexical definitions and fs-transitions in a way that the resulting DP-DS-conversion of the set of input DP would be complete with respect to the underlying gCDG.

References

1. Béchet, D., Dikovsky, A., Foret, A.: Dependency structure grammars. In: Blache, P., Stabler, E.P., Busquets, J.V., Moot, R. (eds.) LACL 2005. LNCS (LNAI), vol. 3492, pp. 18–34. Springer, Heidelberg (2005)
2. Bos, J.: Predicate logic unplugged. In: Dekker, P., Stokhof, M. (eds.) Proc. of the 10th Amsterdam Colloquium, pp. 133–142 (1995)
3. Bresnan, J.: Optimal syntax. In: Dekkers, J., van der Leeuw, F., van de Weijer, J. (eds.) Optimal theory: Phonology, Syntax and Acquisition, pp. 334–385. Oxford University Press, Oxford (2000)
4. Chomsky, N.: The Minimalist Program. MIT Press, Cambridge, Massachusetts (1995)
5. Copestake, A., Flickinger, D., Malouf, R., Riehemann, S., Sag, I.: Translation using minimal recursion semantics. In: TMI95. Proc. of the 6th Int. Conf. on Theoretical and Methodological Issues in Machine Translation, Leuven, Belgium (1995)
6. Dekhtyar, M., Dikovsky, A.: Categorical dependency grammars. In: Moortgat, M., Prince, V. (eds.) Proc. of Intern. Conf. on Categorical Grammars, pp. 76–91, Montpellier (2004)
7. Dekhtyar, M., Dikovsky, A.: Generalized categorical dependency grammars (2007) Submitted paper available from <http://www.sciences.univ-nantes.fr/info/perso/permanents/dikovsky/>
8. Dikovsky, A.: Linguistic meaning from the language acquisition perspective. In: FG 2003. Proc. of the 8th Intern. Conf. Formal Grammar 2003, Vienna, Austria, pp. 63–76 (August 2003)

9. Dikovsky, A.: Dependencies as categories. In: Recent Advances in Dependency Grammars. COLING'04 Workshop, pp. 90–97 (2004)
10. Dikovsky, A.: Multimodal categorial dependency grammars (Submitted paper)
11. Dikovsky, A., Smilga, B.: Semantic roles and diatheses for functional discourse plans. In: MTT 2005. Proc. of the 2nd International Conference Meaning-Text Theory, pp. 98–109 (2005)
12. Dowty, D.R.: Thematic proto-roles and argument selection. *Language* 67, 547–619 (1991)
13. Fillmore, C.: Frame semantics. In: Linguistic Society of Korea (ed.) *Linguistics in the Morning Calm*, pp. 111–138, Seoul, Hanshin (1982)
14. Fox, C., Lappin, S.: Underspecified interpretations in a curry-typed representation language. *Journal of Logic and Computation* 15(2), 131–143 (2005)
15. Grimshaw, J.: Projection, heads and optimality. *Linguistic Inquiry* 28, 373–422 (1997)
16. Hovav, M.R., Levin, B.: Deconstructing the thematic hierarchy. In: 38th Seoul Linguistics Forum, December 10–11, Seoul National University, Seoul, Korea (2004)
17. Kamp, H., Reyle, U.: *From Discourse to Logic: An introduction to modeltheoretic semantics, formal logic and Discourse Representation Theory*. Kluwer Academic Publishers, Dordrecht (1993)
18. Padučeva, E.V.: *Dynamic models in lexical semantics*. Jazyki slavjanskoj kul'tury, Moscow (Russ.) (2004)
19. Kaplan, R., Bresnan, J.: Lexical functional grammar: A formal system for grammatical representation. In: Bresnan, J.W. (ed.) *The Mental Representation of Grammatical Relations*, pp. 173–281. MIT Press, Cambridge (1982)
20. Reyle, U.: Dealing with ambiguities by underspecification. *Journal of Semantics* 10(2), 123–179 (1993)
21. Sgall, P., Hajičová, E., Panevová, J.: *The Meaning of the Sentence in its Semantic and Pragmatic Aspects*. Academia, Dordrecht/Reidel/Prague (1986)
22. Sgall, P.: A dependency based specification of topic and focus ii. SMIL. *Journal of Linguistic Calculus* (1-2) 110–140 (1980)
23. Stabler, E.: Derivational minimalism. In: Retoré, C. (ed.) *LACL 1996. LNCS (LNAI)*, vol. 1328, pp. 68–95. Springer, Heidelberg (1997)
24. V.-Shanker, K., Weir, D.J.: The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory* 27, 511–545 (1994)

Pregroup Calculus as a Logic Functor

Annie Foret

IRISA –Rennes 1
Avenue du Général Leclerc
35042 Rennes Cedex - France

Abstract. The concept of pregroup was introduced by Lambek for natural language analysis, with a close link to non-commutative linear logic. We reformulate the pregroup calculus so as to extend it by composition with other logics and calculi. The *cut elimination property* and the *decidability* property of the sequent calculus proposed in the article are shown. Properties of composed calculi are also discussed.

Keywords: Pregroups, Lambek Categorical Grammars, Logic Functor, Cut Elimination.

1 Introduction

The *pregroup* formalism introduced by Lambek [11] is an efficient and elegant alternative to other categorial grammars used for modeling natural languages. It has been applied to various language syntaxes such as English, French, Italian and Turkish, see e.g. [1,6].

This is a *lexicalized* formalism: the rules are fixed, each particular grammar is determined by its *lexicon* assigning a set of type formulas to each word. The set of rules defines a type calculus that has a close link with non-commutative linear logic. Type logical grammars, including pregroups, enjoy the following nice property : a syntactic analysis corresponds to a derivation in the type calculus (a logical proof). Categorical aspects are given for example in [12,13]. In their basic form, pregroup grammars are equivalent to context-free grammars. Some extensions have been proposed : in [7,10] pregroups are enriched with modalities allowing to limit associativity.

The purpose of this paper is to give an adaptation of the basic calculus that allows modular and rich combinations of type expressions, such as those needed to model fine-grained natural language features, while maintaining good properties of the formalism. We aim to keep the cut elimination and decidability of the rule system, to provide modularity and composability in calculus and modelization levels, and to allow conciseness and simplicity in the grammar design.

This adaptation is performed following a “logic functor paradigm”, as in [8], in order to consider new logics as the composition of abstract and reusable components, all logics sharing common specifications, including a subsumption relation. *Logic functors* are functions from logics to logics, (those without parameter are standalone but reusable logics). The “logic functor paradigm”

also results in concrete software: the *LogFun* tool [9] offers both a library of logic functors and a logic composer. For instance, the functor $Prop(\mathcal{A})$ is the classical propositional logic abstracted over its atoms, that allows to replace atoms in the classical propositional logic by formulas of the logic given as parameter.

In this article, we provide a functor for pregroups, called $FPG(\mathcal{A})$ in some sense similar to the functor $Prop(\mathcal{A})$, that can be viewed as parameterized pregroups abstracted over the atoms. One difficulty was to design appropriate rules, generalizing those of original pregroups (recovering pregroups as the particular case) while maintaining cut elimination and decidability of the calculus.

The definition of pregroups (recalled in Section 2) may seem at first very close to such a functor, because a partial order is already assumed on basic types, treated as constants (pregroup grammars allow postulates as a finite number of special axioms of the form $p \vdash q$). In Section 3, a first version of parameterized pregroup calculus, written $\mathcal{S}^{Adj}_{[\mathcal{A}]}$ is defined from Buszkowski's sequent calculus \mathcal{S}^{Adj} for pregroups. Although natural, this version $\mathcal{S}^{Adj}_{[\mathcal{A}]}$ lacks the subformula property, which is a disadvantage to establish decidability. A second version of parameterized pregroup calculus, that we call $\mathcal{S}_{[\mathcal{A}]}$ is then proposed, to avoid this drawback. Section 4 gives the main properties; the key proofs of this article are shown for $\mathcal{S}_{[\mathcal{A}]}$. Detailed proofs can be found in Section 5 and at the end of this paper. Section 6 concludes.

2 Preliminaries

2.1 Pregroups

Definition 1. A pregroup (or PG in short) is a structure $(P, \leq, \cdot, l, r, 1)$ such that $(P, \leq, \cdot, 1)$ is a partially ordered monoid¹ and l, r are two unary operations on P that satisfy for all $a \in P$: $a^l a \leq 1 \leq aa^l$ and $aa^r \leq 1 \leq a^r a$.

Example 1. Groups are a special case a pregroups, for which the left and right adjoint are the same.

Algebraic properties and iterated adjoints. In a pregroup, we have :

$$\begin{aligned} (XY)^l &= Y^l X^l \text{ and } (XY)^r = Y^r X^r; \\ (X^l)^r &= X \text{ and } (X^r)^l = X \\ 1^l &= 1^r = 1 \end{aligned}$$

Iterated left and right adjoints can be written using integer exponents by:

$$p^{(n-1)} = (p^{(n)})^l \text{ and } p^{(n+1)} = (p^{(n)})^r, \text{ we write } p \text{ for } p^{(0)}.$$

Note also that if $X \leq Y$ then $Y^l \leq X^l$ and $Y^r \leq X^r$ (order-reversing)

¹ We briefly recall that a *monoid* is a structure $\langle M, \cdot, 1 \rangle$, such that \cdot is associative and has a neutral element 1 ($\forall x \in M : 1 \cdot x = x \cdot 1 = x$). A partially ordered monoid is a monoid $(M, \cdot, 1)$ with a partial order \leq that satisfies $\forall a, b, c : a \leq b \Rightarrow c \cdot a \leq c \cdot b$ and $a \cdot c \leq b \cdot c$.

2.2 Free Pregroups

We recall definitions and results, mainly from [4,5], we also introduce notations.

Definition 2. Let (P, \leq) be an ordered set of primitive types, we use integer exponents on P :

- the set of atomic types is : $P^{(Z)} = \{p^{(i)} \mid p \in P, i \in Z\}$
- the set of types is $Cat_{(P, \leq)} = (P^{(Z)})^* = \{p_1^{(i_1)} \cdots p_n^{(i_n)} \mid 0 \leq k \leq n, p_k \in P \text{ and } i_k \in Z\}$
- the relation \leq on $Cat_{(P, \leq)}$ is the smallest reflexive and transitive relation, satisfying these conditions for all $p, q \in P$, $X, Y \in Cat_{(P, \leq)}$ and $n \in Z$:
 $Xp^{(n)}p^{(n+1)}Y \leq XY$ (contraction), $XY \leq Xp^{(n+1)}p^{(n)}Y$ (expansion), and $Xp^{(n)}Y \leq Xq^{(n)}Y$, if $p \leq q$ with n even or $q \leq p$ with n odd (induction)
- the free pregroup generated by (P, \leq) is defined on classes [...] modulo \sim such that $X \sim Y$ iff $X \leq Y$ and $Y \leq X$ with $1 = [\epsilon]$, $[x] \cdot [y] = [xy]$, $[x]^l = [x^l]$, $[x]^r = [x^r]$ and $[x] \leq [y]$ iff $x \leq y$ (with $[xy]^l = [y^l x^l]$, $[xy]^r = [y^r x^r]$).

Buszkowski has proposed a sequent calculus, written \mathcal{S}^{Adj} below; he has shown its equivalence with \leq on $Cat_{(P, \leq)}$ as in the above definition. \mathcal{S}^{Adj} also characterizes deductibility in the free pregroup generated by (P, \leq) , where 1 is the empty string.

Definition 3 (The Sequent calculus of Adjoints \mathcal{S}^{Adj}). Let (P, \leq) be an ordered set of primitive types, system \mathcal{S}^{Adj} is made of the following rules, where $p, q \in P$, $n, k \in Z$ and $X, Y, Z \in Cat_{(P, \leq)}$:

$$\begin{array}{c}
 X \leq X \text{ (Id)} \qquad \frac{X \leq Y \quad Y \leq Z}{X \leq Z} \text{ (Cut)} \\
 \\
 \frac{XY \leq Z}{Xp^{(n)}p^{(n+1)}Y \leq Z} \text{ (A}_L\text{)} \quad \frac{X \leq YZ}{X \leq Yp^{(n+1)}p^{(n)}Z} \text{ (A}_R\text{)} \\
 \\
 \frac{Xp^{(k)}Y \leq Z}{Xq^{(k)}Y \leq Z} \text{ (IND}_L\text{)} \quad \frac{X \leq Yp^{(k)}Z}{X \leq Yq^{(k)}Z} \text{ (IND}_R\text{)} \\
 q \leq p \text{ if } k \text{ is even, and } p \leq q \text{ if } k \text{ is odd}
 \end{array}$$

Simple free pregroup. A simple free pregroup is a free pregroup where the order on primitive type is equality.

We mention an interesting link, explored by Kislak and Buszkowski, between pregroups and other types-logical grammars (the reader not familiar with Lambek grammars may consult [2] for details).

Free pregroup interpretation. Let FP denotes the simple free pregroup with P as primitive types. We associate with each formula C of the Lambek Calculus L or its non-associative version NL , an element in FP written $\llbracket C \rrbracket$ as follows: $\llbracket A \rrbracket = A$

if A is a primitive type, $\llbracket C_1 \setminus C_2 \rrbracket = \llbracket C_1 \rrbracket^r \llbracket C_2 \rrbracket$, $\llbracket C_1 / C_2 \rrbracket = \llbracket C_1 \rrbracket \llbracket C_2 \rrbracket^l$, $\llbracket C_1 \bullet C_2 \rrbracket = \llbracket C_1 \rrbracket \llbracket C_2 \rrbracket$. The notation extends to sequents by: $\llbracket A_1, \dots, A_n \rrbracket = \llbracket A_1 \rrbracket \cdots \llbracket A_n \rrbracket$.

The following property states that FP is a model for L (hence for \mathcal{NL}): if $\Gamma \vdash_L C$ then $\llbracket \Gamma \rrbracket \leq_{FP} \llbracket C \rrbracket$.

Note however that the converse does not hold [3].

Cut elimination. As for the Lambek Calculus L or its non-associative version \mathcal{NL} , the cut rule can be eliminated in \mathcal{S}^{Adj} : every derivable inequality has a cut-free derivation [4,5]. This property is important to get a proof-search algorithm based on the system rules.

2.3 Categorical Grammars and Languages

We first consider categorical grammars in general, these are lexicalized grammars, involving an alphabet Σ , a set of *types* Tp that is usually generated from a set of *primitive types* P and given connectives, and a system of rules defining a derivation relation on types.

Definition 4 (Categorical grammar). A categorical grammar is a structure $G = (\Sigma, I, S)$ where: Σ is a finite alphabet (the words in the sentences); $I : \Sigma \mapsto \mathcal{P}^f(T)$ is a function that maps a set of types to each element of Σ (the possible categories of each word); $S \in P$ is the main type associated to correct sentences.

A sentence belongs to the language of G , provided its words can be assigned types that derive S according to the rules of the type calculus.

Definition 5 (Pregroup grammar and language). A pregroup grammar (PG-grammar in short) based on a finite poset (P, \leq) is a categorical grammar $G = (\Sigma, I, s)$, such that: $s \in P$, I assigns types in $Cat_{(P, \leq)}$.

The language $\mathcal{L}(G)$ of a PG-grammar $G = (\Sigma, I, s)$, based on a finite poset (P, \leq) is the set of strings, that can be assigned a string of types deriving s in the free pregroup on (P, \leq) :

$$\mathcal{L}(G) = \{v_1 \dots v_n \in \Sigma^* : \forall i, 1 \leq i \leq n, \exists X_i \in I(v_i) \text{ such that } X_1 \dots X_n \leq s\}$$

The following property, shown in [3], characterizes the generative capacity of PGs.

Proposition 1. The languages generated by PG-grammars based on finite posets are the context-free languages (without the empty string).

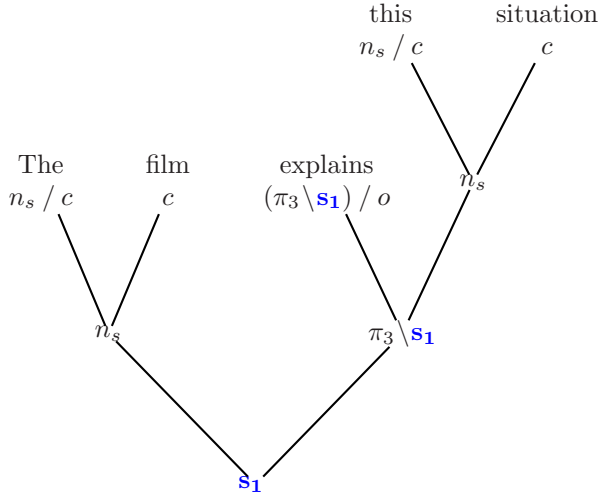
Example 2. We consider the following grammar fragment, with a parsed sentence (the type assignments are below each word, underlinks indicate contractions of types), the string of types derives $s_1 \leq s$:

$$\begin{array}{lcl} \text{sentence:} & \text{The} & \text{film} & \text{explains} & \text{this} & \text{situation} \\ \text{types:} & (n_s \ c^l) & c & (\pi_3^r \ \mathbf{s}_1 \ o^l) & (n_s \ c^l) & c \end{array}$$

using primitive types and order postulates as follows:

$n_s \leq \pi_3$	c = count noun n_s = singular noun phrase $\pi_k = k^{th}$ personal subject pronoun	(film, situation) (John) (π_3 =he/she/it)
$n_s \leq o$	o = direct object	
$s_1 \leq s \leq \bar{s}$	s_1 = statement in present tense s = declarative sentence \bar{s} =indirect sentence	(the tense does not matter)

Using the $\llbracket \dots \rrbracket$ translation to Lambek calculus L, this sentence is also parsed in L:



where types n_s , π_3 , o ... are to be replaced with complex types such that: $n_s \vdash \pi_3$, $n_s \vdash o$, and $s_1 \vdash s \vdash \bar{s}$.

3 Towards a Logic Functor

We propose an adaptation of pregroup following a “logic functor paradigm”, as in [8,9], written *FPG*. In [9], a *logic* \mathcal{A} is viewed as the association of an *abstract syntax* $AS_{\mathcal{A}}$, a *semantics* $S_{\mathcal{A}}$, an *implementation* $P_{\mathcal{A}}$, and a *type* $T_{\mathcal{A}}$ made of a set of properties, including a *subsumption (or entailment) relation*, $\leq_{\mathcal{A}}$. The class of these logics is denoted by \mathbb{L} . A *logic functor* F takes logics of \mathbb{L} as parameters and returns a logic in \mathbb{L} ; it is also viewed as a tuple (AS_F, S_F, P_F, T_F) of functions such that $(AS_{F(L_1, \dots, L_n)}, S_{F(L_1, \dots, L_n)}, P_{F(L_1, \dots, L_n)}, T_{F(L_1, \dots, L_n)}) = (AS_F(AS_{L_1}, \dots, AS_{L_n}), S_F(S_{L_1}, \dots, S_{L_n}), P_F(P_{L_1}, \dots, P_{L_n}), T_F(T_{L_1}, \dots, T_{L_n}))$.

This paper does not deal with all those aspects (the semantics part is left).

The adaptation proposed here can be viewed as parameterized pregroups abstracted over the atoms. We detail the syntax AS_{FPG} and properties for implementation and module composition ; we focus on the subsumption relation and its implementation P_{FPG} based on a sequent calculus formulation.

3.1 Extending Formulas

The logic functor FPG takes as argument a calculus or a logic; when it is applied to a calculus or a logic (written \mathcal{A}) it can be seen as a formulation of Pregroup Calculus, where atoms are replaced by the formulas of its parameter \mathcal{A} , while these subformulas follow the rules of \mathcal{A} . Next definition follows definition 2, where P is replaced by $AS_{\mathcal{A}}$ of the parameter \mathcal{A} .

Definition 6 (Iterated Adjoints on \mathcal{A} : syntax of $FPG(\mathcal{A})$). Let $\mathcal{A} = \langle AS_{\mathcal{A}}, \leq_A \rangle$ where \leq_A is a preorder on $AS_{\mathcal{A}}$. We write $AS_{\mathcal{A}}$ for the set of formulas of the parameter \mathcal{A} , and we define the set of basic types on \mathcal{A} as: $AS_{\mathcal{A}}^{(Z)} = \{p^{(i)} \mid p \in AS_{\mathcal{A}}, i \in Z\}$

the set of types $Cat_{[\mathcal{A}]}$ consists in strings of basic types on $AS_{\mathcal{A}}$.

In the logic functor setting, $Cat_{[\mathcal{A}]}$ is $AS_{FPG}(AS_{\mathcal{A}})$.

3.2 Discussing a First Axiomatization : $\mathcal{S}^{Adj}_{[\mathcal{A}]}$

We consider $\mathcal{S}^{Adj}_{[\mathcal{A}]}$, parameterizing Pregroups on a Sub-Logic \mathcal{A} . We shall in fact consider a second version as discussed after.

Definition 7 ($\mathcal{S}^{Adj}_{[\mathcal{A}]}$, the calculus of adjoints on \mathcal{A}). For X and $Y \in Cat_{[\mathcal{A}]}$, a sequent $X \leq Y$ holds in the calculus of adjoints on \mathcal{A} iff it is deducible in the system $\mathcal{S}^{Adj}_{[\mathcal{A}]}$ axiomatized with rules (Id), (A_L), (IND_{L+}), (A_R), (IND_{R+}) where $p, q \in P$, $n, k \in Z$ and $X, Y, Z \in Cat_{[\mathcal{A}]}$, such that :

$$\frac{Xp^{(k)}Y \leq Z}{Xq^{(k)}Y \leq Z} (IND_{L+}) \quad \frac{X \leq Yp^{(k)}Z}{X \leq Yq^{(k)}Z} (IND_{R+}) \quad \begin{array}{l} \text{with } q \leq_A p \text{ if } k \text{ is even} \\ \text{or } p \leq_A q \text{ if } k \text{ is odd} \\ \text{for rules } (IND_{L+}), (IND_{R+}) \end{array}$$

This version is a direct adaptation of \mathcal{S}^{Adj} (IND_{L+} and IND_{R+} stand for explicitly parameterized versions of IND_L and IND_R). However we shall develop another version, avoiding some drawbacks of IND_{L+} and IND_{R+} ; consider for example IND_{R+} : this rule does not have the subformula property², also : for the given q in the conclusion, the set of $p \in AS_{\mathcal{A}}$ such that $p \leq_A q$ is potentially infinite (in contrast to PG-grammars based on *finite* posets).

3.3 Defining a Second Axiomatization: $\mathcal{S}_{[\mathcal{A}]}$ for FPG

The above version is not appropriate for our purpose. We now propose another version, in view of proof search algorithms: the remaining sections establish its *cut elimination property* and decidability.

Definition 8 ($\mathcal{S}_{[\mathcal{A}]}$ and $\mathcal{S}'_{[\mathcal{A}]}$, a parameterized calculus on \mathcal{A})

- For X and $Y \in Cat_{[\mathcal{A}]}$, a sequent $X \leq Y$ holds in the parameterized calculus $\mathcal{S}'_{[\mathcal{A}]}$, iff this relation is deducible in the following system where $p, q \in P$, $n, k \in Z$ and $X, Y, Z \in Cat_{[\mathcal{A}]}$:

² in the sense that an antecedent of a rule has all its formulas among the subformulas of the conclusion of that rule, this is often a key property to show the decidability of an inference system ; for example, if $\forall i \in \mathbb{N} : p_i \leq_A q$, to test whether $X \leq q$ may involve $X \leq p_i$ for any $i \in \mathbb{N}$ as antecedent, where p_i does not need to occur in X .

$$\begin{array}{c}
 \frac{a \leq_A b, \text{ if } m \text{ is even}}{a^{(m)} \leq b^{(m)}} (Sub) \quad \frac{XY \leq Z \quad a \leq_A b, \text{ if } m \text{ is even}}{Xa^{(m)}b^{(m+1)}Y \leq Z} (A_{L+}) \\
 \\
 \frac{b \leq_A a, \text{ if } m \text{ is odd}}{a^{(m)} \leq b^{(m)}} (Sub) \quad \frac{XY \leq Z \quad b \leq_A a, \text{ if } m \text{ is odd}}{Xa^{(m)}b^{(m+1)}Y \leq Z} (A_{L+}) \\
 \\
 X \leq X \quad (Id) \quad \frac{Xa^{(m+1)} \leq Y}{X \leq Ya^{(m)}} (I_R) \quad \frac{X \leq Y \quad Y \leq Z}{X \leq Z} (Cut)
 \end{array}$$

- $\mathcal{S}_{[\mathcal{A}]}$ denotes the same system as $\mathcal{S}'_{[\mathcal{A}]}$ without the cut rule.
- For ease of proofs, we define a condensed presentation for rules (Sub) and (A_{L+}):

$$\frac{a^{(m)} \leq_A b^{(m)}}{a^{(m)} \leq b^{(m)}} (Sub) \quad \frac{XY \leq Z \quad a^{(m)} \leq_A b^{(m)}}{Xa^{(m)}b^{(m+1)}Y \leq Z} (A_{L+})$$

where $a^{(m)} \leq_A b^{(m)}$ stands for $a \leq_A b$ if m is even, $b \leq_A a$ if m is odd

- Pregroup grammars on \mathcal{A} and their language are defined as in definition 5, but using $\mathcal{S}_{[\mathcal{A}]}$ instead.

In the logic functor setting, $\leq_{FPG}(\mathcal{A})$ stands for \leq in $\mathcal{S}_{[\mathcal{A}]}$.

Note 1. The rules in $\mathcal{S}^{Adj}_{[\mathcal{A}]}$ (in the non-condensed presentation) have the subformula property, except I_R . However I_R enjoys a pseudo-subformula property, since this rule can have only one antecedent ; we can also write a^{m+1} as $a^{m'}$ and write a^m in the conclusion as $a^{m'-1} = (a^{m'})^{(-1)}$.

4 Extending Pregroup Calculus to a Logic Functor : Properties

Let $\mathcal{S}_{[\mathcal{A}]}$ denote the system axiomatized by (Id), (Sub), (A_{L+}) (I_R), and let $\mathcal{S}'_{[\mathcal{A}]}$ denote the system axiomatized by **Cut** and the rules in $\mathcal{S}_{[\mathcal{A}]}$.

We shall write $a^{(m)} \leq_A b^{(m)}$ for $a \leq_A b$ if m even, $b \leq_A a$ if m is odd.

4.1 Properties of $\mathcal{S}_{[\mathcal{A}]}$

Let $\mathcal{A} = \langle \mathcal{AS}_{\mathcal{A}}, \leq_A \rangle$ where \leq_A is a preorder on $\mathcal{AS}_{\mathcal{A}}$ (assumed in all this subsection)

The main property of Cut elimination is based on the following two lemmas 1, 2.

The lemmas show several analogues of rules or of there inverse (IND_{R-} is a weaker form of IND_R).

Lemma 1. For $\mathcal{S} = \mathcal{S}_{[\mathcal{A}]}$ and $\mathcal{S} = \mathcal{S}'_{[\mathcal{A}]}$, for $a, b \in AS_{\mathcal{A}}$ and $X, Y \in Cat_{[\mathcal{A}]}$:

$[I_R^{-1}]$: rule I_R is reversible: If $X \leq Y a^{(m)}$ is provable in \mathcal{S} then $X a^{(m+1)} \leq Y$ is also provable in \mathcal{S}

$[IND_L]$: If $X b^{(m)} Y \leq Z$ in \mathcal{S} and $a^{(m)} \leq_A b^{(m)}$ then $X a^{(m)} Y \leq Z$ in \mathcal{S}

$[IND_{R-}]$: if $X \leq a^{(m)}$ in \mathcal{S} and $a^{(m)} \leq_A b^{(m)}$ then $X \leq b^{(m)}$ in \mathcal{S}

Lemma 2. For $\mathcal{S} = \mathcal{S}_{[\mathcal{A}]}$ and $\mathcal{S} = \mathcal{S}'_{[\mathcal{A}]}$, we have (for $a, b \in AS_{\mathcal{A}}$ and $X, Y \in Cat_{[\mathcal{A}]}$):

if $X a^{(m+1)} b^{(m)} Y \leq Z$ in \mathcal{S} and $a^{(m)} \leq_A b^{(m)}$ then $XY \leq Z$ in \mathcal{S}

Theorem 3 (Cut elimination). $\mathcal{S}_{[\mathcal{A}]}$ and $\mathcal{S}'_{[\mathcal{A}]}$ are equivalent: for all types $X, Y \in Cat_{[\mathcal{A}]}$, $X \leq Y$ is provable in $\mathcal{S}_{[\mathcal{A}]}$ iff it is provable in $\mathcal{S}'_{[\mathcal{A}]}$.

Proof : see next section and the proofs of lemmas (detailed in Appendix).

Theorem 5 relates $\mathcal{S}_{[\mathcal{A}]}$ to PG and $\mathcal{S}^{Adj}_{[\mathcal{A}]}$, it is a corollary of Cut elimination for $\mathcal{S}'_{[\mathcal{A}]}$, Lemma1 $[IND_L]$, and Lemma4:

Lemma 4. The $[IND_R]$ and $[A_R]$ rules (also $[A_{R+}]$) are admissible in $\mathcal{S}'_{[\mathcal{A}]}$:

$[IND_R]$: if $X \leq Y a^{(k)} Z$ is provable in $\mathcal{S}'_{[\mathcal{A}]}$ and $a^{(k)} \leq_A b^{(k)}$ ($a, b \in AS_{\mathcal{A}}$)

then $X \leq Y b^{(k)} Z$ is also provable in $\mathcal{S}'_{[\mathcal{A}]}$

$[A_R]$: if $X \leq Y Z$ in $\mathcal{S}'_{[\mathcal{A}]}$ then $X \leq Y a^{(n+1)} a^{(n)} Z$ in $\mathcal{S}'_{[\mathcal{A}]}$, for each $a \in AS_{\mathcal{A}}$.

$[A_{R+}]$: if $X \leq Y Z$ in $\mathcal{S}'_{[\mathcal{A}]}$, and $a^{(m)} \leq_A b^{(m)}$ then $X \leq Y a^{(m+1)} b^{(m)} Z$ in $\mathcal{S}'_{[\mathcal{A}]}$

Theorem 5 (Pregroups as a particular case). (i) Let PG denote a pregroup with order \leq on primitive types P : $X \leq Y$ is provable in \mathcal{S}^{Adj} iff it is derivable in $\mathcal{S}_{[\mathcal{A}]}$, taking \leq_A as \leq

(ii) More generally, $\mathcal{S}_{[\mathcal{A}]}$ and $\mathcal{S}^{Adj}_{[\mathcal{A}]}$ are equivalent : for all types $X, Y \in Cat_{[\mathcal{A}]}$, $X \leq Y$ is provable in $\mathcal{S}_{[\mathcal{A}]}$ iff it is provable in $\mathcal{S}^{Adj}_{[\mathcal{A}]}$.

4.2 Properties of Composed Calculii

Proposition 2. Let $\mathcal{A} = \langle AS_{\mathcal{A}}, \leq_{\mathcal{A}} \rangle$ where \leq_A is a preorder. If \leq_A is a decidable calculus, then $\mathcal{S}_{[\mathcal{A}]}$ and $\mathcal{S}'_{[\mathcal{A}]}$ (applied to \mathcal{A}) are decidable.

This is clear, using the cut elimination theorem, and the subformula property for $\mathcal{S}_{[\mathcal{A}]}$ (with the special case of I_R , having only one possible antecedent).

Proposition 3. The language generated by a PG -grammar G on a Sub-Logic $\mathcal{A} = \langle AS_{\mathcal{A}}, \leq_{\mathcal{A}} \rangle$ where \leq_A is a preorder (G based on the deduction system $\mathcal{S}_{[\mathcal{A}]}$ or equivalently $\mathcal{S}'_{[\mathcal{A}]}$) is a context-free language.

This can be shown by associating to G a free PG -grammar G_{PG} , obtained by replacing, in the assignment, all subformulas F that belong to \mathcal{A} by a new constant c_F , with $c_F \leq c_{F'}$ whenever $F \leq_A F'$: G_{PG} generates the same language as G .

5 Cut Elimination in $\mathcal{S}'_{[\mathcal{A}]}$: Proof Details

This proof assumes lemmas 1, 2, detailed in the Appendix. Clearly a proof in $\mathcal{S}_{[\mathcal{A}]}$ is also a proof in $\mathcal{S}'_{[\mathcal{A}]}$; to show the converse, we proceed by induction on the number of **Cut** and the length of a derivation \mathcal{D} in $\mathcal{S}'_{[\mathcal{A}]}$, ending in **Cut**:

\mathcal{D} with its final cut rule is:

$$\boxed{\frac{\gamma_l \left\{ \begin{array}{c} \vdots \\ \text{---} R_l \\ X \leq \textcolor{red}{Y} \end{array} \quad \gamma_r \left\{ \begin{array}{c} \vdots \\ \text{---} R_r \\ \textcolor{red}{Y} \leq Z \end{array} \right.}{X \leq Z} \text{Cut}}$$

- If R_l is the axiom rule, the last rule (cut) can be suppressed since R_r has the same conclusion as \mathcal{D} . If R_r is the axiom rule, the last rule (cut) can also be suppressed since R_l has the same conclusion as \mathcal{D} . We now assume that neither R_l nor R_r is the axiom rule.
- If R_l is the **Cut** rule, the induction hypothesis applies to γ_l , this **Cut** can be suppressed, in a proof γ'_l , and the final **Cut** can be suppressed in this deduction. If R_r is the **Cut** rule, we proceed similarly.

We now assume that neither γ_l or γ_r has a **Cut** rule.

- We shall prove Cut elimination for Y *simple*, that is of the form $p^{(u)}$, with $p \in P$ (we consider 1 separately); the general case is proved by induction on Y , using I_R^{-1} and Lemma2:

-if $Y = Y_1 p^{(u)}$: $\mathcal{D} \mapsto$

$$\boxed{\frac{\frac{\frac{\vdots}{X \leq Y = Y_1 p^{(u)}}{X p^{(u+1)} \leq Y_1} I_R^{-1} \quad \frac{\frac{\vdots}{Y = Y_1 p^{(u)} \leq Z}}{Y_1 \leq Z p^{(u-1)}} I_R}{X p^{(u+1)} \leq Z p^{(u-1)}} \text{rec. on } Y}{X p^{(u+1)} p^{(u)} \leq Z} I_R^{-1}{X \leq Z} \text{Lemma2}}$$

- We consider the remaining possibilities when Y is simple, for R_l (left part) and R_r (right part): (no A_{L+} on the right as R_r , no I_R on the left as R_l , for Y simple) these cases are detailed blow.

R_l	R_r	method
Sub	I_R	lemma1 $[IND_L]$
A_{L+}	Sub	lemma1 $[IND_{R-}]$
A_{L+}	I_R	permute R_l with cut
Sub	Sub	transitivity of \leq_A

- When $Y = 1$ (empty string), the only case is $[R_l, R_r] = [A_{L+}, I_R]$ and can be treated as for Y simple (permute R_l with **Cut**)
- Cut elimination details for Y simple

$$[Sub, I_R] \quad \frac{\frac{a^{(m)} \leq_A b^{(m)}}{X = a^{(m)} \leq b^{(m)} = Y} \text{Sub} \quad \mathcal{D}' \left\{ \frac{Yp^{(u+1)} \leq Z_1}{Y = b^{(m)} \leq Z_1p^{(u)} = Z} \begin{array}{l} \vdots \\ I_R \end{array} \right.}{X \leq Z} \text{Cut}$$

$$\mapsto \frac{\mathcal{D}' \left\{ \frac{Yp^{(u+1)} \leq Z_1}{Y = b^{(m)} \leq Z_1p^{(u)} = Z} \begin{array}{l} \vdots \\ I_R \end{array} \right. \quad a^{(m)} \leq_A b^{(m)}}{X = a^{(m)} \leq Z_1p^{(u)} = Z} \text{[IND}_L\text{]}$$

(for Y simple)

$$[A_{L+}, Sub] \quad \frac{\mathcal{D}' \left\{ \frac{X_1X_2 \leq Y = a^{(m)} \quad c^{(n)} \leq_A d^{(n)}}{X = X_1c^{(n)}d^{(n+1)}X_2 \leq Y = a^{(m)}} \begin{array}{l} \vdots \\ A_{L+} \end{array} \right. \quad \frac{a^{(m)} \leq_A b^{(m)}}{Y = a^{(m)} \leq b^{(m)} = Z} \text{Sub}}{X \leq Z} \text{Cut}$$

$$\mapsto \frac{\mathcal{D}' \left\{ \frac{X_1X_2 \leq Y = a^{(m)} \quad c^{(n)} \leq_A d^{(n)}}{X = X_1c^{(n)}d^{(n+1)}X_2 \leq Y = a^{(m)}} \begin{array}{l} \vdots \\ A_{L+} \end{array} \right. \quad a^{(m)} \leq_A b^{(m)}}{X \leq b^{(m)} = Z} \text{[IND}_{R-}\text{]}$$

(for Y simple or 1)

$$[A_{L+}, I_R] \quad \frac{\frac{X_1X_2 \leq Y \quad a^{(m)} \leq_A b^{(m)}}{X = X_1a^{(m)}b^{(m+1)}X_2 \leq Y} \begin{array}{l} \vdots \\ A_{L+} \end{array} \quad \mathcal{D}' \left\{ \frac{Yp^{(u+1)} \leq Z_1}{Y \leq Z_1p^{(u)} = Z} \begin{array}{l} \vdots \\ I_R \end{array} \right.}{X \leq Z} \text{Cut}$$

$$\mapsto \frac{\frac{X_1X_2 \leq Y \quad \mathcal{D}' \left\{ \frac{Yp^{(u+1)} \leq Z_1}{Y \leq Z_1p^{(u)} = Z} \begin{array}{l} \vdots \\ I_R \end{array} \right.}{X_1X_2 \leq Z_1p^{(u)} = Z} \text{Cut} \quad a^{(m)} \leq_A b^{(m)}}{X = X_1a^{(m)}b^{(m+1)}X_2 \leq Z} A_{L+}$$

6 Conclusion and Perspectives

We have reformulated and proposed to extend the pregroup calculus, so as to allow its composition with other logics and calculi. The *cut elimination property* and the *decidability* property of this proposed formulation have been shown. It is thus ready to be implemented as a decision procedure and a parsing algorithm. ■

Integrating pregroups as a logic functor component should allow to customize such a calculus to actual applications in a rather easy way. Parameterized pregroups may take a calculus as parameter, providing more structure to the basic types. But they may also in turn become the argument of another logic functor, such as Prop available in [9]. An interesting example is the extension with intersection types, such that $t \leq c \wedge d$ iff $t \leq c$ and $t \leq d$ going beyond context-free languages, as already explained by Lambek. As more logic functors become defined and implemented, more interesting compositions can be considered, that can enrich the type formulas of categorial grammars themselves, in a clear way, both formally and practically. This may give another view, other reformulations and investigations on these frameworks, with experimental tools at the same time for the composed logics and customized calculi.

Acknowledgments. Thanks to S. Ferré, O. Ridoux and the LIS team for useful discussions on logic functors.

References

1. Bargelli, D., Lambek, J.: An Algebraic Approach to French Sentence Structure. In: de Groote, P., Morill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, Springer, Heidelberg (2001)
2. Buszkowski, W.: Mathematical linguistics and proof theory. In: van Benthem, J., ter Meulen, A. (eds.) Handbook of Logic and Language, pp. 683–736. Elsevier, North-Holland (1997)
3. Buszkowski, W.: Lambek grammars based on pregroups. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, Springer, Heidelberg (2001)
4. Buszkowski, W.: Cut elimination for the Lambek calculus of adjoints. In: New Perspectives in Logic and Formal Linguistics, Proceedings Vth ROMA Workshop, Bulzoni Editore
5. Buszkowski, W.: Sequent systems for compact bilinear logic. *Mathematical Logic Quarterly* 49(5), 467–474 (2003)
6. Casadio, C., Lambek, J.: An Algebraic Analysis of Clitic Pronouns in Italian. In: de Groote, P., Morill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, Springer, Heidelberg (2001)
7. Fadda, Mario,: Towards flexible pregroup grammars. In: New Perspectives in Logic and Formal Linguistics, pp. 95–112. Bulzoni Editore, Roma (2002)
8. Ferré, S., Ridoux, O.: A Framework for Developing Embeddable Customized Logic. In: Pettorossi, A. (ed.) LOPSTR 2001. LNCS, vol. 2372, pp. 191–215. Springer, Heidelberg (2002)
9. Ferré, S., Ridoux, O.: Logic Functors : a Toolbox of Components for Building Customized and Embeddable Logics. Research report, n° RR-5871, Inria (March 2006) <http://www.inria.fr/rrrt/rr-5871.html>
10. Kislak-Malinowska, Aleksandra, Pregroups with modalities, FG2006: the 11th conference on Formal Grammar, Malaga, Spain (July 2006)
11. Lambek, J.: Type grammars revisited. In: Lecomte, A., Perrier, G., Lamarche, F. (eds.) LACL 1997. LNCS (LNAI), vol. 1582, pp. 22–24. Springer, Heidelberg (1999)

12. Preller, A.: Category Theoretical Semantics for Pregroup Grammars. In: Blache, P., Stabler, E., Busquets, J.V., Moot, R. (eds.) LACL 2005. LNCS (LNAI), vol. 3492, Springer, Heidelberg (2005)
13. Preller, A., Lambek, J.: Free compact 2-categories. Mathematical Structures for Computer Sciences (January 2007)

A Appendix: Lemma in $\mathcal{S}_{[\mathcal{A}]}$ and $\mathcal{S}'_{[\mathcal{A}]}$: Proof Details

A.1 Lemma1 [I_R^{-1}]: Proof by Induction on Derivations of:

if $\boxed{\frac{\vdots}{X \leq Y a^{(m)}} R_l}$ in \mathcal{S} then $\boxed{\frac{\vdots}{X a^{(m+1)} \leq Y}}$ in \mathcal{S}

- Case R_l is Id : $X = Y a^{(m)}$ then A_{L+} applies (\leq_A refl.)

$$\boxed{\frac{Y \leq Y \quad a^{(m)} \leq_A a^{(m)}}{Y a^{(m)} a^{(m+1)} \leq Y} A_{L+}}$$

- Case R_l is Sub : $Y = 1, X = c^{(m)}$ with $c^{(m)} \leq_A a^{(m)}$ then A_{L+} applies

$$\boxed{\frac{Y \leq Y \quad c^{(m)} \leq_A a^{(m)}}{c^{(m)} a^{(m+1)} = Y c^{(m)} a^{(m+1)} \leq Y} A_{L+}}$$

- Case R_l is A_{L+}

$$\boxed{\frac{\frac{\vdots}{X_1 \leq Y a^{(m)}} \quad c^{(n)} \leq_A d^{(n)}}{X_1 c^{(n)} d^{(n+1)} \leq Y a^{(m)}} A_{L+}}$$

\mapsto

$$\boxed{\frac{\frac{\frac{\vdots}{X_1 \leq Y a^{(m)}}}{X_1 a^{(m+1)} \leq Y} \text{rec.} \quad c^{(n)} \leq_A d^{(n)}}{X_1 c^{(n)} d^{(n+1)} a^{(m+1)} \leq Y} A_{L+}}$$

- Case R_l is I_R , the antecedent is the desired sequent $X a^{(m+1)} \leq Y$
- (Case R_l is Cut)

$$\boxed{\frac{\frac{\vdots}{X \leq Z} \quad \frac{\vdots}{Z \leq Y a^{(m)}}}{X \leq Y a^{(m)}} Cut}$$

\mapsto

$$\boxed{\frac{\frac{\frac{\vdots}{X \leq Z}}{X a^{(m+1)} a^{(m+2)} \leq Z} A_{L+} \quad \frac{\frac{\vdots}{Z \leq Y a^{(m)}}}{Z a^{(m+1)} \leq Y} \text{rec.}}{X a^{(m+1)} \leq Z a^{(m+1)}} I_R \quad Cut}$$

■

A.2 Lemma1 [IND_L] We Prove That:

$$\text{if } \boxed{\frac{\vdots}{Xb^{(m)}Y \leq Z} R_l} \text{ in } \mathcal{S} \text{ and } a^{(m)} \leq_A b^{(m)} \text{ then } \boxed{\frac{\vdots}{Xa^{(m)}Y \leq Z}} \text{ in } \mathcal{S}12$$

- Case R_l is Id , then $Z = Xb^{(m)}Y$, we proceed by induction on Y

$$\begin{array}{c} \text{-if } Y \text{ is empty } \swarrow \\ \boxed{\frac{X \leq X \quad a^{(m)} \leq_A b^{(m)}}{Xa^{(m)}b^{(m+1)} \leq X} A_{L+} \quad \frac{Xa^{(m)}b^{(m+1)} \leq X}{Xa^{(m)} \leq Xb^{(m)} = Z} I_R} \end{array} \quad \begin{array}{c} \text{-if } Y = Y_1p^{(u)} \swarrow \\ \boxed{\frac{Xb^{(m)}Y_1 \leq Xb^{(m)}Y_1 \quad a^{(m)} \leq_A b^{(m)}}{Xa^{(m)}Y_1 \leq Xb^{(m)}Y_1} \text{rec. on } Y \quad p \leq_A p} \\ \frac{Xa^{(m)}Y_1p^{(u)}p^{(u+1)} \leq Xb^{(m)}Y_1}{Xa^{(m)}Y_1p^{(u)} \leq Xb^{(m)}Y_1p^{(u)} = Z} A_{L+} \quad I_R \end{array}$$

- Case R_l is Sub , then X and Y are empty, $Z = c^{(m)}$, with $b^{(m)} \leq c^{(m)}$, we get the conclusion $a^{(m)} \leq c^{(m)}$ by transitivity of \leq_A .
- Case R_l is A_{L+}

$$\boxed{\frac{\vdots}{X_1X_2 \leq Z} \quad \frac{c^{(n)} \leq_A d^{(n)}}{Xb^{(m)}Y = X_1c^{(n)}d^{(n+1)}X_2 \leq Z} A_{L+}} \mapsto \boxed{\frac{\vdots}{X_1X_2 \leq Z} \quad \frac{a^{(m)} \leq_A b^{(m)}}{Xa^{(m)}X_4X_2 \leq Z} \text{rec.} \quad \frac{c^{(n)} \leq_A d^{(n)}}{Xa^{(m)}Y = Xa^{(m)}X_4c^{(n)}d^{(n+1)}X_2 \leq Z} A_{L+}}$$

-if $b^{(m)} \in X_2$, the transformation is similar ;

-if $X = X_1$, $b^{(m)} = c^{(n)}$, $Y = d^{(n+1)}X_2$, we use A_{L+} for $a^{(m)} \leq_A d^{(n)}$ (transitivity) and get: $X_1a^{(m)}d^{(n+1)}X_2 = Xa^{(m)}Y \leq Z$

-if $X = X_1c^{(n)}$, $b^{(m)} = d^{(n+1)}$, $Y = X_2$, we use A_{L+} for $c^{(n)} \leq_A a^{(m-1)}$ and get: $X_1c^{(n)}a^{(m)}X_2 = Xa^{(m)}Y \leq Z$

- Case R_l is I_R , then $Z = Z_1p^{(u)}$ and the antecedent is $Xb^{(m)}Yp^{(u+1)} \leq Z_1$, by rec. we get $Xa^{(m)}Yp^{(u+1)} \leq Z_1$, then the desired sequent by I_R : $Xa^{(m)}Y \leq Z_1p^{(u)}$
- (Case R_l is Cut)

$$\boxed{\frac{\vdots}{Xb^{(m)}Y \leq Z'} \quad \frac{\vdots}{Z' \leq Z} \quad Cut} \mapsto \boxed{\frac{\vdots}{Xb^{(m)}Y \leq Z'} \quad \frac{a^{(m)} \leq_A b^{(m)}}{Xa^{(m)}Y \leq Z'} \text{rec.} \quad \frac{\vdots}{Z' \leq Z} \quad Cut} \quad \blacksquare$$

A.3 Lemma2 : We Prove, Using Lemma1 That

if $\boxed{\frac{\vdots}{Xa^{(m+1)}b^{(m)}Y \leq Z} R_l}$ in \mathcal{S} and $a^{(m)} \leq_A b^{(m)}$ then $\boxed{\frac{\vdots}{XY \leq Z}}$ in \mathcal{S}

we proceed by induction on Y and I_R^{-1}, I_R :

- if $Y = Y_1p^{(u)}$: by I_R^{-1} , $Xa^{(m+1)}b^{(m)}Y \leq Z$ implies $Xa^{(m+1)}b^{(m)}Y_1 \leq Zp^{(u-1)}$, by induction on Y , this gives $XY_1 \leq Zp^{(u-1)}$, then by I_R , $XY = XY_1p^{(u)} \leq Z$.
A proof is given below for $Y = \emptyset$;

- Case R_l is Id ($Y = \emptyset$), $Z = Xa^{(m+1)}b^{(m)} \mapsto$

$$\boxed{\begin{array}{c} X \leq X \quad (Id) \quad a^{(m)} \leq_A b^{(m)} \\ \hline Xb^{(m+1)}a^{(m+2)} \leq X \\ \hline Xb^{(m+1)} \leq Xa^{(m+1)} \\ \hline X \leq Xa^{(m+1)}b^{(m)} = Z \end{array} \begin{array}{c} A_{L+} \\ \\ I_R \\ I_R \end{array}}$$
- Case R_l is Sub ($Y = \emptyset$), impossible (complex formula on the left by R_l)
- Case R_l is A_{L+} ($Y = \emptyset$)

$$\boxed{\frac{\mathcal{D}' : X_1X_2 \leq Z \quad c^{(n)} \leq_A d^{(n)}}{Xa^{(m+1)}b^{(m)} = X_1c^{(n)}d^{(n+1)}X_2 \leq Z} A_{L+}}$$

- if $X_2 = X_2'a^{(m+1)}b^{(m)}$

with $X = X_1c^{(n)}d^{(n+1)}X_2'$

$$\boxed{\frac{\mathcal{D}' : X_1X_2'a^{(m+1)}b^{(m)} \leq Z \quad a^{(m)} \leq_A b^{(m)}}{X_1X_2' \leq Z \quad c^{(n)} \leq_A d^{(n)}} \begin{array}{c} \\ \\ A_{L+} \end{array} \begin{array}{c} \\ \\ \text{rec.} \end{array}}$$

- if $X_2 \neq \emptyset$ and $X_2 = b^{(m)}$ then $a^{(m+1)} = d^{(n+1)}$ and $X = X_1c^{(n)}$; the antecedent is $X_1X_2 = X_1b^{(m)} \leq Z$; we have $c^{(n)} \leq_A d^{(n)} = a^{(m)} \leq_A b^{(m)}$, hence by Lemma1 [IND_L], using $c^{(n)} \leq_A b^{(m)}$ (transitivity): $X = X_1c^{(n)} \leq Z$
- if $X_2 = \emptyset$, $X = X_1$, but exponents $m+1 = n$ and $m = n+1$ are impossible
- Case R_l is I_R ($Y = \emptyset$), then $Z = Z_1p^{(u)}$ and the antecedent is $Xa^{(m+1)}b^{(m)}p^{(u+1)} \leq Z_1$, by rec. we get $Xp^{(u+1)} \leq Z_1$, finally by I_R : $X \leq Z_1p^{(u)} = Z$
- (Case R_l is Cut)($Y = \emptyset$)

$$\boxed{\frac{\frac{\vdots}{Xa^{(m+1)}b^{(m)} = X_1 \leq Z'}{\quad} \quad \frac{\vdots}{Z' \leq Z}}{Xa^{(m+1)}b^{(m)} = X_1 \leq Z} \text{Cut}}$$

\mapsto

$$\boxed{\frac{\frac{Xa^{(m+1)}b^{(m)} = X_1 \leq Z'}{\quad} \quad \frac{\vdots}{Z' \leq Z}}{X \leq Z'} \begin{array}{c} \\ \\ \text{rec.} \end{array} \quad \frac{\vdots}{Z' \leq Z} \quad \text{Cut} \quad \frac{\quad}{X \leq Z}}$$

A.4 Lemma1 [IND_{R-}] Proof by Induction, According to R_l of:

if $\boxed{\frac{\vdots}{X \leq a^{(m)}} R_l}$ in \mathcal{S} and $a^{(m)} \leq_A b^{(m)}$ then $\boxed{\frac{\vdots}{X \leq b^{(m)}}}$ in \mathcal{S}

- Case R_l is (Id) , then $X = a^{(m)}$, the result $X \leq b^{(m)}$ is an application of Sub .
- Case R_l is (Sub) , the antecedent is $X = c^{(n)} \leq_A a^{(m)}$, where $c^{(n)} \leq_A a^{(m)}$ and $a^{(m)} \leq_A b^{(m)}$, the result $X = c^{(n)} \leq b^{(m)}$, comes from the transitivity of \leq_A and from Sub .
- Case R_l is (A_{L+}) ,

$$\boxed{
 \begin{array}{c}
 \vdots \\
 X_1 X_2 \leq a^{(m)} \\
 \hline
 X = X_1 c^{(n)} d^{(n+1)} X_2 \leq a^{(m)} \quad A_{L+}
 \end{array}
 } \mapsto \boxed{
 \begin{array}{c}
 \vdots \\
 X_1 X_2 \leq a^{(m)} \quad a^{(m)} \leq_A b^{(m)} \\
 \hline
 X_1 X_2 \leq b^{(m)} \quad \textcolor{red}{rec.} \quad c^{(n)} \leq_A d^{(n)} \\
 \hline
 X = X_1 c^{(n)} d^{(n+1)} X_2 \leq b^{(m)} \quad A_{L+}
 \end{array}
 }$$

- Case R_l is (I_R) , the antecedent is $X a^{(m+1)} \leq 1$ to which we apply **Lemma1** IND_L using $a^{(m)} \leq_A b^{(m)}$: $X b^{(m+1)} \leq 1$ we then apply I_R and get $X \leq b^{(m)}$
- (Case R_l is Cut), with antecedent $X \leq Y$ and $Y \leq a^{(m)}$, we apply the induction hypothesis to $Y \leq a^{(m)}$, then **Cut** on Y ■

A.5 Proof of Lemma4: $[IND_R]$ and $[A_R]$ (also $[A_{R+}]$) Are Admissible Rules in $\mathcal{S}'_{[A]}$

- Proof of $[IND_R]$ using Cut \mapsto

$$\boxed{
 \begin{array}{c}
 \vdots \\
 X \leq Y a^{(m)} Z \\
 \hline
 \begin{array}{c}
 Y b^{(m)} Z \leq Y b^{(m)} Z \quad (Id) \quad a^{(m)} \leq_A b^{(m)} \\
 \hline
 Y a^{(m)} Z \leq Y b^{(m)} Z \quad \textcolor{red}{[IND_L]}
 \end{array} \\
 \hline
 X \leq Y b^{(m)} Z \quad Cut
 \end{array}
 }$$

- Proof of $[A_{R+}]$ using Cut \mapsto

$$\boxed{
 \begin{array}{c}
 \vdots \\
 X \leq Y_1 Y_2 \\
 \hline
 \begin{array}{c}
 Y_1 a^{(m+1)} b^{(m)} Y_2 \leq Y_1 a^{(m+1)} b^{(m)} Y_2 \quad (Id) \quad a^{(m)} \leq_A b^{(m)} \\
 \hline
 Y_1 Y_2 \leq Y_1 a^{(m+1)} b^{(m)} Y_2 \quad \textcolor{red}{Lemma2}
 \end{array} \\
 \hline
 X \leq Y_1 a^{(m+1)} b^{(m)} Y_2 \quad Cut
 \end{array}
 }$$

A Formal Calculus for Informal Equality with Binding

Murdoch J. Gabbay¹ and Aad Mathijssen²

¹ School of Mathematical and Computer Sciences, Heriot-Watt University
Edinburgh EH14 4AS, Scotland, Great Britain

`murdoch.gabbay@gmail.com`

² Department of Mathematics and Computer Science, Eindhoven University of
Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

`A.H.J.Mathijssen@tue.nl`

Abstract. In informal mathematical usage we often reason using languages with binding. We usually find ourselves placing capture-avoidance constraints on where variables can and cannot occur free. We describe a logical derivation system which allows a direct formalisation of such assertions, along with a direct formalisation of their constraints. We base our logic on equality, probably the simplest available judgement form. In spite of this, we can axiomatise systems of logic and computation such as first-order logic or the lambda-calculus in a very direct and natural way. We investigate the theory of derivations, prove a suitable semantics sound and complete, and discuss existing and future research.

1 Introduction

The theory of equalities $t = u$ is perhaps the simplest of all foundational logical theories. Informal specification of logic and computation often involves equalities with *binding* and subject to conditions about *freshness*. For example:

- λ -calculus: $\lambda x.(tx) = t$ if x is fresh for t
- π -calculus: $\nu x.(P \mid Q) = P \mid \nu x.Q$ if x is fresh for P
- First-order logic: $\forall x.(\phi \supset \psi) = \phi \supset \forall x.\psi$ if x is fresh for ϕ

and for any binder $\zeta \in \{\lambda, \nu, \forall\}$:

- Substitution: $(\zeta y.u)[x \mapsto t] = \zeta y.(u[x \mapsto t])$ if y is fresh for t

It is not hard to extend this short list with many more examples.

In the equalities above there are *two* levels of variable; x and y are variables of the system being axiomatised, we call these **object-level** variables; t , u , P , Q , ϕ , and ψ range over terms of that syntax, we call them **meta-level** variables. Unfortunately these equalities are subject to freshness side-conditions which make them something other than ‘just equalities’.

Ways have been developed to attain the simplicity and power of the theory of equality between terms. For example we can work with combinators [1] or

combinatory logic [2], cylindric algebra [3], higher-order algebra [4] or higher-order logic [5]. Roughly speaking: combinatory approaches reject object-level variables entirely; cylindric approaches also reject them as independent syntactic entities but enrich the language of term-formers to regain some lost expressivity; higher-order approaches model the difference between the two using a hierarchy of types. These approaches do not permit a *direct* representation of the two-level structure which informal syntax displays in terms such as $\lambda x.t$ or $\forall x.\phi$.

In this paper we describe **Nominal Algebra**. This is a logic based on equality which *embraces* the two-level variable structure by representing it directly in its syntax. Informal equivalences can be represented as axioms almost symbol-for-symbol. For example the equalities above are represented by:

- λ -calculus: $a\#X \vdash \lambda[a](Xa) = X$
- π -calculus: $a\#X \vdash \nu[a](X \mid Y) = X \mid \nu[a]Y$
- First-order logic: $a\#X \vdash \forall[a](X \supset Y) = X \supset \forall[a]Y$
- Substitution: $b\#X \vdash (\zeta[b]Y)[a \mapsto X] = \zeta[b](Y[a \mapsto X])$

Here a and b are distinct *atoms* representing object-level variables; X and Y are *unknowns* representing meta-level variables. Each equality is equipped with a *freshness condition* of the form $a\#X$ that guarantees that X can only be instantiated to a term for which a is fresh. The rest of this paper makes this formal.

In Sect. 2 we introduce the syntax of nominal algebra. In Sect. 3 we define a notion of derivation. In Sect. 4 we define a notion of validity, and in Sect. 5 we prove soundness and completeness. Sections 6 and 7 discuss related work and draw conclusions.

2 Syntax

Fix a countably infinite collection of **atoms** a, b, c, \dots representing object-level variables. We shall use a *permutative convention* that a, b, c, \dots range permutatively over atoms, so that for example a and b are always distinct. Also, fix a countably infinite collection of **unknowns** X, Y, Z, \dots representing meta-level variables. Fix **term-formers** f to each of which is associated some unique **arity** n which is a nonnegative number. Assume these collections are disjoint.

There is no proper sorting system so it will be possible to write ‘silly’ terms. There is no problem in principle with extending the system with a sort or type system if convenient, perhaps along the lines of other work [6, 7].

Let π range over **(finitely supported) permutations**. So π bijects atoms with themselves and there is a finite set of atoms S such that $\pi(a) = a$ for all atoms *not* in S . Write **Id** for the identity permutation such that **Id**(a) = a always. Write $\pi \circ \pi'$ for functional composition and write π^{-1} for inverse. This makes permutations into a group — write \mathbb{P} for the set of all permutations.

Then nominal terms t, u, v are inductively defined by:

$$t ::= a \mid \pi \cdot X \mid [a]t \mid f(t_1, \dots, t_n)$$

We call $[a]t$ an *abstractor*; it uniformly represents the ‘ $x.t$ ’ or $x.\phi$ ’ part of expressions such as ‘ $\lambda x.t$ ’ or ‘ $\forall x.\phi$ ’.

We call $\pi \cdot X$ a **moderated unknown**. We write $\text{Id} \cdot X$ just as X , for brevity. In $\pi \cdot X$, X will get substituted for a term and then π will permute the atoms in that term; see Sect. 3. This notion is grounded in semantics [8] and permits a succinct treatment of α -renaming atoms (see CORE below and [9]).

A **signature** Σ is some set of term-formers with their arities. For example:

- $\{\text{lam} : 1, \text{app} : 2\}$ is a signature for the λ -calculus; we indicate arities with a colon as a convenient shorthand. When we define axioms for the λ -calculus, we shall extend this signature with a term-former for representing capture-avoiding substitution.

We generally sugar $\text{lam}([a]t)$ to $\lambda[a]t$ and $\text{app}(t, u)$ to tu .

- $\{\supset : 2, \forall : 1, \approx : 2, \perp : 0\}$ is a signature for first-order logic with equality (the symbol for equality inside the logic is \approx).

We sugar $\supset(\phi, \psi)$ to $\phi \supset \psi$, $\forall([a]\phi)$ to $\forall[a]\phi$ and $\approx(t, u)$ to $t \approx u$.

Write $t \equiv u$ for **syntactic identity** of terms. *There is no quotient by abstraction* so for example $[a]a \not\equiv [b]b$. Write $a \in t$ for ‘ a **occurs in (the syntax of)** t ’, Occurrence is literal, e.g. $a \in [a]a$ and $a \in \pi \cdot X$ when $\pi(a) \neq a$. Similarly write $a \notin t$ for ‘ a does not occur in the syntax of t ’.

A **freshness (assertion)** is a pair $a\#t$ of an atom a and a term t . An **equality (assertion)** is a pair $t = u$ where t and u are terms. Call a freshness of the form $a\#X$ (so $t \equiv X$) **primitive**. Write Δ for a finite set of *primitive* freshnesses and call it a **freshness context**. We drop set brackets in freshness contexts, e.g., writing $a\#X, b\#Y$ for $\{a\#X, b\#Y\}$.

Nominal algebra has two **judgement forms**, a pair $\Delta \vdash a\#t$ of a freshness context and a freshness assertion, and a pair $\Delta \vdash t = u$ of a freshness context and an equality assertion. We generally write $\Delta \vdash A$ for an arbitrary judgement form, i.e. A is an equality or freshness. We may write $\emptyset \vdash A$ as $\vdash A$.

A **theory** $\mathsf{T} = (\Sigma, Ax)$ is a pair of a signature Σ and a possibly infinite set of *equality* judgement forms Ax in that signature; we call them the **axioms**.

Here are some nominal algebra theories:

- LAM has signature $\{\text{lam} : 1, \text{app} : 2, \text{sub} : 2\}$ and two axioms

$$\begin{array}{ll} (\beta) & \vdash (\lambda[a]Y)X = Y[a \mapsto X] \\ (\eta) & a\#X \vdash \lambda[a](Xa) = X \end{array}$$

where we sugar $\text{sub}([a]t, u)$ to $t[a \mapsto u]$. LAM on its own is not terribly useful because we need to give **sub** the correct behaviour:

- SUB has axioms

$$\begin{array}{ll} (\text{var} \mapsto) & \vdash a[a \mapsto X] = X \\ (\# \mapsto) & a\#Y \vdash Y[a \mapsto X] = Y \\ (\mathbf{f} \mapsto) & \vdash \mathbf{f}(Y_1, \dots, Y_n)[a \mapsto X] = \mathbf{f}(Y_1[a \mapsto X], \dots, Y_n[a \mapsto X]) \\ (\mathbf{abs} \mapsto) & b\#X \vdash ([b]Y)[a \mapsto X] = [b](Y[a \mapsto X]) \\ (\mathbf{ren} \mapsto) & b\#Y \vdash Y[a \mapsto b] = (b \ a) \cdot Y \end{array}$$

Axiom ($\mathbf{f} \mapsto$) represents three axioms, one for each of $\mathbf{f} \in \{\text{lam}, \text{app}, \text{sub}\}$. A theory of substitution for a different signature would have a suitable axiom for each term-former \mathbf{f} . Note the heavy use of freshness side-conditions to manage the relationship between atoms and unknowns.

But there is *one more axiom*. We would like $\lambda[a]a$ to be equal to $\lambda[b]b$. In other words we want α -equivalence:

- CORE has just one axiom

$$(\mathbf{perm}) \quad a \# X, b \# X \vdash (b \ a) \cdot X = X$$

Lemma 3.2 below shows that this axiom with the derivation rules of nominal algebra give the native notion of α -equivalence on nominal terms from previous work [9].

See [10] for an axiomatisation of first-order logic. Similar development for other systems with binding, such as System F [11] and the π -calculus [12], should also be possible.

3 A Derivation System

Now we need a notion of derivation which represents freshness assumptions on meta-variables, and permits axioms involving abstraction and conditioned on freshness assumptions, just like we do in informal reasoning.

We define a **permutation action** $\pi \cdot t$ by:

$$\begin{aligned} \pi \cdot a &\equiv \pi(a) & \pi \cdot (\pi' \cdot X) &\equiv (\pi \circ \pi') \cdot X & \pi \cdot [a]t &\equiv [\pi(a)](\pi \cdot t) \\ \pi \cdot \mathbf{f}(t_1, \dots, t_n) &\equiv \mathbf{f}(\pi \cdot t_1, \dots, \pi \cdot t_n) \end{aligned}$$

A **substitution** σ is a finitely supported function from unknowns to terms. Here, finite support means: for some finite set of unknowns $\sigma(X) \not\equiv X$, and for all other unknowns $\sigma(X) \equiv X$. Write $[t_1/X_1, \dots, t_n/X_n]$ for the substitution σ such that $\sigma(X_i) \equiv t_i$ and $\sigma(Y) \equiv Y$, for all $Y \not\equiv X_i$, $1 \leq i \leq n$.

We can define a **substitution action** $t\sigma$ on terms by:

$$\begin{aligned} a\sigma &\equiv a & (\pi \cdot X)\sigma &\equiv \pi \cdot \sigma(X) & ([a]t)\sigma &\equiv [a](t\sigma) \\ \mathbf{f}(t_1, \dots, t_n)\sigma &\equiv \mathbf{f}(t_1\sigma, \dots, t_n\sigma) \end{aligned}$$

Substitution for an unknown does not avoid capture with abstraction, for example $([a]X)[a/X] \equiv [a]a$. This is designed to mirror informal practice, where instantiation of meta-variables does not avoid capture with binding.

Extend notation for permutation and substitution action to freshness contexts Δ pointwise to the terms it contains. It reduces parentheses to give substitution a higher priority than permutation and abstraction, so we do. The following **commutation** is easy to prove [9, 6]:

Lemma 3.1. $\pi \cdot t\sigma \equiv (\pi \cdot t)\sigma$.

Define **derivability** on freshnesses (in some signature Σ) by the rules in Fig. 1. Here f ranges over the term-formers of Σ , and in accordance with our permutative convention a and b range over *distinct* atoms. Write $\Delta \vdash a\#t$ when $a\#t$ may be derived from Δ .

Define **derivability** on equalities (between terms in the signature of T) by the rules in Fig. 2. Here **(fr)** is subject to a condition that $a \notin t, u, \Delta$ and the square brackets denote discharge of assumptions in natural deduction style [13]. Write $\Delta \vdash_{\tau} t = u$ when we may derive $t = u$ from Δ , using the signature from theory T and admitting only the axioms it contains. We write $\Delta \vdash_{\tau} A$ as a convenient shorthand for $\Delta \vdash_{\tau} t = u$ when A is $t = u$ and $\Delta \vdash a\#t$ when A is $a\#t$ (subscript T disappears in the freshness case).

$$\frac{}{a\#b} (\#ab) \quad \frac{\pi^{-1}(a)\#X}{a\#\pi \cdot X} (\#X) \quad \frac{}{a\#[a]t} (\#[a]) \quad \frac{a\#t}{a\#[b]t} (\#[b]) \quad \frac{a\#t_1 \cdots a\#t_n}{a\#f(t_1, \dots, t_n)} (\#f)$$

Fig. 1. Derivation rules for freshness

$$\begin{array}{c} \frac{}{t=t} (\text{refl}) \quad \frac{t=u}{u=t} (\text{symm}) \quad \frac{t=u \quad u=v}{t=v} (\text{tran}) \quad \frac{\pi \cdot \Delta\sigma}{\pi \cdot t\sigma = \pi \cdot u\sigma} (\text{ax}_{\Delta \vdash t=u}) \\[10pt] \frac{t=u}{[a]t = [a]u} (\text{cong}[]) \quad \frac{t=u}{f(\dots, t, \dots) = f(\dots, u, \dots)} (\text{cong}f) \quad \frac{[a\#X_1, \dots, a\#X_n] \quad \Delta \quad \vdots \quad t=u}{t=u} (\text{fr}) \end{array}$$

Fig. 2. Derivation rules for equality

$(\text{ax}_{\Delta \vdash t=u})$ allows us to permutatively rename atoms and to instantiate unknowns. This gives the effect that atoms in axioms can be understood to range over *any* (distinct) atoms, and unknowns can be understood to range over *any* terms. So these derivations

$$\frac{}{(\lambda[b]a)b = a[b \mapsto b]} (\text{ax}_{\beta}) \quad \frac{}{(\lambda[b]b)a = b[b \mapsto a]} (\text{ax}_{\beta})$$

are valid in theory LAM. The right derivation also shows that substitution of terms for unknowns does not avoid capture, reflecting informal practice.

The use of the $(\text{ax}_{\Delta \vdash t=u})$ rule introduces new proof obligations on the freshness side-conditions in Δ , as the following derivations show:

$$\frac{\frac{}{a\#b} (\#ab)}{\lambda[a](ba) = b} (\text{ax}_{\eta}) \quad \frac{a\#a}{\lambda[a](aa) = a} (\text{ax}_{\eta})$$

The left derivation is valid but the right one is not, because $a\#a$ is not derivable.

Note that we cannot conclude $([c]Y)[c \mapsto X] = [c](Y[c \mapsto X])$ by an application of $(\mathbf{ax}_{\# \mapsto})$ (even if $c \# X$ is derivable), since permutations are bijective: there is no π such that both $\pi(a) = c$ and $\pi(b) = c$.

We conclude these examples with two derivations in theory **CORE**:

$$\frac{\frac{\overline{a \# b}}{a \# [b]b} (\# \blacksquare \mathbf{b}) \quad \frac{\overline{b \# [b]b}}{b \# [b]b} (\# \blacksquare \mathbf{a})}{[a]a = [b]b} (\mathbf{ax}_{\text{perm}}) \quad \frac{\frac{\overline{a \# X}}{a \# [b]X} (\# \blacksquare \mathbf{b}) \quad \frac{\overline{b \# [b]X}}{b \# [b]X} (\# \blacksquare \mathbf{a})}{[a](b \ a) \cdot X = [b]X} (\mathbf{ax}_{\text{perm}})$$

So $\vdash_{\text{CORE}} [a]a = [b]b$ and $a \# X \vdash_{\text{CORE}} [a](b \ a) \cdot X = [b]X$. To see that the instances of (\mathbf{perm}) are valid, we note that $[a]a \equiv (b \ a) \cdot [b]b$ and $[a](b \ a) \cdot X \equiv (b \ a) \cdot [b]X$.

CORE is a novel and pleasingly succinct way to algebraically express the syntax-directed equality given to nominal terms [9]:

Lemma 3.2. $\Delta \vdash_{\text{CORE}} t = u$ if and only if $\Delta \vdash t = u$ holds in the sense of [9, 6].

Proof. By induction on derivations of $\Delta \vdash_{\text{CORE}}$ and $\Delta \vdash t = u$. \square

We will *always* assume that theories contain the axiom (\mathbf{perm}) from theory **CORE**.¹

The rule (\mathbf{fr}) introduces a fresh atom into the derivation. To illustrate the extra power this gives, we show that $X[a \mapsto a] = X$ is derivable in **SUB**:

$$\frac{\frac{\frac{\overline{a \# [a]X}}{a \# [a]X} (\# \blacksquare \mathbf{a}) \quad \frac{\frac{[b \# X]^1}{b \# [a]X} (\# \blacksquare \mathbf{b})}{[b](b \ a) \cdot X = [a]X} (\mathbf{perm})}{[a]X = [b](b \ a) \cdot X} (\mathbf{symm}) \quad \frac{\frac{[b \# X]^1}{a \# (b \ a) \cdot X} (\# \mathbf{X})}{((b \ a) \cdot X)[b \mapsto a] = X} (\mathbf{ax}_{\text{ren} \mapsto})}{X[a \mapsto a] = ((b \ a) \cdot X)[b \mapsto a]} (\mathbf{cong f}) \quad \frac{X[a \mapsto a] = X}{X[a \mapsto a] = X} (\mathbf{fr})^1}{X[a \mapsto a] = X} (\mathbf{tran})$$

In the above derivation, the superscript number one ¹ is an annotation associating the instance of the rule (\mathbf{fr}) with the assumption it discharges in the derivation. Furthermore, the instance of axiom $(\mathbf{ren} \mapsto)$ is valid since we have used the fact that $X \equiv (a \ b) \cdot (b \ a) \cdot X$ in the right-hand side of the equation.

We cannot derive $X[a \mapsto a] = X$ *without* (\mathbf{fr}) ; intuitively this is because to α -rename a so that we can use $(\mathbf{ren} \mapsto)$, we need an atom fresh for X . The tools to make this argument formal are elsewhere [14].

Note that (\mathbf{fr}) mirrors the generation of a fresh name in rules such as the \forall right-introduction rule ‘from $\Gamma \vdash \phi$ derive $\Gamma \vdash \forall x. \phi$ provided x is not free in Γ ’.

We conclude this section with some proof-theoretical results. We can permute atoms in freshnesses and equations without changing the freshness contexts:

¹ Equivalently we could add $(\mathbf{ax}_{\text{perm}})$ as a derivation rule $\frac{a \# t \quad b \# t}{(a \ b) \cdot t = t}$.

Theorem 3.3. *For any permutation π' , if $\Delta \vdash_{\tau} A$ then $\Delta \vdash_{\tau} \pi' \cdot A$.*

Proof. By induction on derivations. For $(\#X)$ we note that permutations are *bijective* and $\pi'(a) \# \pi' \circ \pi \cdot X$ if and only if $\pi^{-1}(a) \# X$. For (\mathbf{fr}) we might have to rename the freshly chosen atom a if $\pi'(a) \neq a$.² \square

We can substitute terms for unknowns provided those terms violate no freshness assumptions made on the unknowns:

Theorem 3.4. *If $\Delta \vdash_{\tau} A$ then $\Delta' \vdash_{\tau} A\sigma$ for all Δ' such that $\Delta' \vdash_{\tau} \Delta\sigma$.*

Proof. Natural deduction derivations are such that the conclusion of one derivation may be ‘plugged in’ to an assumption in another derivation. For $(\#X)$ we use Theorem 3.3. For (\mathbf{fr}) we might have to rename the freshly chosen atom a if it is mentioned by σ (see footnote 2). \square

4 Semantics

A model of a nominal algebra theory \mathbb{T} is a nominal set which interprets the term-formers so as to make the axioms valid. We use *nominal* sets [8] because they permit a direct semantic interpretation of freshness judgements $a \# x$ and permutations $\pi \cdot x$, an interpretation which is not conveniently definable on ‘ordinary’ sets.

A nominal set is a pair (\mathbb{X}, \cdot) of a(n ordinary) set \mathbb{X} with a group action by \mathbb{P} such that each $x \in \mathbb{X}$ has **finite support**.³ This means that there is some *finite* set of atoms $\{a_1, \dots, a_n\}$ such that for any π if $\pi(a_i) = a_i$ for each $1 \leq i \leq n$, then $\pi \cdot x = x$. It is a fact ([8, Proposition 3.4]) that a unique least such set of atoms exists, we call it the **support** of x . We write $a \# x$ when a is not in the support of x , and call a **fresh for** x . For example:

- The set $\mathbb{A} = \{a, b, c, \dots\}$ of atoms with action $\pi \cdot a = \pi(a)$ is a nominal set; the support of $a \in \mathbb{A}$ is $\{a\}$, so $b \# a$ but not $a \# a$.
- The powerset $\mathcal{P}(\mathbb{A}) = \{U \mid U \subseteq \mathbb{A}\}$ of \mathbb{A} with action $\pi \cdot U = \{\pi \cdot u \mid u \in U\}$, is *not* a nominal set; $\{a_1, a_3, a_5, \dots\} \in \mathcal{P}(\mathbb{A})$ does not have finite support, since for no finite set of atoms is it the case that all permutations fixing that set map $\{a_1, a_3, a_5, \dots\}$ to itself. Note that the support of $\mathbb{A} \in \mathcal{P}(\mathbb{A})$ is \emptyset , so $a \# \mathbb{A}$ for any a .
- If \mathbb{X} and \mathbb{Y} are nominal sets write $\mathbb{X} \times \mathbb{Y}$ for $\{(x, y) \mid x \in \mathbb{X}, y \in \mathbb{Y}\}$ with action $\pi \cdot (x, y) = (\pi \cdot x, \pi \cdot y)$. This is also a nominal set; the support of $(x, y) \in \mathbb{X} \times \mathbb{Y}$ is the union of the supports of x and y .

It is useful to write \mathbb{X}^n for $\overbrace{\mathbb{X} \times \dots \times \mathbb{X}}^n$, so we do.

² It is easy to prove that the resulting ‘name-clash-avoiding’ derivation *is* a derivation, and we can use induction on *depth* of derivations to preserve the inductive hypothesis. However, we have used the principle of meta-level Fraenkel-Mostowski equivariance [8], which lets us rename atoms permutatively *without* losing structural inductive properties, so our structural induction is actually perfectly valid.

³ \cdot is a function $\mathbb{P} \times \mathbb{X} \rightarrow \mathbb{X}$ such that $\mathbf{Id} \cdot x = x$ and $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$.

We assume these permutation actions on these sets henceforth.

Lemma 4.1. *For any nominal set \mathbb{X} and element $x \in \mathbb{X}$, if $a \# x$ and $b \# x$ then $(a \ b) \cdot x = x$.*

Proof. Since $a, b \notin \text{supp}(x)$, $(a \ b)(c) = c$ for every $c \in \text{supp}(x)$. \square

Functions $f \in \mathbb{X} \rightarrow \mathbb{Y}$ (on the underlying sets) have a natural conjugation permutation action given by $(\pi \cdot f)(x) = \pi \cdot (f(\pi^{-1} \cdot x))$. Call f **equivariant** if $\pi \cdot (f(x)) = f(\pi \cdot x)$ always.

Lemma 4.2. *For any nominal sets \mathbb{X}, \mathbb{Y} , equivariant function $f \in \mathbb{X} \rightarrow \mathbb{Y}$ and $x \in \mathbb{X}$, if $a \# x$ then $a \# f(x)$.*

Proof. By an elementary calculation using the fact that $\pi \cdot (f(x)) = f(\pi \cdot x)$. \square

We can now give a semantics to nominal algebra theories. An **interpretation** $\llbracket _ \rrbracket$ of a signature is a nominal set \mathbb{T} with *equivariant* functions

- $\llbracket _ \rrbracket_{\mathbb{T}} \in \mathbb{A} \rightarrow \mathbb{T}$ to interpret atoms;
- $\llbracket _ \rrbracket_{-} \in \mathbb{A} \times \mathbb{T} \rightarrow \mathbb{T}$ such that $a \# [a]x$ always, to interpret abstraction;
- $\llbracket f \rrbracket \in \mathbb{T}^n \rightarrow \mathbb{T}$ for each term-former $f : n$, to interpret term-formers.

A **valuation** ς maps unknowns X to elements $\varsigma(X) \in \mathbb{T}$. The pair of an interpretation and a valuation extend easily to terms $\llbracket t \rrbracket_{\varsigma}$:

$$\begin{aligned} \llbracket a \rrbracket_{\varsigma} &= \llbracket a \rrbracket_{\mathbb{T}} & \llbracket \pi \cdot X \rrbracket_{\varsigma} &= \pi \cdot \varsigma(X) & \llbracket [a]t \rrbracket_{\varsigma} &= [a]\llbracket t \rrbracket_{\varsigma} \\ \llbracket f(t_1, \dots, t_n) \rrbracket_{\varsigma} &= \llbracket f \rrbracket(\llbracket t_1 \rrbracket_{\varsigma}, \dots, \llbracket t_n \rrbracket_{\varsigma}) \end{aligned}$$

This extends to a notion of **validity** for our judgement forms:

$$\begin{aligned} \llbracket a \# t \rrbracket_{\varsigma} \text{ (is valid) when } a \# \llbracket t \rrbracket_{\varsigma} & \quad \llbracket t = u \rrbracket_{\varsigma} \text{ when } \llbracket t \rrbracket_{\varsigma} = \llbracket u \rrbracket_{\varsigma} \\ \llbracket \Delta \rrbracket_{\varsigma} \text{ when } a \# \varsigma(X) \text{ for each } a \# X \in \Delta & \quad \llbracket \Delta \vdash A \rrbracket_{\varsigma} \text{ when } \llbracket \Delta \rrbracket_{\varsigma} \text{ implies } \llbracket A \rrbracket_{\varsigma} \\ \llbracket \Delta \vdash A \rrbracket \text{ when } \llbracket \Delta \vdash A \rrbracket_{\varsigma} \text{ for all valuations } \varsigma \end{aligned}$$

A **model** of a theory \mathbb{T} is an interpretation $\llbracket _ \rrbracket$ such that $\llbracket \Delta \vdash t = u \rrbracket$ for all axioms $\Delta \vdash t = u$ of \mathbb{T} — we say the model **validates the axioms**.

Write $\Delta \models_{\mathbb{T}} A$ when $\llbracket \Delta \vdash A \rrbracket$ for all models $\llbracket _ \rrbracket$ of \mathbb{T} .

5 Soundness and Completeness

Derivability of freshness and equality is sound for the semantics presented in the previous section.

Theorem 5.1 (Soundness). *If $\Delta \vdash_{\mathbb{T}} A$ then $\Delta \models_{\mathbb{T}} A$.*

Proof. Let $\llbracket _ \rrbracket$ be a model of \mathbb{T} . We must show that if A is derived from Δ then $\llbracket \Delta \rrbracket_{\varsigma}$ implies $\llbracket A \rrbracket_{\varsigma}$ for any valuation ς . We work by induction on derivation rules.

- (**#ab**). By definition, $\llbracket a\#b \rrbracket_\varsigma$ when $a\#\llbracket b \rrbracket_\mathbb{T}$. By Lemma 4.2 this follows from $a\#b$, which is a standard property of (semantic) freshness.
- (**#X**). By properties of the group of permutations, $a\#\pi \cdot x$ when $\pi^{-1}(a)\#x$.
- (**#[a]**). $a\#[a]x$ for any $x \in \mathbb{T}$ by construction.
- (**#[b]**). By Lemma 4.2, $a\#x$ implies $a\#[b]x$.
- (**#f**). Take any $f : n$. If $a\#\llbracket t_i \rrbracket_\varsigma$ for $1 \leq i \leq n$ then $a\#\llbracket f \rrbracket(\llbracket t_1 \rrbracket_\varsigma, \dots, \llbracket t_n \rrbracket_\varsigma)$ follows using Lemma 4.2.
- (**refl**), (**symm**), (**tran**), (**cong[]**), (**cong f**). By properties of equality.
- (**ax $_{\Delta' \vdash t=u}$**). This follows from the definition of what it is to be a model, and the fact that $\llbracket \pi \cdot t\sigma \rrbracket_\varsigma = \pi \cdot \llbracket t \rrbracket_{\varsigma'}$ where $\varsigma'(X) = \llbracket \sigma(X) \rrbracket_\varsigma$.
- (**fr**). Suppose there is a derivation of $t = u$ from $\Delta, a\#X_1, \dots, a\#X_n$, and suppose that $a \notin t, u, \Delta$. If a is amongst the atoms in the support of $\varsigma(X_i)$ for any X_i , rename a in the derivation to avoid this. By inductive hypothesis $\llbracket t \rrbracket_\varsigma = \llbracket u \rrbracket_\varsigma$ follows from $\llbracket \Delta, a\#X_1, \dots, a\#X_n \rrbracket_\varsigma$. So $\llbracket t \rrbracket_\varsigma = \llbracket u \rrbracket_\varsigma$ as required. \square

In order to show completeness of derivability of equality, we need some technical machinery. Given a theory \mathbb{T} we construct a **term model** $\llbracket _ \rrbracket^\mathbb{T}$ of \mathbb{T} as follows:

- For each $n > 0$ introduce a term-former $\mathbf{d}_n : n$.
- Take as \mathbb{T} the set of closed terms of sort \mathbb{T} (terms without unknowns) in the enriched signature, quotiented by provable equality, with the permutation action given pointwise.
- Take $\llbracket a \rrbracket_\mathbb{T} = \{t \mid \vdash_\mathbb{T} t = a, t \text{ closed}\}$, $\llbracket a \rrbracket x = \{t \mid \vdash_\mathbb{T} t = [a]u, t \text{ closed}, u \in x\}$, and for each term-former $f : n$ take as $\llbracket f \rrbracket^\mathbb{T}$ the function defined by

$$\llbracket f \rrbracket^\mathbb{T}(x_1, \dots, x_n) = \{t \mid \vdash_\mathbb{T} t = f(t_1, \dots, t_n), t \text{ closed}, t_i \in x_i\}.$$

We have to enrich the signature with the \mathbf{d}_n to ensure the term model has enough elements; term-formers must be interpreted by *equivariant* functions so the usual method of adding constants \mathbf{c} (0-ary term-formers) would add only elements such that $\vdash_\mathbb{T} \pi \cdot \mathbf{c} = \mathbf{c}$ always, and this does not suffice. This idea goes back to [15].

It is possible to prove by some elementary calculations that the definition above is well-defined and an interpretation; that is, that \mathbb{T} is a nominal set, that $\llbracket _ \rrbracket_\mathbb{T}$, $\llbracket _ \rrbracket$, and all $\llbracket f \rrbracket^\mathbb{T}$ are equivariant, and that $a\#\llbracket [a]t \rrbracket^\mathbb{T}$ always.

Lemma 5.2. *If $a\#\llbracket t \rrbracket_\varsigma^\mathbb{T}$ then there is some $t' \in \llbracket t \rrbracket_\varsigma^\mathbb{T}$ such that $\vdash_\mathbb{T} a\#t'$.*

Proof. Take b fresh (so b does not occur in t or ς). Then $b\#\llbracket t \rrbracket_\varsigma^\mathbb{T}$. Since also $a\#\llbracket t \rrbracket_\varsigma^\mathbb{T}$ we obtain $(b\ a) \cdot \llbracket t \rrbracket_\varsigma^\mathbb{T} = \llbracket t \rrbracket_\varsigma^\mathbb{T}$ by Lemma 4.1. But then also $\vdash_\mathbb{T} (b\ a) \cdot t = t$, by construction. Take $t' \equiv (b\ a) \cdot t$. \square

To show why Lemma 5.2 is not trivial, consider a theory **IOTA** with one axiom $\vdash a = b$. It is easy to verify that $a\#\llbracket a \rrbracket_\varsigma^\mathbb{T}$ (since $\llbracket a \rrbracket_\varsigma^\mathbb{T} = \mathbb{A}$) but $a\#a$ is not derivable. Of course $a = b$ and $a\#b$ are derivable. Similarly in **LAM** it is a fact that $a\#\llbracket (\lambda[a]b)a \rrbracket_\varsigma^\mathbb{T}$ but $a\#(\lambda[a]b)a$ is not derivable; of course $(\lambda[a]b)a = b$ and $a\#b$ are derivable.

Theorem 5.3. *The term model $\llbracket _ \rrbracket^\tau$ of T is a model.*

Proof. We show that if $\Delta \vdash t = u$ is an axiom of T then $\llbracket \Delta \vdash t = u \rrbracket_\varsigma^\tau$ is valid for any ς . By definition, we must show that if $a \# \varsigma(X)$ for each $a \# X \in \Delta$, then $\llbracket t \rrbracket_\varsigma^\tau = \llbracket u \rrbracket_\varsigma^\tau$. Use Lemma 5.2 to choose a term $t' \in \varsigma(X)$ such that $\vdash a \# t'$ for each $a \# X \in \Delta$. By $(\mathbf{ax}_{\Delta \vdash t = u})$, taking $\pi = \mathbf{Id}$ and $\sigma(X) \equiv t'$ for each $a \# X \in \Delta$ we obtain $\vdash_\tau t\sigma = u\sigma$, since $\vdash a \# \sigma(X)$. Then $\llbracket t\sigma \rrbracket_\varsigma^\tau = \llbracket u\sigma \rrbracket_\varsigma^\tau$ by Theorem 5.1. By construction $\llbracket t \rrbracket_\varsigma^\tau = \llbracket t\sigma \rrbracket_\varsigma^\tau$ and $\llbracket u \rrbracket_\varsigma^\tau = \llbracket u\sigma \rrbracket_\varsigma^\tau$, so $\llbracket t \rrbracket_\varsigma^\tau = \llbracket u \rrbracket_\varsigma^\tau$ as required. \square

A certain amount of technical subtlety is hidden in Theorem 5.4 and in particular in its use of the extra term-formers \mathbf{d} :

Theorem 5.4 (Completeness). *If $\Delta \models_\tau t = u$ then $\Delta \vdash_\tau t = u$.*

Proof. By assumption $\llbracket \Delta \vdash t = u \rrbracket_\varsigma$ is valid for any model $\llbracket _ \rrbracket$ of T and any valuation ς . Let $\llbracket _ \rrbracket$ be $\llbracket _ \rrbracket^\tau$, the term model of T . In order to choose a suitable ς , we introduce the following:

- Let S be the set of atoms mentioned anywhere in Δ , t , and u .
- Let X_1, \dots, X_n be the set of unknowns mentioned anywhere in Δ , t , or u , (*not* just in Δ !) in some arbitrary order.
- For each $1 \leq i \leq n$ let S_i be the set of all atoms $a \in S$ such that $a \# X_i \notin \Delta$.
- For each $1 \leq i \leq n$ let $\sigma(X_i) \equiv \mathbf{d}_i(a_1, \dots, a_{n_i})$ where $S_i = \{a_1, \dots, a_{n_i}\}$.

Here we extend the signature with distinct term-formers \mathbf{d}_i . Let ς map X_i to $\llbracket \sigma(X_i) \rrbracket_\varsigma^\tau$, for $1 \leq i \leq n$. By construction $\llbracket \Delta \rrbracket_\varsigma^\tau$, so by assumption $\llbracket t \rrbracket_\varsigma^\tau = \llbracket u \rrbracket_\varsigma^\tau$, and by definition this means that $t\sigma = u\sigma$ is derivable. This derivation can be transformed rule by rule into a derivation of $\Delta \vdash_\tau t = u$, since the only freshnesses and equalities we can assert of the \mathbf{d}_i are those we can also assert of the X_i . The only complication is when *perhaps* for some fresh b we use a freshness derivation to derive $b \# v\sigma$ for some fresh b ; then we modify the derivation to use (\mathbf{fr}) instead. \square

5.1 The Status of Freshness Derivations

Completeness holds for equalities — but not for freshnesses. That is, $\Delta \models_\tau b \# t$ does *not* imply $\Delta \vdash b \# t$ necessarily; for counterexamples see the discussion involving theories IOTA and LAM just after Lemma 5.2.

To understand why this is desirable we must draw a distinction between the intension and the extension of a term.

‘ $a \# (\lambda[a]b)a$ ’ has the status of ‘ x is fresh for $(\lambda x.y)x$ ’; both are false. Yet $(\lambda x.y)x$ is β -convertible to y and x is fresh for y . A freshness judgement $\Delta \vdash a \# t$ is a(n intensional) judgement on concrete syntax; call this *syntactic freshness*.⁴ All the capture-avoidance side-conditions we know of are in accordance with the slogan ‘ ϵ away from informal practice’, this is what $\Delta \vdash a \# t$ models.

Nominal sets is unusual amongst semantics in that it has a(n extensional) *semantic* notion of freshness $a \# x$. In fact, semantic freshness is hiding in nominal algebra — but in a slightly unexpected place; in the theory of *equality*.

⁴ The reader familiar with a theorem-prover such as Isabelle [16] might like to imagine that $\#$ maps to *Prop* and $=$ maps to *o*.

Theorem 5.5. *Suppose $\{X_1, \dots, X_n\}$ and $\{a_1, \dots, a_m\}$ are the unknowns and atoms mentioned in Δ and t . Suppose that b is fresh (so $b \notin \{a_1, \dots, a_m\}$).*

$\Delta \models_{\tau} a \# t$ if and only if $\Delta, b \# X_1, \dots, b \# X_n \vdash_{\tau} (b \ a) \cdot t = t$.

Proof. Choose a model and any ς such that $b \# \varsigma(X_1), \dots, b \# \varsigma(X_n)$; it follows by an induction on syntax that $b \# \llbracket t \rrbracket_{\varsigma}$. It is a fact that $a \# \llbracket t \rrbracket_{\varsigma}$ if and only if $(b \ a) \cdot \llbracket t \rrbracket_{\varsigma} = \llbracket t \rrbracket_{\varsigma}$ (see [8] for details). Also, since term-formers are interpreted by *equivariant* functions, we rewrite $(b \ a) \cdot \llbracket t \rrbracket_{\varsigma}$ as $\llbracket (b \ a) \cdot t \rrbracket_{\varsigma}$.

Now suppose $\Delta \models_{\tau} a \# t$. By definition $a \# \llbracket t \rrbracket_{\varsigma}$ for any $\llbracket - \rrbracket$ and ς such that $\llbracket \Delta \rrbracket_{\varsigma}$. By the arguments above $\llbracket (b \ a) \cdot t \rrbracket_{\varsigma} = \llbracket t \rrbracket_{\varsigma}$ if $b \# \varsigma(X_1), \dots, b \# \varsigma(X_n)$. By Theorem 5.4 it follows that $\Delta, b \# X_1, \dots, b \# X_n \vdash_{\tau} (b \ a) \cdot t = t$. The reverse implication is similar. \square

Recall the examples we used just after Lemma 5.2. Theorem 5.5 tells us that $\models_{\text{IOTA}} a \# a$ if and only if $\vdash_{\text{IOTA}} b = a$ (which follows by the axiom $\vdash a = b$), and that $\models_{\text{LAM}} a \# (\lambda[a]b)a$ if and only if $\vdash_{\text{LAM}} (\lambda[a]b)a = (\lambda[c]b)c$ (which follows since both sides are derivably equal to b).

So the semantic freshness $\Delta \models_{\tau} a \# t$ can be expressed as an equality axiom. Any undecidability or algorithmic complexity is isolated in the equality judgement form.

6 Related Work

Nominal algebra is derived from Fraenkel-Mostowski set theory. This provided a semantics for names and gave an unexpected set-theoretic semantics for parse trees (abstract syntax trees) *with* α -equivalence [8]. Existing technology included De Bruijn indexes [17], higher-order abstract syntax [18], the theory of contexts [19], TWELF [20], and other approaches. In this crowded arena approaches based on Fraenkel-Mostowski sets were catchily labelled ‘nominal’ by Pitts. The derivation rules expressed in this paper by the theory **CORE** are from this semantics. The name ‘nominal algebra’ acknowledges this debt.

Nominal unification considers the computational aspects of unifying trees-with-binding [9]. Nominal logic describes inductive programming and reasoning principles for trees-with-binding [21]. FreshML is a programming language... for trees-with-binding [22] in ML [23] style.

Aside from applications to trees-with-binding, ideas from nominal techniques have been used in logic [24] (this was still reasoning on trees, since the logic had a fixed syntactic model) and recently in semantics [25].

Nominal *rewriting* considers equalities between terms *not* directly to do with an underlying model of trees [6]. α -prolog [26] takes a similar tack but in logic programming. Still, the emphasis is squarely on the computational benefits compared to those of other approaches.

Nominal algebra champions nominal techniques as a *logical framework* to represent and reason about semantic structures — i.e. functions, sets, and other mathematical structures. That is new, and this paper sets it on a sound semantic/logical basis.

The full case for nominal algebra as a framework for applications in which higher-order logic can be used [27] remains to be made. However we can make some remarks:

Unification of nominal terms is decidable whereas higher-order unification is not [28]. (First-order) logic can be axiomatised [10] and the treatment of quantification is very smooth. In particular it closely models the informal specification of quantification; the \forall -intro rule in Isabelle is

$$\bigwedge x.[P \Longrightarrow \forall x.P]$$

(see [16]) and this is *not* like the usual rule used by logicians

‘from $\Gamma \vdash P$ deduce $\Gamma \vdash \forall x.P$ if x is fresh for Γ ’.

We believe (though we have not yet checked this in detail) that treatments of substructural logics are possible in nominal algebra; they are not necessarily so in higher-order logic because structural properties of the framework’s connectives ‘infect’ those of the logic being axiomatised.

Since the conception of nominal algebra, Pitts and Clouston have derived *nominal equational logic* [29]. Nominal equational logic makes some slightly different design decisions, notably it cannot express syntactic freshness (see Sect. 5.1).

Higher-order algebra [4] uses typed λ -calculus up to $\alpha\beta$ -equivalence and can serve as a foundational framework. The simply-typed λ -calculus itself [16, Figures 6 and 7] can also be used. Other algebraic systems, such as Sun’s extensional binding algebras [30, Definition 2.2.3] introduce dedicated systems based on functional semantics.

A host of ‘cylindric’ algebraic techniques exist. These embrace meta-variables and reject object-level variables, preferring to encode their expressive power in the term-formers. Examples are lambda-abstraction algebras [31] for the λ -calculus and cylindric algebras [32, 33] for first-order logic. Polyadic algebras have a slightly more general treatment, for a brief but clear discussion of the design of these and related systems is in [34, Appendix C]. Combinators [1] reject object-level variables altogether. Algebras over (untyped) combinators can then express first-order predicate logic [35]. These systems are effective for their applications but there are things that nominal algebra allows us to say particularly naturally, because of the way it isolates abstraction, has explicit meta-variables, and permits freshness side-conditions on equalities.

7 Conclusions

Nominal terms embrace the difference between the object-level and the meta-level; there are two classes of variables, atoms a and unknowns X . Substitution for X does not avoid capture in abstraction by a . Freshness side-conditions of the form $a\#X$ manage the interaction between the two levels and ‘recover’ capture-avoidance where this is required.

Previous ‘nominal’ work has considered reasoning on datatypes of syntax up to α -equivalence. In this paper we use nominal terms as a foundational syntax for mathematics, and we have constructed a logic for them using a suitable theory of equality. The two-level variable structure permits the framework to get exceptionally close to informal practice; see the end of the Introduction for examples.

Nominal algebra could be suitable to specify and reason algebraically on languages with binding; we have in mind particularly process calculi such as those derived from the π -calculus, which often feature quite complex binding side-conditions and for which algebraic reasoning principles are frequently developed [36, 37, 38].

It is possible to extend nominal algebra with abstractions of the form $[t]u$, and freshness judgements of the form $t\#u$. Other extensions are possible and case studies will tell which of them are most important.

We see nominal algebra as the first of a family of two-level logics as yet to be created. We intend to begin by creating a first-order logic with two levels of variable. For example we would like to be able to write an expression like

$$\forall[X]\forall[a](a\#X \supset (X \Leftrightarrow \forall[a]X)).$$

We made a start in that direction by designing one-and-a-halfth order logic [10]. However that system does not allow quantification of unknowns, and we are now working on lifting that restriction.

In our semantics object-level variables are first-class entities in the denotation; a represents an object-level variable symbol in the syntax *and* in the semantics of nominal algebra. Yet we may impose equalities, such as theory SUB from Sect. 2. If we can substitute for a then it has the flavour of a variable ‘ranging over’ denotational elements; so SUB is a theory of ‘variables in denotation’. Nominal algebra is the beginning of an intriguing investigation into what this enrichment of the denotation does to the theory of validity and proof.

We are also interested in developing logics with *hierarchies* of variables. Since nominal algebra offers two levels of variable, why not extend this to allow an infinite hierarchy of variables, by analogy with type hierarchies in the λ -calculus [1], or stratification in set theory [39]? The first author has considered this extension of nominal terms in other publications [40, 41] but introducing such a hierarchy in a *logical* context poses unique challenges, and in particular, we have not yet obtained a satisfactory notion of model. This too is current research.

References

- [1] Barendregt, H.P.: The Lambda Calculus: its Syntax and Semantics. North-Holland (revised edn.)
- [2] Curry, H.B., Feys, R.: Combinatory Logic. vol. 1, North Holland (1958)
- [3] Henkin, L., Monk, J.D., Tarski, A.: Cylindric Algebras. North Holland (1971, 1985) Parts I and II
- [4] Meinke, K.: Universal algebra in higher types. Theoretical Computer Science 100(2), 385–417 (1992)

- [5] Leivant, D.: Higher order logic. In: Gabbay, D., Hogger, C., Robinson, J. (eds.) *Handbook of Logic in Artificial Intelligence and Logic Programming*. vol. 2, pp. 229–322. Oxford University Press, Oxford, UK (1994)
- [6] Fernández, M., Gabbay, M.J.: Nominal rewriting. *Information and Computation* (in press)
- [7] Fernández, M., Gabbay, M.J.: Curry-style types for nominal rewriting. In: *TYPES'06* (2006)
- [8] Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. *Formal Aspects of Computing* 13(3–5), 341–363 (2001)
- [9] Urban, C., Pitts, A.M., Gabbay, M.J.: Nominal unification. *Theoretical Computer Science* 323(1–3), 473–497 (2004)
- [10] Gabbay, M.J., Mathijssen, A.: One-and-a-halfth-order logic. In: *PPDP '06. Proc. of the 8th ACM SIGPLAN symposium on Principles and Practice of Declarative Programming*, pp. 189–200. ACM Press, New York (2006)
- [11] Girard, J.Y., Taylor, P., Lafont, Y.: *Proofs and types*. Cambridge University Press, Cambridge (1989)
- [12] Parrow, J.: An introduction to the pi-calculus. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) *Handbook of Process Algebra*, pp. 479–543. Elsevier, Amsterdam (2001)
- [13] Hodges, W.: Elementary predicate logic. In: Gabbay, D., Guenther, F. (eds.) *Handbook of Philosophical Logic*, 2nd edn. vol. 1, pp. 1–131. Kluwer Academic Publishers, Dordrecht (2001)
- [14] Gabbay, M.J., Mathijssen, A.: Capture-avoiding substitution as a nominal algebra. In: Barkaoui, K., Cavalcanti, A., Cerone, A. (eds.) *ICTAC 2006. LNCS*, vol. 4281, pp. 198–212. Springer, Heidelberg (2006)
- [15] Gabbay, M.J.: Fresh logic. *Journal of Logic and Computation* (2006) In press.
- [16] Paulson, L.C.: The foundation of a generic theorem prover. *Journal of Automated Reasoning* 5(3), 363–397 (1989)
- [17] de Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae* 5(34), 381–392 (1972)
- [18] Pfenning, F., Elliot, C.: Higher-order abstract syntax. In: *PLDI '88. Proc. of the ACM SIGPLAN 1988 conf. on Programming Language design and Implementation*, pp. 199–208. ACM Press, New York (1988)
- [19] Miculan, M.: Developing (meta)theory of lambda-calculus in the theory of contexts. *ENTCS* 1(58) (2001)
- [20] Pfenning, F., Schürmann, C.: System description: Twelf - a meta-logical framework for deductive systems. In: Ganzinger, H. (ed.) *CADE-16. LNCS (LNAI)*, vol. 1632, pp. 202–206. Springer, Heidelberg (1999)
- [21] Pitts, A.M.: Nominal logic, a first order theory of names and binding. *Information and Computation* 186(2), 165–193 (2003)
- [22] Shinwell, M.R., Pitts, A.M., Gabbay, M.J.: FreshML: Programming with binders made simple. In: *ICFP 2003. Eighth ACM SIGPLAN Int'l Conf. on Functional Programming*, Uppsala, Sweden, pp. 263–274. ACM Press, New York (2003)
- [23] Paulson, L.C.: *ML for the working programmer* (2nd eds.). Cambridge University Press, Cambridge (1996)
- [24] Luís Caires, L.C.: A spatial logic for concurrency (part II). *Theoretical Computer Science* 322(3), 517–565 (2004)
- [25] Benton, N., Leperchey, B.: Relational reasoning in a nominal semantics for storage. In: *Urzyczyn, P. (ed.) TLCA 2005. LNCS*, vol. 3461, pp. 86–101. Springer, Heidelberg (2005)

- [26] Cheney, J., Urban, C.: System description: Alpha-Prolog, a fresh approach to logic programming modulo alpha-equivalence. In: de Valencia, U.P. (ed.) Proc. 17th Int. Workshop on Unification, UNIF'03, 15–19 (2003)
- [27] Paulson, L.C.: Isabelle: the next 700 theorem provers. In: Odifreddi, P. (ed.) Logic and Computer Science, pp. 361–386. Academic Press, London (1990)
- [28] Huet, G.: Higher order unification 30 years later. In: Carreño, V.A., Muñoz, C.A., Tahar, S. (eds.) TPHOLs 2002. LNCS, vol. 2410, pp. 3–12. Springer, Heidelberg (2002)
- [29] Clouston, R.A., Pitts, A.M.: Nominal equational logic. ENTCS 172, 223–257 (2007)
- [30] Sun, Y.: An algebraic generalization of frege structures - binding algebras. Theoretical Computer Science 211, 189–232 (1999)
- [31] Salibra, A.: On the algebraic models of lambda calculus. Theoretical Computer Science 249(1), 197–240 (2000)
- [32] Burris, S., Sankappanavar, H.: A Course in Universal Algebra. Springer, Heidelberg (1981)
- [33] Andréka, H., Németi, I., Sain, I.: Algebraic logic. In: Gabbay, D., Guenther, F. (eds.) Handbook of Philosophical Logic, 2nd edn. pp. 133–249. Kluwer Academic Publishers, Dordrecht (2001)
- [34] Blok, W.J., Pigozzi, D.: Algebraizable logics. Memoirs of the AMS 77(396) (1989)
- [35] Barendregt, H., Dekkers, W., Bunder, M.: Completeness of two systems of illative combinatory logic for first-order propositional and predicate calculus. Archive für Mathematische Logik 37, 327–341 (1998)
- [36] Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: the spi calculus. In: CCS '97. Proc. of the 4th ACM conf, pp. 36–47. ACM Press, New York (1997)
- [37] Luttik, B.: Choice Quantification in Process Algebra. PhD thesis, University of Amsterdam (2002)
- [38] Katoen, J.P., D'Argenio, P.R.: General distributions in process algebra. In: Lectures on formal methods and performance analysis: first EEF/Euro summer school on trends in computer science, Springer, pp. 375–429. Springer, Heidelberg (2002)
- [39] Forster, T.: Quine's NF, 60 years on. American Mathematical Monthly 104(9), 838–845 (1997)
- [40] Gabbay, M.J.: A new calculus of contexts. In: PPDP '05. Proc. of the 7th ACM SIGPLAN int'l conf. on Principles and Practice of Declarative Programming, pp. 94–105. ACM Press, New York (2005)
- [41] Gabbay, M.J.: Hierarchical nominal rewriting. In: LFMTP'06: Logical Frameworks and Meta-Languages: Theory and Practice, pp. 32–47 (2006)

Formal Verification of an Optimal Air Traffic Conflict Resolution and Recovery Algorithm^{*}

André L. Galdino^{1,**}, César Muñoz², and Mauricio Ayala-Rincón^{1,**}

¹ Universidade de Brasília, Brazil
{galdino, ayala}@unb.br

² National Institute of Aerospace, USA
munoz@nianet.org

Abstract. Highly accurate positioning systems and new broadcasting technology have enabled air traffic management concepts where the responsibility for aircraft separation resides on pilots rather than on air traffic controllers. The Formal Methods Group at the National Institute of Aerospace and NASA Langley Research Center has proposed and formally verified an algorithm, called KB3D, for distributed three dimensional conflict resolution. KB3D computes resolution maneuvers where only one component of the velocity vector, i.e., ground speed, vertical speed, or heading, is modified. Although these maneuvers are simple to implement by a pilot, they are not necessarily optimal from a geometrical point of view. In general, optimal resolutions require the combination of all the components of the velocity vector. In this paper, we propose a two dimensional version of KB3D, which we call KB2D, that computes resolution maneuvers that are optimal with respect to ground speed and heading changes. The algorithm has been mechanically verified in the Prototype Verification System (PVS). The verification relies on algebraic proof techniques for the manipulation of the geometrical concepts relevant to the algorithm as well as standard deductive techniques available in PVS.

1 Introduction

Air traffic management concepts such as *Free Flight* [12,5] and *Distributed Air/Ground Traffic Management* [10] target the predicted increase of air traffic by distributing the responsibility for separation among pilots and air traffic controllers. This new mode of operation is supported by advances in surveillance, communications, and software technologies. In particular, conflict detection and resolution (CD&R) systems are designed to increase traffic awareness and provide corrective maneuvers for impending conflicts [7]. On-board CD&R systems

^{*} This work was supported by the National Aeronautics and Space Administration, Langley Research Center under the Research Cooperative Agreement No. NCC-1-02043 and by the Brazilian Research Council CNPq under Grant 471791/2004-0.

^{**} Authors partially supported by the Brazilian Research Council CNPq.

are particularly attractive as they support the required decentralized decision making of new air traffic management concepts.

KB3D [2] is a three dimensional (3D) distributed CD&R algorithm, designed and formally verified by the Formal Methods Group at NIA and NASA Langley Research Center. Since KB3D uses state information, e.g., position and velocity vectors, to detect and solve conflicts between two aircraft, namely, *ownship* and *traffic*, it can be characterized as a *pairwise tactical* algorithm. KB3D assumes that the ownship and traffic aircraft are surrounded by a cylindrical protected zone of diameter D and height H centered at the aircraft's positions. A *loss of separation* between the two aircraft is defined as the overlapping of their protected zones. A *conflict* is a predicted loss of separation within a lookahead time T . When a conflict is detected, KB3D outputs a choice of resolution maneuvers for the ownship. Each resolution maneuver is expressed as a new velocity vector for the ownship that yields a conflict-free trajectory. The resolutions computed by KB3D modify only one component of the ownship's velocity vector:

- *Vertical speed*: the aircraft keeps the horizontal component of its velocity vector, but modifies its vertical speed;
- *Ground speed*: the aircraft keeps its heading and vertical speed but modifies its ground speed;
- *Heading*: the aircraft keeps its ground speed and vertical speed but modifies its heading.

Not all conflict situations have all three kinds of resolutions. However, if the aircraft are in conflict but they are still separated, KB3D guarantees at least one theoretical vertical solution.

Two important extensions to KB3D are (1) *time arrival constraints* and (2) *coordinated strategies*.

1. **Time Arrival Constraints.** KB3D has been extended to compute *recovery* maneuvers [4, 8]. A recovery maneuver brings back the ownship to its target point at the scheduled time, once the conflict has been avoided.
2. **Coordinated Strategies.** A *strategy* is a procedure that selects a subset of the resolution maneuvers proposed by a CD&R algorithm. A strategy is *coordinated* if it ensures separation when both aircraft simultaneously maneuver to solve the conflict. Coordinated strategies guarantee that aircraft using the same CD&R system will not fly into each other during the resolution maneuver. The coordination is *implicit* if the only communication required to achieve the coordination is the broad-casted state information of the aircraft. It has been formally verified that KB3D supports an implicitly coordinated resolution strategy [3].

KB3D resolutions yield trajectories for the ownship where the protected zones of the aircraft touch but not overlap. These trajectories require a small change of course for the ownship. However, since KB3D resolutions modify only one component of the original velocity vector, i.e., heading, vertical speed, or ground speed; KB3D resolutions are not necessarily optimal from a geometrical point

of view. In general, an optimal resolution requires simultaneous variations in all three components. In this paper, we propose a two dimensional (2D) version of KB3D, called KB2D, that computes optimal resolution maneuvers for combined variations of ground speed and heading. As KB3D, KB2D has been extended with time arrival constraints and coordinated resolutions. Moreover, KB2D has been specified and mechanically verified in the Prototype Verification System (PVS) [11]. In addition to the standard deductive techniques available in PVS, we have extensively used strategies in the PVS packages Field [9] and Manip [13]. These packages provide tools for the algebraic manipulations required to deal with the geometrical concepts used by the algorithm.

This paper is organized as follows. Section 2 introduces the geometric framework for specifying the 2D CD&R problem. Section 3 presents the KB2D algorithm. The formal proofs that the resolution and recovery maneuvers computed by KB2D are correct and optimal, and that KB2D supports coordinated maneuvers are presented in Section 4.

2 Geometric Framework

We consider the airspace as a 2D Cartesian coordinate system in \mathbb{R}_2 . The ownship's and traffic's initial positions, at time $t = 0$, are given by the vectors $\mathbf{s}_o = (s_{ox}, s_{oy})$ and $\mathbf{s}_i = (s_{ix}, s_{iy})$, respectively. The ownship's and traffic's original velocity vectors are given by $\mathbf{v}_o = (v_{ox}, v_{oy})$ and $\mathbf{v}_i = (v_{ix}, v_{iy})$, respectively.

The representation of the airspace by a 2D Cartesian system hints that the KB2D logic is based on a flat earth assumption. Indeed, we represent aircraft dynamics by a simple point moving at constant speed along a linear trajectory. Hence, the course of an aircraft can be described by a position, a velocity vector, and a time interval. We also assume that aircraft can change course and speed in zero time. All these unrealistic assumptions are typical of tactical CD&R systems with short lookahead times (usually, $T = 5$ minutes) and large *protected zones* (usually, $D = 5$ nautical miles and $H = 1000$ feet), to be defined below.

As it simplifies the mathematical development, we consider the ownship's motion relative to the traffic aircraft. Hence, we introduce a *relative coordinate system* where the traffic's position is at the origin, and the relative position and velocity vectors of the ownship with respect to the traffic aircraft are given by $\mathbf{s} = (s_x, s_y) = \mathbf{s}_o - \mathbf{s}_i$ and $\mathbf{v} = (v_x, v_y) = \mathbf{v}_o - \mathbf{v}_i$, respectively. In the relative coordinate system, the protected zone P is a cylinder of diameter $2D$ located in the center of the coordinate system:

$$P = \{(x, y, 0) \mid x^2 + y^2 < D^2\}. \quad (1)$$

Consequently, we define *loss of separation* at time t as the incursion of the ownship in the relative protected zone of the traffic aircraft at time t , i.e.,

$$\mathbf{s} + t\mathbf{v} \in P. \quad (2)$$

Given a lookahead time T , the aircraft are said to be in *conflict* if there exists a time $0 < t < T$ when they are predicted to lose separation.

We note that the relative protected zone P is twice the size of each individual protected zone in the absolute coordinate system. It can be easily checked that the definitions of “conflict” and “loss of separation” in the relative coordinate system are equivalent to the definitions in the absolute one.

A *resolution* is a new velocity vector \mathbf{v}'_o for the ownship. The resolution is *correct* if for all $t > 0$,

$$\mathbf{s} + t(\mathbf{v}'_o - \mathbf{v}_i) \notin P. \quad (3)$$

A resolution \mathbf{v}'_o for the ownship is *smaller* than a resolution \mathbf{v}'_a , denoted by the order relation $\mathbf{v}'_o \preceq \mathbf{v}'_a$, if and only if

$$\|\mathbf{v}'_o - \mathbf{v}_o\| \leq \|\mathbf{v}'_a - \mathbf{v}_o\|, \quad (4)$$

where $\|\mathbf{v}\|$ denotes the norm of \mathbf{v} .

Resolutions \mathbf{v}'_o and \mathbf{v}'_i are *coordinated* for the ownship and traffic aircraft, respectively, if for all $t > 0$,

$$\mathbf{s} + t(\mathbf{v}'_o - \mathbf{v}'_i) \notin P. \quad (5)$$

An *arrival time* $t'' > 0$ determines a way point \mathbf{s}'' , called *target point*:

$$\mathbf{s}'' = \mathbf{s} + t''\mathbf{v}. \quad (6)$$

A *resolution/recovery maneuver* for an arrival time t'' is a triple $(t', \mathbf{v}'_o, \mathbf{v}''_o)$ where $t' > 0$ is a *time of switch*, \mathbf{v}'_o is a resolution velocity for the ownship, and \mathbf{v}''_o is a recovery velocity for the ownship. The resolution/recovery maneuver is *correct* if and only if

- \mathbf{v}'_o is a correct resolution for the ownship, and
- \mathbf{v}''_o is a correct recovery for the ownship, i.e., for all times $0 \leq t \leq t'' - t'$,

$$\mathbf{s} + t'\mathbf{v}' + t\mathbf{v}'' \notin P, \text{ and} \quad (7)$$

$$\mathbf{s} + t'\mathbf{v}' + (t'' - t')\mathbf{v}'' = \mathbf{s}'', \quad (8)$$

where $\mathbf{v}' = \mathbf{v}'_o - \mathbf{v}_i$ and $\mathbf{v}'' = \mathbf{v}''_o - \mathbf{v}_i$.

3 KB2D

KB2D, like KB3D, computes resolution maneuvers that yield trajectories that are tangent to the protected zone in the relative coordinate system. Figure 1 shows a conflict situation and the resolution and recovery courses in a two dimensional geometry. This figure illustrates two symmetries in the conflict resolution and recovery problem. First, resolution and recovery maneuvers are solved in a symmetric way, i.e., a recovery course for the conflicting situation described by the relative position and velocity vectors \mathbf{s} , \mathbf{v} , and arrival time t'' is a resolution

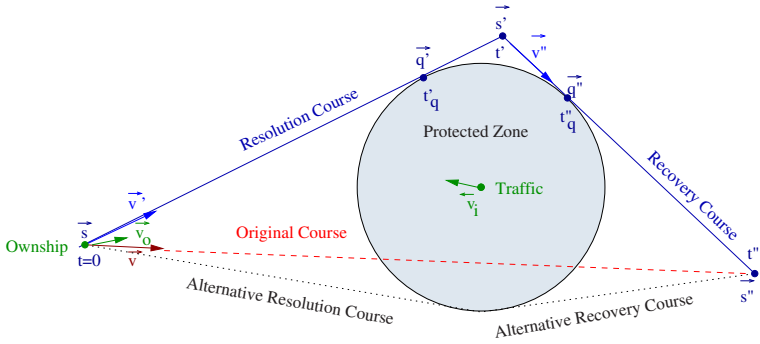


Fig. 1. Resolution and recovery courses (2D)

course for the conflicting situation described by the target point $s'' = s + t''v$, relative velocity vector $-v$, and arrival time t'' .

The second symmetry is due to the fact that for any relative position s exterior to P , i.e., $s_x^2 + s_y^2 > D^2$, there are two tangent courses to the protected zone. Each tangent to the protected zone determines a family of resolution and recovery maneuvers. Intuitively, the optimal relative resolution maneuver, with respect to the order relation \preceq defined by Formula 4, is member of one of these families. Furthermore, for each family, the optimal relative resolution maneuver v' is the perpendicular projection of the relative vector v on the corresponding tangent course.

Based on all these observations, we define two functions **kb2d** and **recovery**. The function **kb2d** has as inputs

- the initial relative ownship's position $s = (s_x, s_y)$,
- the absolute velocities $v_o = (v_{ox}, v_{oy})$, $v_i = (v_{ix}, v_{iy})$ of the ownship and traffic aircraft, respectively, and
- a parameter $e = \pm 1$, which determines a particular tangent for the resolution course.

It returns a couple (v'_{ox}, v'_{oy}) that defines a resolution velocity vector for the ownship $v'_o = (v'_{ox}, v'_{oy})$. The function **recovery** has the same inputs as **kb2d** and, additionally, an arrival time t'' . It returns a triple (t', v''_{ox}, v''_{oy}) that defines a time of switch t' (from resolution to recovery) and a recovery velocity vector for the ownship $v''_o = (v''_{ox}, v''_{oy})$. The functions are defined in Listing 1.1.

We have formally verified that the resolution/recovery maneuvers computed by **kb2d** and **recovery** are *correct*, i.e., they satisfy formulas 3, 7, and 8. In other words, if the ownship flies the resolution course from time 0 to t' and the recovery course from time t' to t'' , then

- (a) it shall not be in conflict at any time and
- (b) it shall arrive to the target point s'' at the arrival time t'' .

Furthermore, we also show that the resolution v'_o computed by **kb2d** is optimal with respect to the order \preceq defined by Formula 4, and that it is coordinated with

Listing 1.1. The functions `kb2d` and `recovery`

```

kb2d( $s_x, s_y, v_{ox}, v_{oy}, v_{ix}, v_{iy}, e$ ) : [real,real] =
  let  $(v_x, v_y) = (v_{ox} - v_{ix}, v_{oy} - v_{iy})$  in
  let  $(q'_x, q'_y) = (Q(s_x, s_y, e), Q(s_y, s_x, -e))$  in
  let  $t'_q = \text{contact\_time}(s_x, s_y, q'_x, q'_y, v_x, v_y, e)$  in
  if  $t'_q > 0$  then  $((q'_x - s_x)/t'_q + v_{ix}, (q'_y - s_y)/t'_q + v_{iy})$ 
  elsif  $t'_q = 0$  then  $(v_{ix}, v_{iy})$ 
  else  $(0,0)$ 
  endif

recovery( $s_x, s_y, v_{ox}, v_{oy}, v_{ix}, v_{iy}, t'', e$ ) : [real,real,real] =
  let  $(v_x, v_y) = (v_{ox} - v_{ix}, v_{oy} - v_{iy})$  in
  let  $(s'', s'') = (s_x + t''v_x, s_y + t''v_y)$  in
  let  $(v'_{ox}, v'_{oy}) = \text{kb2d}(s_x, s_y, v_{ox}, v_{oy}, v_{ix}, v_{iy}, e)$  in
  let  $(v'_x, v'_y) = (v'_{ox} - v_{ix}, v'_{oy} - v_{iy})$  in
  let  $t' = \text{switching\_time}(s_x, s_y, s'', s'', v'_x, v'_y, e)$  in
  if  $t' > 0$  AND  $t'' - t' > 0$  then
     $(t', (t''v_x - t'v'_x)/(t'' - t') + v_{ix}, (t''v_y - t'v'_y)/(t'' - t') + v_{iy})$ 
  else  $(0,0,0)$ 
  endif

alpha( $s_x, s_y$ ) : real =  $D^2/(s_x^2 + s_y^2)$ 

beta( $s_x, s_y$ ) : real =  $D\sqrt{s_x^2 + s_y^2 - D^2}/(s_x^2 + s_y^2)$ 

Q( $s_x, s_y, e$ ):real = alpha( $s_x, s_y$ ) $s_x + e$  beta( $s_x, s_y$ ) $s_y$ 

contact_time( $s_x, s_y, q_x, q_y, v_x, v_y, e$ ) : real =
  let  $d = v_x(q_x - s_x) + v_y(q_y - s_y)$  in
  if  $d \neq 0$  then  $((q_x - s_x)^2 + (q_y - s_y)^2)/d$ 
  else 0
  endif

switching_time( $s_x, s_y, s''_x, s''_y, v'_x, v'_y, e$ ) : real =
  if  $s''_x^2 + s''_y^2 > D^2$  then
    let  $(q'', q'') = (Q(s''_x, s''_y, -e), Q(s''_y, s''_x, e))$  in
    let  $(u_x, u_y) = (q''_x - s''_x, q''_y - s''_y)$  in
    let  $d = v'_y u_x - v'_x u_y$  in
    if  $d \neq 0$  then  $((s_x - s''_x)u_y + (s''_y - s_y)u_x)/d$ 
    else 0
  endif
  else 0
  endif

```


is taken when the contact time is negative or when \mathbf{v}' does not point toward \mathbf{q} . In those cases, a null value is returned to indicate that there is no solution.

The function **recovery** first finds the time of switch t' which defines a point that lies on L and on the tangent line that passes through the target point \mathbf{s}'' . Then, it uses the fact that resolution and recovery maneuvers are solved in a symmetric way to compute the relative recovery vector \mathbf{v}'' . Finally, **recovery** returns a triple formed by the time of switch t' and the components of the absolute ownship recovery vector $\mathbf{v}_o'' = \mathbf{v}'' + \mathbf{v}_i$.

The predicate $\text{conflict?}(\mathbf{s}, \mathbf{v}, T)$ holds if the aircraft are in conflict, i.e., if there is a time $0 < t < T$ that satisfies Formula 2:

$$\text{conflict?}(\mathbf{s}, \mathbf{v}, T) \equiv \exists 0 < t < T : (s_x + tv_x)^2 + (s_y + tv_y)^2 < D^2.$$

or equivalently, $\exists 0 < t < T : \mathbf{s} + t\mathbf{v} \in P$ according to Equation 2.

The predicate $\text{separation?}(\mathbf{s}, \mathbf{v})$ holds if the relative velocity \mathbf{v} guarantees separation for all time $t > 0$:

$$\text{separation?}(\mathbf{s}, \mathbf{v}) \equiv \forall t > 0 : (s_x + tv_x)^2 + (s_y + tv_y)^2 \geq D^2.$$

or equivalently, $\forall t > 0 : \mathbf{s} + t\mathbf{v} \notin P$ according to Equation 2.

A moving point $\mathbf{s} + t\mathbf{v}$ intersects the surface of the protected zone P if and only if

$$(s_x + tv_x)^2 + (s_y + tv_y)^2 = D^2. \quad (9)$$

Since we consider that the ground speed is not null, i.e., $v_x^2 + v_y^2 \neq 0$, Formula 9 reduces to a quadratic equation in t :

$$t^2(v_x^2 + v_y^2) + 2t(s_x v_x + s_y v_y) + s_x^2 + s_y^2 - D^2 = 0. \quad (10)$$

The discriminant of Formula 10, $\Delta(\mathbf{s}, \mathbf{v})$, is defined as

$$\Delta(\mathbf{s}, \mathbf{v}) = D^2(v_x^2 + v_y^2) - (s_x v_y - s_y v_x)^2. \quad (11)$$

If $\Delta(\mathbf{s}, \mathbf{v}) < 0$ then the moving point does not intersect P . Furthermore, if $\Delta(\mathbf{s}, \mathbf{v}) = 0$ we have the tangent case.

Lemma 1 (tangent_correctness). *For all $\mathbf{s}, \mathbf{v} \neq \mathbf{0}$,*

$$\begin{aligned} & s_x^2 + s_y^2 \geq D^2 \quad \wedge \\ & \Delta(\mathbf{s}, \mathbf{v}) = 0 \\ & \Rightarrow \\ & \text{separation?}(\mathbf{s}, \mathbf{v}). \end{aligned}$$

Proof. See Maddalon et al [8]. □

A resolution maneuver is said to be a *line solution* for parameter e if it lies on the tangent corresponding to e . Formally [3],

$$\text{line_solution?}(\mathbf{s}, \mathbf{v}, e) \equiv s_x v_y - s_y v_x = e \frac{D(s_x v_x + s_y v_y)}{\sqrt{s_x^2 + s_y^2 - D^2}}. \quad (12)$$

The corresponding resolution maneuvers of two conflicting aircraft are coordinated if they are line solutions for the same parameter e .

Lemma 2 (*coordinated_line*). *For all $s, v = v_o - v_i, T > 0, D > 0, v' = v'_o - v'_i, e = \pm 1,$*

$$\begin{aligned}
 & \text{conflict?}(s, v, T) \wedge \\
 & s_x^2 + s_y^2 > D^2 \wedge \\
 & \text{line_solution?}(s, v'_o - v_i, e) \wedge \\
 & \text{line_solution?}(-s, v'_i - v_o, e) \\
 & \Rightarrow \\
 & \text{separation?}(s, v').
 \end{aligned}$$

Proof. See Dowek and Munoz [3]. □

4.2 Correctness of Resolution

The following theorem states that if the resolution computed by **kb2d** is not null, then it is correct, i.e., the resolution verifies Formula 3.

Theorem 1 (*kb2d_correct*). *For all $s, v = v_o - v_i, T > 0, D > 0, v', v'_o, e = \pm 1,$*

$$\begin{aligned}
 & \text{conflict?}(s, v, T) \wedge \\
 & s_x^2 + s_y^2 > D^2 \wedge \\
 & v'_o = \text{kb2d}(s_x, s_y, v_{ox}, v_{oy}, v_{ix}, v_{iy}, e) \wedge \\
 & v' = v'_o - v_i \wedge v'_o \neq 0 \\
 & \Rightarrow \\
 & \text{separation?}(s, v').
 \end{aligned}$$

Proof (Sketch). We prove that the line L is tangent to the protected zone and that the relative velocity vector v' lies on L ; i.e., $\Delta(s, v') = 0$. We conclude by using Lemma 1 (*tangent_correctness*). □

4.3 Optimality

We prove that the resolution velocity v'_o computed by **kb2d** is optimal with respect to any velocity vector v'_a , where $v'_a - v_i$ lies on the same tangent L , i.e, $v'_o \preceq v'_a$ as defined by Formula 4.

Theorem 2 (*kb2d_optimal*). *For all $s, v = v_o - v_i, T > 0, D > 0, v', v'_o, v'_a, k, t'_q, e = \pm 1,$*

$$\begin{aligned}
 & \text{conflict?}(s, v, T) \wedge \\
 & s_x^2 + s_y^2 > D^2 \wedge \\
 & v'_o = \text{kb2d}(s_x, s_y, v_{ox}, v_{oy}, v_{ix}, v_{iy}, e) \wedge \\
 & t'_q = \text{contact_time}(s_x, s_y, q_x, q_y, v_x, v_y, e) \wedge t'_q > 0 \wedge \\
 & v'_a - v_i = kv' \\
 & \Rightarrow \\
 & v'_o \preceq v'_a.
 \end{aligned}$$

Proof (Sketch). The result follows by establishing that the factor $\frac{1}{t'_q}$ minimizes the distance between the point A (ending point of \mathbf{v}) and the ending point of a velocity vector that lies on L . As this factor is used for the definition of \mathbf{v}' , we have that for all k , $\|\mathbf{v}' - \mathbf{v}\| \leq \|k\mathbf{v}' - \mathbf{v}\|$. Therefore, $\|\mathbf{v}'_o - \mathbf{v}_o\| \leq \|\mathbf{v}'_a - \mathbf{v}_o\|$. \square

4.4 Correctness of Recovery

The following theorem states that if the time of switch satisfies the condition $0 < t' < t''$ and the resolution velocity computed by **recovery** is not null, then the recovery is correct, i.e., it satisfies formulas 7 and 8.

Theorem 3 (recovery_correct). *For all $s, s', \mathbf{v} = \mathbf{v}_o - \mathbf{v}_i, T > 0, D > 0, \mathbf{v}', \mathbf{v}'_o, \mathbf{v}'', \mathbf{v}''_o, t', t'', e = \pm 1$,*

$$\begin{aligned}
& \text{conflict?}(s, \mathbf{v}, T) \wedge \\
& s_x^2 + s_y^2 > D^2 \wedge \\
& \mathbf{s}'' = \mathbf{s} + t''\mathbf{v} \wedge \\
& s_x''^2 + s_y''^2 > D^2 \wedge \\
& \mathbf{v}'_o = kb2d(s_x, s_y, v_{ox}, v_{oy}, v_{ix}, v_{iy}, e) \wedge \\
& (t', \mathbf{v}'_o) = \text{recovery}(s_x, s_y, v_{ox}, v_{oy}, v_{ix}, v_{iy}, t'', e) \wedge \\
& \mathbf{v}' = \mathbf{v}'_o - \mathbf{v}_i \wedge \mathbf{v}'' = \mathbf{v}''_o - \mathbf{v}_i \wedge \\
& \mathbf{s}' = \mathbf{s} + t'\mathbf{v}' \wedge 0 < t' < t'' \wedge \\
& \mathbf{v}'_o \neq \mathbf{0} \\
& \Rightarrow \\
& \text{separation?}(\mathbf{s}', \mathbf{v}'') \wedge \\
& \mathbf{s}' + (t'' - t')\mathbf{v}'' = \mathbf{s}''.
\end{aligned}$$

Proof (Sketch). We proof that the time of switch t' defines a point $\mathbf{s}' = \mathbf{s} + t'\mathbf{v}'$ that lies on both the tangent line L and the tangent line that passes through the target point \mathbf{s}'' . We conclude by using Theorem 1 and the fact that a recovery maneuver is a resolution maneuver for the conflicting situation described by the target point \mathbf{s}'' , relative velocity vector $-\mathbf{v}$, and arrival time t'' . \square

4.5 Coordinated Maneuvers

Let \mathbf{v}'_o and \mathbf{v}'_i be the resolutions computed by **kb2d** for the ownship and traffic aircraft, respectively. If neither the resolution velocity vector \mathbf{v}'_o nor \mathbf{v}'_i are null, then they are coordinated, i.e., they verify Formula 5.

Theorem 4 (kb2d_coordinated). *For all $s, \mathbf{v} = \mathbf{v}_o - \mathbf{v}_i, T > 0, D > 0, \mathbf{v}' = \mathbf{v}'_o - \mathbf{v}'_i, e = \pm 1$,*

$$\begin{aligned}
& \text{conflict?}(s, \mathbf{v}, T) \wedge \\
& s_x^2 + s_y^2 > D^2 \wedge \\
& \mathbf{v}'_o = kb2d(s_x, s_y, v_{ox}, v_{oy}, v_{ix}, v_{iy}, e) \wedge \\
& \mathbf{v}'_i = kb2d(-s_x, -s_y, v_{ix}, v_{iy}, v_{ox}, v_{oy}, e) \wedge \\
& \mathbf{v}'_o \neq \mathbf{0} \wedge \mathbf{v}'_i \neq \mathbf{0} \\
& \Rightarrow \\
& \neg \text{conflict?}(s, \mathbf{v}', T).
\end{aligned}$$

Proof (Sketch). Notice that the time of loss of separation is the same for both aircraft and that the relative positions computed by each aircraft are opposite, that is; the relative position of ownship is \mathbf{s} and that of traffic is $-\mathbf{s}$. Hence, it is not difficult to see that KB2D always selects the same e to compute a tangent trajectory for both aircraft. Then, we show that \mathbf{v}'_o and \mathbf{v}'_i are line solutions for the same e , i.e., they satisfy Formula 12. We conclude by using Lemma 2 (`coordinated_line`). \square

5 Conclusions and Future Work

We proposed a pairwise resolution and recovery algorithm for two dimensional air traffic conflicts. The algorithm, which we call KB2D, computes maneuvers that are correct, i.e., they guarantee traffic separation, and coordinated, i.e, the separation is guaranteed even if both aircraft simultaneously maneuver to solve the conflict. This coordination is achieved without explicit exchange of intent information.

KB2D is strongly inspired on KB3D [2], a three dimensional algorithm designed and formally verified by the Formal Methods Group at NIA and NASA Langley Research Center. In contrast to KB3D, resolution maneuvers computed by KB2D are geometrically optimal for simultaneous changes of heading and ground speed. KB2D, like KB3D, uses the geometrical analysis proposed by K. Bilimoria in [1]. However, KB2D and Bilimoria's geometric optimization algorithm are completely different solutions to the same problem. In particular, we intentionally avoid the use of trigonometric functions in KB2D. Therefore, we conjecture that our algorithm is less susceptible to numerical instability due to floating point errors.

Using computer algebra tools, F. Kirchner has solved the optimality problem for a 3D geometry [6]. A purely geometrical optimization for a 3D airspace may not be practical since horizontal and vertical velocity changes compare differently in terms of fuel efficiency, passenger comfort, and aircraft performance. Future work includes the development of a verification framework for optimal CD&R for a parametric cost function.

Finally, we stress the fact that KB2D has been mechanically verified in PVS [11]. The PVS development consists of 37 lemmas and 800 lines of specification. Given the critical nature of conflict detection and resolution systems, we believe that formal verification of CD&R algorithms is a necessary step toward the safety analysis of new air traffic management concepts.

References

1. Bilimoria, K.: A geometric optimization approach to aircraft conflict resolution. In: Guidance, Navigation, and Control Conference, vol. AIAA 2000-4265, Denver, CO (August 2000)
2. Dowek, G., Geser, A., Muñoz, C.: Tactical conflict detection and resolution in a 3-D airspace. In: Proceedings of the 4th USA/Europe Air Traffic Management R&DSeminar, ATM 2001, Santa Fe, New Mexico, 2001. A long version appears as report NASA/CR-2001-210853 ICASE Report No. 2001-7 (2001)

3. Dowek, G., Muñoz, C., Carreño, V.: Provably safe coordinated strategy for distributed conflict resolution. In: Dowek, G. (ed.) *Proceedings of the AIAA Guidance Navigation, and Control Conference and Exhibit 2005*, San Francisco, California, AIAA-2005-6047 (2005)
4. Geser, A., Muñoz, C., Dowek, G., Kirchner, F.: *Air Traffic Conflict Resolution and Recovery*. ICASE Report 2002-12, ICASE, Langley Research Center (2002)
5. Hoekstra, J., Ruigrok, R., van Gent, R., Visser, J., Gijsbers, B., Valenti, M., Heesbeen, W., Hilburn, B., Groeneweg, J., Bussink, F.: *Overview of NLR free flight project 1997-1999*. Technical Report NLR-CR-2000-227, National Aerospace Laboratory (NLR) (May 2000)
6. Kirchner, F.: *Optimal unconstrained solution to conflict resolution in 3-d airspace*. Manuscript (2001)
7. Kuchar, J., Yang, L.: A review of conflict detection and resolution modeling methods. *IEEE Transactions on Intelligent Transportation Systems* 1(4), 179–189 (2000)
8. Maddalon, J., Butler, R., Geser, A., Muñoz, C.: *Formal verification of a conflict resolution and recovery algorithm*. Technical Report NASA/TP-2004-213015, NASA Langley Research Center, NASA LaRC, Hampton VA 23681-2199, USA (April 2004)
9. Muñoz, C., Mayero, M.: *Real automation in the field*. ICASE Interim Report 39 NASA/CR-2001-211271, NASA Langley Research Center, NASA Langley Research Center (December 2001)
10. NASA: *Concept definition for Distributed Air/Ground Traffic Management (DAG-TM), version 1.0*. Advanced Air Transportation Technologies (AATT) Project. NASA Ames and Langley Research Centers (1999)
11. Owre, S., Rushby, J.M., Shankar, N.: *PVS: A Prototype Verification System*. In: Kapur, D. (ed.) *Automated Deduction - CADE-11*. LNCS, vol. 607, pp. 748–752. Springer, Heidelberg (1992)
12. RTCA: *Final report of the RTCA board of directors' select committee on free flight*. Technical Report Issued 1-18-95, RTCA, Washington, DC (1995)
13. Di Vito, B.L.: *Manip User's Guide, Version 1.1*. NASA Langley Research Center, Hampton, Virginia (February 18, 2003)

An Introduction to Context Logic

Philippa Gardner and Uri Zarfaty

Imperial College, London

{pg,udz}@doc.ic.ac.uk

Abstract. This paper provides a gentle introduction to Context Logic. It contains work previously published with Calcagno [1,2], and is based on Gardner’s notes for her course on *Local Reasoning about Data Update* at the Appsem PhD summer school [3] and Zarfaty’s thesis [4].

1 Introduction

Structured data update is pervasive in computer systems: examples include heap update on local machines, information storage on hard disks, the update of distributed XML databases, and general term rewriting. Programs for manipulating such dynamically-changing data are notoriously difficult to write correctly. We therefore need analysis techniques and verification tools for reasoning about such programs. Hoare reasoning is a well-known technique for reasoning about programs that alter memory. In contrast, such reasoning has hardly been studied for other examples of update, such as tree update, and there has been little attempt at an integrated theory.

There has been a recent breakthrough in Hoare reasoning. Researchers previously used Hoare reasoning based on First-order Logic to specify how programs interacted with the *whole* memory. O’Hearn, Reynolds and Yang instead introduced *local* Hoare reasoning based on Separation Logic (SL) [5]. The key idea is to specify how programs interact with the small, local part of the memory (the footprint) touched by a program. Their work is proving to be essential for modular reasoning about large programs [6], and for reasoning about modern applications involving concurrent, distributed programming [7]. Inspired by this work, Calcagno, Gardner and Zarfaty studied local Hoare reasoning about tree update (XML update) [1]. We initially assumed that we could base our reasoning on Cardelli and Gordon’s Ambient Logic (AL) [8], since AL analyses static trees (firewalls, XML [9]) in a similar way to the SL-analysis of heaps. In fact, we proved that this is not possible [10]. Instead, we had to fundamentally change the way we reason about structured data, by introducing Context Logic (CL) for reasoning about data and contexts.

Local data update typically identifies the portion of data to be replaced, removes it, and inserts new data *in the same place*. CL reasons about both data and this place of insertion (contexts). In this paper, we provide a gentle introduction to the theory of CL. We first introduce the general theory of CL, describing the models and forcing semantics. We then describe an extension of CL which incorporates an additional zero formula for specifying empty data, since many

example models have such empty data. This extension yields interesting logical structure. Finally, we study the application of CL to specific examples of structured data: sequences, heaps, trees and terms. This paper illustrates that there is an intuitive, albeit ad hoc, way of applying CL to our examples of structured data. One future challenge is to capture the spirit of this informal intuition, by applying CL reasoning to an appropriate general account of structured data.

2 What Is Context Logic?

Context Logic (CL) is a logic which analyses both data and contexts using the simple idea of ‘context application’. Here we provide a general account of CL, analogous to Bunched Logic (BI) of O’Hearn and Pym [11,12]. CL consists of two sets of formulæ: one for reasoning about data, the other contexts. Both sets contain *additive* formulæ constructed from the standard boolean connectives, and *structural* formulæ constructed from an application connective and its two corresponding adjoints. In Section 4, we apply CL to specific models of structured data, extending CL to include specific formulæ determined by the model.

Definition 1 (CL formulæ). *The formulæ of CL consist of a set of data formulæ $P \in \mathcal{P}$ and context formulæ $K \in \mathcal{K}$, described by the grammars*

$$\begin{array}{lll} P ::= K \cdot P \mid K \triangleleft P & K ::= P \triangleright P \mid I & \text{structural formulæ} \\ \mid P \Rightarrow P \mid \text{false} & \mid K \Rightarrow K \mid \text{False} & \text{additive formulæ} \end{array}$$

The other standard formulæ are derived as usual. Binding precedence, from tightest to loosest, is \neg , \cdot , \wedge , \vee , $\{\triangleright, \triangleleft\}$ and \Rightarrow .

The nub of CL lies in its four structural formulæ. The *context application* formula $K \cdot P$ is satisfied by the result of inserting data satisfying P into contexts satisfying K . Its two right adjoints, meanwhile, express the corresponding adjunct implications. The *left-triangle* formula $K \triangleleft P$ is satisfied by some data if, whenever *any* context satisfying K is applied to it, the resulting data satisfies P . Similarly, the *right-triangle* formula $P_1 \triangleright P_2$ describes contexts that, when applied to *any* subdata satisfying P_1 , result in data satisfying P_2 . Finally, the *identity context* formula I describes a context or set of contexts that does not affect data, acting as a left-identity for context application.

We interpret CL formulæ using a forcing semantics. The CL models consist of a set of contexts, a set of data, a partial application function, and a subset of the contexts acting as the ‘identity contexts’. Data formulæ are interpreted on members of the data set, and context formulæ on members of the context set. A more general approach permits context application to be a relation [10], but partial functions are simpler, more intuitive and sufficient for this paper.

Definition 2 (CL models). *A CL model $\mathcal{M} = (\mathcal{C}, \mathcal{D}, \text{ap}, \text{I})$ consists of a set \mathcal{C} of contexts, a set \mathcal{D} of data, a partial application function $\text{ap}: \mathcal{C} \times \mathcal{D} \rightharpoonup \mathcal{D}$ and a subset $\text{I} \subseteq \mathcal{C}$ that acts as a left-identity for ap :*

$$\forall d \in \mathcal{D}. (\exists i \in \mathcal{I}. \text{ap}(i, d) = d) \wedge (\forall i \in \mathcal{I}. \text{ap}(i, d) \downarrow \Rightarrow \text{ap}(i, d) = d)$$

where $\text{ap}(c, d) \downarrow$ denotes that $\text{ap}(c, d)$ is defined.

Definition 3 (CL forcing semantics). Given a CL model \mathcal{M} , the forcing semantics is given by two satisfaction relations $\mathcal{M}, d \models_{\mathcal{D}} P$ and $\mathcal{M}, c \models_{\mathcal{C}} K$ defined inductively on the structure of data and context formulæ:

$$\begin{aligned} \mathcal{M}, d \models_{\mathcal{D}} K \cdot P &\Leftrightarrow \exists c \in \mathcal{C}, d' \in \mathcal{D}. (d = \text{ap}(c, d') \wedge \mathcal{M}, c \models_{\mathcal{C}} K \wedge \mathcal{M}, d' \models_{\mathcal{D}} P) \\ \mathcal{M}, d \models_{\mathcal{D}} K \triangleleft P &\Leftrightarrow \forall c \in \mathcal{C}. ((\text{ap}(c, d) \downarrow \wedge \mathcal{M}, c \models_{\mathcal{C}} K) \Rightarrow \mathcal{M}, \text{ap}(c, d) \models_{\mathcal{D}} P) \\ \mathcal{M}, d \models_{\mathcal{D}} P_1 \Rightarrow P_2 &\Leftrightarrow \mathcal{M}, d \models_{\mathcal{D}} P_1 \Rightarrow \mathcal{M}, d \models_{\mathcal{D}} P_2 \\ \mathcal{M}, d \models_{\mathcal{D}} \text{false} &\text{ never} \\ \mathcal{M}, c \models_{\mathcal{C}} P_1 \triangleright P_2 &\Leftrightarrow \forall d \in \mathcal{D}. ((\text{ap}(c, d) \downarrow \wedge \mathcal{M}, d \models_{\mathcal{D}} P_1) \Rightarrow \mathcal{M}, \text{ap}(c, d) \models_{\mathcal{D}} P_2) \\ \mathcal{M}, c \models_{\mathcal{C}} I &\Leftrightarrow c \in \mathcal{I} \\ \mathcal{M}, c \models_{\mathcal{C}} K_1 \Rightarrow K_2 &\Leftrightarrow \mathcal{M}, c \models_{\mathcal{C}} K_1 \Rightarrow \mathcal{M}, c \models_{\mathcal{C}} K_2 \\ \mathcal{M}, c \models_{\mathcal{C}} \text{False} &\text{ never} \end{aligned}$$

It is possible to prove a soundness and completeness result for CL, by interpreting CL as a modal logic and applying a general theory due to Salqvist to this particular setting [10,4].

Example 1 (CL models). The following are all models of CL:

1. $\text{Emp} = (\emptyset, \emptyset, \emptyset: \emptyset \rightarrow \emptyset, \emptyset)$, the *empty model*.
2. $\text{Fun}_{\mathcal{D}} = (\mathcal{D} \rightarrow \mathcal{D}, \mathcal{D}, \text{ap}, \{i\})$, the *function model*, where \mathcal{D} is an arbitrary set, contexts are total functions on \mathcal{D} , ap is function application, and i is the identity function on \mathcal{D} .
3. $\text{PartFun}_{\mathcal{D}, I} = (\mathcal{D} \rightharpoonup \mathcal{D}, \mathcal{D}, \text{ap}, I)$, the *partial function model*, where \mathcal{D} is an arbitrary set, contexts are partial functions on \mathcal{D} , ap is function application, and I is a set of partial identity functions, the union of whose domains is the whole of \mathcal{D} . Note that different choices of I result in different models.
4. $\text{Mon}_{\mathcal{D}, \circ, 0} = (\mathcal{D}, \mathcal{D}, \circ, \{0\})$, the *monoid model*, where \mathcal{D} is a monoid with monoidal operator \circ and identity 0 .
5. $\text{PartMon}_{\mathcal{D}, \circ, 0} = (\mathcal{D}, \mathcal{D}, \circ, \{0\})$, the *partial monoid model*, where \mathcal{D} is a partial monoid with monoidal operator \circ and identity 0 .

Example 2 (CL constructions). Given two CL models, $\mathcal{M}_1 = (\mathcal{C}_1, \mathcal{D}_1, \text{ap}_1, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{C}_2, \mathcal{D}_2, \text{ap}_2, \mathcal{I}_2)$, it is possible to construct the following derived models, for arbitrary $c_i \in \mathcal{C}_i, d_i \in \mathcal{D}_i$:

1. $\mathcal{M}_1 + \mathcal{M}_2 = (\mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{D}_1 \cup \mathcal{D}_2, \text{ap}, \mathcal{I}_1 \cup \mathcal{I}_2)$, the *union model*, where $\text{ap}(c_i, d_j) = \text{ap}_i(c_i, d_j)$ if $i = j$ and is undefined otherwise;
2. $\mathcal{M}_1 \times \mathcal{M}_2 = (\mathcal{C}_1 \times \mathcal{C}_2, \mathcal{D}_1 \times \mathcal{D}_2, \text{ap}, \mathcal{I}_1 \times \mathcal{I}_2)$, the *product model*, where $\text{ap}((c_1, c_2), (d_1, d_2)) = (\text{ap}_1(c_1, d_1), \text{ap}_2(c_2, d_2))$ if both applications are defined, and is undefined otherwise;
3. $\mathcal{M}_1^{\mathcal{A}} = (\mathcal{A} \times \mathcal{C}_1, \mathcal{A} \times \mathcal{D}_1, \text{ap}, \mathcal{A} \times \mathcal{I}_1)$, the *set-indexed model*, where \mathcal{A} is an arbitrary index set, and $\text{ap}((a_1, c_1), (a_2, d_1)) = (a_1, \text{ap}_1(c_1, d_1))$ if $a_1 = a_2$ and is undefined otherwise, for arbitrary $a_1, a_2 \in \mathcal{A}$.

Note that the set-indexed model $\mathcal{M}_1^{\mathcal{A}}$ corresponds to $\mathcal{M}_{\mathcal{A}} \times \mathcal{M}_1$, where $\mathcal{M}_{\mathcal{A}} \triangleq (\mathcal{A}, \mathcal{A}, \text{ap}, \mathcal{A})$ and $\text{ap}(\mathbf{a}_1, \mathbf{a}_2) = \mathbf{a}_1$ if $\mathbf{a}_1 = \mathbf{a}_2$ and is undefined otherwise.

In addition to the core definitions above, it is also useful to define various derived formulæ. Two simple but useful examples are the existential duals of the context application adjoints, and formulæ corresponding to somewhere and everywhere modalities.

Definition 4 (Derived formulæ). *The existential duals of the application adjoints (left) and the somewhere and everywhere modalities (right) are defined by:*

$$\begin{aligned} P_1 \blacktriangleright P_2 &\triangleq \neg(P_1 \triangleright \neg P_2) & \Diamond P &\triangleq \text{True} \cdot P \\ K \blacktriangleleft P &\triangleq \neg(K \triangleleft \neg P) & \Box P &\triangleq \neg(\Diamond \neg P) \end{aligned}$$

The existential duals of the adjoints describe existential, as opposed to universal, properties. Thus $K \blacktriangleleft P$ is satisfied by some data if it is *possible* to apply a context satisfying K to that data and obtain data satisfying P . Similarly, $P_1 \blacktriangleright P_2$ is satisfied by a context if it is possible to insert in that context data satisfying P_1 and obtain data satisfying P_2 . In our presentation of CL as a modal logic [10], these existential duals are the fundamental connectives rather than the universally-quantified right adjoints. The somewhere and everywhere modalities, meanwhile, concern the possibility and necessity of context splitting. The former, $\Diamond P$, states that it is possible to split data into a context and some subdata satisfying P . The latter, $\Box P$, states that however one splits some data into a context and subdata, the subdata must satisfy P .

3 Extending Context Logic

Apart from the basic context structure given above, certain groups of CL models display additional shared structural properties: one example is the many models that permit composition of contexts; another is the models that contain ‘empty’ data elements. CL can easily be extended to analyse such structural properties.

Here we concentrate on extending CL to analyse empty data. Many (but not all) structured data models have an element, or set of elements, that correspond to empty data: heap models, for example, have the empty heap, and tree models the empty tree. In contrast, fixed-arity terms (discussed in Section 4) have no natural empty element. To analyse empty data, we add a *zero* formula to CL.

Definition 5 (CL₀ formulæ). *The formulæ of Context Logic with Zero (CL₀) consist of the same formulæ as in CL, with an additional formula for zero:*

$$P ::= \dots \mid 0 \quad K ::= \dots$$

Empty data elements in structured data typically satisfy a number of common properties. For example, our trees (Defn. 19) always contain the empty tree as subdata. It is always possible to apply the empty tree to a tree context and obtain

a unique result. It is not possible to split the empty tree into a context and a non-empty tree. Here we extend CL models with the condition that the projection relation obtained by placing a zero element in a context is a total, surjective function. This condition corresponds to the first two properties mentioned for trees, and is enough to provide some interesting results.

Definition 6 (CL_\emptyset models). A CL_\emptyset model $\mathcal{M} = (\mathcal{C}, \mathcal{D}, \text{ap}, \text{l}, 0)$ is a CL model with the addition of a zero set $0 \subseteq \mathcal{D}$ for which the relation $p \subseteq \mathcal{C} \times \mathcal{D}$ defined by $(c, d) \in p \Leftrightarrow \exists o \in 0. \text{ap}(c, o) = d$ is a total, surjective function.

Definition 7 (CL_\emptyset forcing semantics). Given a CL_\emptyset model, the forcing semantics extends that of CL, given in Defn. 3, with the additional case:

$$\mathcal{M}, d \models_{\mathcal{D}} 0 \Leftrightarrow d \in 0$$

Example 3 (CL_\emptyset models). The following extensions to CL models from Example 1 are all models of CL_\emptyset :

1. $\text{Emp} = (\emptyset, \emptyset, \emptyset : \emptyset \rightarrow \emptyset, \emptyset, \emptyset)$, the *empty model*.
2. $\text{Fun}_{\mathcal{D}} = (\mathcal{D} \rightarrow \mathcal{D}, \mathcal{D}, \text{ap}, \{i\}, \{d\})$, the *function model*, where the zero set contains an arbitrary data element $d \in \mathcal{D}$.
3. $\text{Mon}_{\mathcal{D}, \circ, 0} = (\mathcal{D}, \mathcal{D}, \circ, \{0\}, \{0\})$, the *monoid model*.
4. $\text{PartMon}_{\mathcal{D}, \circ, 0} = (\mathcal{D}, \mathcal{D}, \circ, \{0\}, \{0\})$, the *partial monoid model*. Note that the commutative $\text{PartMon}_{\mathcal{D}, \circ, 0}$ models, where \circ is commutative, correspond to the BI models characterised by the partial commutative monoids $(\mathcal{D}, \circ, 0)$ [12].

The CL model $\text{PartFun}_{\mathcal{D}, I}$ does not admit a zero, since nothing can be placed inside the everywhere undefined context, contradicting the totality condition.

Example 4 (CL_\emptyset constructions). Given two CL_\emptyset models, $\mathcal{M}_1 = (\mathcal{C}_1, \mathcal{D}_1, \text{ap}_1, \text{l}_1, 0_1)$ and $\mathcal{M}_2 = (\mathcal{C}_2, \mathcal{D}_2, \text{ap}_2, \text{l}_2, 0_2)$, it is possible to extend the constructions given in Example 2 as follows:

1. $\mathcal{M}_1 + \mathcal{M}_2 = (\mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{D}_1 \cup \mathcal{D}_2, \text{ap}, \text{l}_1 \cup \text{l}_2, 0_1 \cup 0_2)$, the *union model*;
2. $\mathcal{M}_1 \times \mathcal{M}_2 = (\mathcal{C}_1 \times \mathcal{C}_2, \mathcal{D}_1 \times \mathcal{D}_2, \text{ap}, \text{l}_1 \times \text{l}_2, 0_1 \times 0_2)$, the *product model*;
3. $\mathcal{M}_1^{\mathcal{A}} = (\mathcal{A} \times \mathcal{C}_1, \mathcal{A} \times \mathcal{D}_1, \text{ap}, \mathcal{A} \times \text{l}_1, \mathcal{A} \times 0_1)$, the *set-indexed model*.

One particularly interesting property of zero is its use in defining a natural structural binary connective, plus its two corresponding right adjoints, on data formulæ. This connective, though not in general commutative or associative, is called $*$, as it corresponds to the $*$ of BI for the commutative $\text{PartMon}_{\mathcal{D}, \circ, 0}$ models described above.

Definition 8 (Star connective). For CL_\emptyset , the logical data connective $*$ and its two adjoints are defined as follows:

$$\begin{aligned} P_1 * P_2 &\triangleq (0 \triangleright P_1) \cdot P_2 & P_1 * - P &\triangleq (0 \triangleright P_1) \triangleleft P \\ P_2 - * P &\triangleq \neg(\neg(P_2 \triangleright P) \cdot 0) \end{aligned}$$

The formula $P_1 * P_2$ splits the data into two parts: subdata satisfying P_2 and a context which, when zero is put in its hole, satisfies P_1 . The intuition for the definitions of the two right adjoints is not so obvious; these definitions are determined by the required adjoint properties (see [4] for details). The interpretation of $*$ on specific structured data models is given in the following section. In addition, $*$ can be used to prove a strong embedding result for BI. As noted previously, the BI models $(\mathcal{D}, \circ, 0)$ correspond to the commutative $PartMon_{\mathcal{D}, \circ, 0}$ models. For these models, the derived $*$ of CL coincides precisely with the $*$ connective of BI, and the two derived adjoints coincide with the $-*$ of BI. Furthermore, we have the following expressivity result:

Theorem 1 (BI embedding result). *Boolean BI for the model $(\mathcal{D}, \circ, 0)$ is just as expressive as CL_{\emptyset} for the model $PartMon_{\mathcal{D}, \circ, 0}$.*

Proof. See [4].

4 Structured Data Models

CL is a natural logic for reasoning about structured data models. This final section illustrates the application of CL to sequences, heaps, trees and terms.

4.1 Sequences

Sequences provide a simple example of structured data that illustrates well the differences in reasoning between CL and BI. The sequences in this section are generated from an alphabet \mathcal{A} , with a concatenation operation $:$ and an empty sequence 0 . Sequence contexts are sequences with a hole in the middle, while context application corresponds to inserting a sequence into the hole.

Definition 9 (Sequences). *Given an infinite set $\mathcal{A} = \{a, \dots\}$ of elements, sequences $s \in \mathcal{S}$ and sequence contexts $c \in \mathcal{C}$ are defined by the grammars*

$$s ::= 0 \mid a \mid s : s \qquad c ::= - \mid c : s \mid s : c$$

with a structural congruence \equiv specifying that the concatenation operation $:$ is associative with identity 0 . The insertion of sequences into sequence contexts is given by a total application function (which is well-defined up to congruence):

$$\text{ap}(-, s) = s \quad \text{ap}(c : s', s) = \text{ap}(c, s) : s' \quad \text{ap}(s' : c, s) = s' : \text{ap}(c, s)$$

It is easy to show that $Seq = (\mathcal{S}, \mathcal{C}, \text{ap}, \{-\}, \{0\})$ is a model of CL_{\emptyset} . CL_{\emptyset} formulae, however, are not enough to provide useful reasoning about sequences: for example, there is no formula which differentiates between different elements. We therefore add sequence-specific formulae to the logic for analysing the sequence structure, just as SL adds heap-specific formulae to BI in order to analyse heaps.

The additional structural formulae are chosen to correspond directly to the data structure definitions in Defn. 9. There are already formulae corresponding to the empty context $-$ and empty sequence 0 , which means we need to add formulae only for singleton sequences a and the concatenation operation $:$.

Definition 10 (CL_{seq} formulæ). *The formulæ of Context Logic for Sequences (CL_{seq}) consist of the same formulæ as CL_\emptyset (Defn. 5), with the addition of special formulæ for singleton sequences and context concatenation:*

$$P ::= \cdots \mid \mathbf{a} \in \mathcal{A} \quad K ::= \cdots \mid P : K \mid K : P$$

The interpretation of the new formulæ is straightforward: the data formula \mathbf{a} is satisfied by the empty singleton sequence \mathbf{a} ; the context formula $P : K$ is satisfied by the result of concatenating a context satisfying K to the right of a sequence satisfying P ; and the context formula $K : P$ is satisfied by the result of concatenating the context to the left. Thus, for example, both $(\mathbf{a} : I) \cdot \mathbf{b}$ and $(I : \mathbf{b}) \cdot \mathbf{a}$ describe the two-element sequence $\mathbf{a} : \mathbf{b}$. Note that it is not necessary to include a formula $P_1 : P_2$ to express the concatenation of two sequences, since this is derivable using the context concatenation connectives; the formula $P_1 : P_2$ can be expressed either as $(P_1 : I) \cdot P_2$ or as $(I : P_2) \cdot P_1$. While the connective can therefore be omitted from the logic, its simpler notation is still used for shorthand.

Definition 11 (CL_{seq} forcing semantics). *The semantics of CL_{seq} extends that of CL_\emptyset for the model Seq , defined in Defn. 7, by the following cases:*

$$\begin{aligned} s \models_{\mathcal{D}} \mathbf{a} & \quad \Leftrightarrow s \equiv \mathbf{a} \\ c \models_C P : K & \Leftrightarrow \exists s \in \mathcal{S}, c' \in \mathcal{C}. (s \models_{\mathcal{D}} P \wedge c' \models_C K \wedge c \equiv s : c') \\ c \models_C K : P & \Leftrightarrow \exists s \in \mathcal{S}, c' \in \mathcal{C}. (s \models_{\mathcal{D}} P \wedge c' \models_C K \wedge c \equiv c' : s) \end{aligned}$$

As noted in Sections 2 and 3, it is possible to define derived operators such as the somewhere modality \Diamond (Defn. 4) and the $*$ connective on data (Defn. 8). The somewhere modality describes arbitrary subsequences: $\Diamond P$ holds for a sequence if it contains a subsequence satisfying P . The star connective, meanwhile, describes the insertion of one sequence into another: $P_1 * P_2$ holds for a sequence if it is possible to remove a subsequence satisfying P_2 and obtain a sequence satisfying P_1 . In addition to these derived connectives, it is also possible to define the adjoints of the concatenation connective, which describe the addition of sequences to either side of a sequence. Thus, the adjoint $P_1 \vdash P_2$ holds if whenever a sequence satisfying P_1 is concatenated on the right, the result satisfies P_2 ; the other adjoint $P_1 \vdash^- P_2$ is similar, involving concatenating on the left.

Definition 12 (Concatenation adjoints). *The adjoints of concatenation are:*

$$P_1 \vdash P_2 \triangleq (I : P_1) \triangleleft P_2 \quad P_1 \vdash^- P_2 \triangleq (P_1 : I) \triangleleft P_2$$

Example 5 (CL_{seq} formulæ). The following are examples of formulæ in CL_{seq} , together with their interpretations on sequences:

1. $\mathbf{a} : \mathbf{b} : \mathbf{a}$ or $(\mathbf{a} : I) \cdot ((\mathbf{b} : I) \cdot \mathbf{a})$ or $(\mathbf{a} : I) \cdot ((I : \mathbf{a}) \cdot \mathbf{b})$
the sequence containing \mathbf{a} then \mathbf{b} then \mathbf{a} .
2. $\mathbf{a} : \text{true}$ or $(\mathbf{a} : I) \cdot \text{true}$ or $(I : \text{true}) \cdot \mathbf{a}$
any sequence beginning with an \mathbf{a} .

3. $a * \text{true} \text{ or } (0 \triangleright a) \cdot \text{true} \text{ or } (\text{true} : a) \vee (a : \text{true})$
a sequence that either begins or ends with an a .
4. $\Diamond a \text{ or } \text{True} \cdot a \text{ or } \text{true} * a \text{ or } \text{true} : a : \text{true}$
any sequence that contains an a .
5. $(a \vdash P) : b \text{ or } ((a : I) \triangleleft P) : b$
a sequence ending in b that would satisfy P were the b replaced by an a .
6. $(a * P) * b \text{ or } (0 \triangleright ((a \triangleright P) \cdot 0)) \cdot b$
a sequence containing a b that would satisfy P were the b removed and an a added anywhere in the sequence.
7. $(a \triangleright P) \cdot b$
a sequence containing a b that would satisfy P were the b removed and an a added *in the same place*.
8. $0^+ \triangleq \text{true}, (n+1)^+ \triangleq \neg I \cdot n^+ \text{ and } n \triangleq n^+ \wedge \neg(n+1)^+ \text{ for all } n \in \mathbb{N}$
any sequence with at least n elements (for n^+) or precisely n (for n).
9. $\Box(1 \Rightarrow a) \text{ or } \neg\Diamond(1 \wedge \neg a)$
any sequence containing just a 's

4.2 Heaps and Stores

Heaps form the main reasoning target of SL, and provide a good example where CL reasoning collapses to BI reasoning. A heap is typically viewed as a set of locations, with values at each location. Heap contexts, meanwhile, correspond to heaps with a hole. Unlike in sequences, this hole is not ‘located’ anywhere, making heap contexts structurally equivalent to heaps. For simplicity, the heaps considered in this section contain only pointer values, which are either location addresses or ‘nil’. To allow the modelling of pointer arithmetic, addresses are represented by the positive integers, while nil is represented by zero.

Definition 13 (Heaps). Heaps $h \in \mathcal{H}$ are defined by the grammar

$$h ::= \text{emp} \mid n \mapsto v \mid h * h$$

with $n \in \mathbb{N}^+, v \in \mathbb{N}$, and a structural congruence \equiv specifying that the composition operator $*$ is associative and commutative with identity emp . Well-formed heaps are those with unique locations, making $*$ a partial operation.

Apart from heaps, SL also considers variable stores, which map variables to values. This allows reasoning about variable-based heap update languages. To refer to locations and values, we also include value expressions, which incorporate constants, variables and pointer arithmetic.

Definition 14 (Variable stores). Given an infinite set $\text{Var} = \{x, y, \dots\}$ of variables, a variable store $s \in \mathcal{S}$ is a total function $s : \text{Var} \rightarrow \mathbb{N}$.

Definition 15 (Expressions). A value expression E is given by the grammar

$$E ::= v \mid x \mid E + E \mid E - E$$

where $v \in \mathbb{N}$. The valuation of E on a store s is written $\llbracket E \rrbracket s$.

It is easy to show that $\text{Heap} = (\mathcal{H}, \mathcal{H}, *, \{\text{emp}\}, \{\text{emp}\})$ is a simple monoidal model of CL_\emptyset . By Thm. 1, we see that CL_\emptyset for this model is just as expressive as BI for the model $(\mathcal{H}, *, \text{emp})$. A more difficult question is how to incorporate store variables into this model. In fact, this is straightforward using the set-indexing construction defined in Example 2: $\text{Heap}^{\mathcal{S}}$ describes a heap-store model where every heap and heap context is indexed by a store $\mathbf{s} \in \mathcal{S}$ and context application is only defined for states with matching stores.

Again, we extend the logic with formulæ specific to the data model. In this case, however, we need only add formulæ for singleton heaps and (if desired) for variable quantification. It is unnecessary to add any formulæ for the star operation, as this is already expressible using context application: $P_1 * P_2$ is defined as $(0 \triangleright P_1) \cdot P_2$ and corresponds to the derived star operation from Section 3. The formula expressing singleton heaps is written $E \mapsto F$, and uses expressions to refer to both the heap location and its value. Note that the presence of variables in the expression language means that this formula can depend on the store, as well as the heap.

Definition 16 (CL_{heap} formulæ). *The formulæ of Context Logic for Heaps (CL_{heap}) consist of the same formulæ as CL_\emptyset with the addition of an atomic formula for specifying a cell in the heap, and quantification over variables:*

$$P ::= \dots \mid E \mapsto F \mid \exists n. P \quad K ::= \dots \mid \exists n. K$$

Definition 17 (CL_{heap} forcing semantics). *The semantics of CL_{heap} extends that of CL_\emptyset for the model Heap , defined in Defn. 7, by the following cases:*

$$\begin{aligned} \mathbf{s}, \mathbf{h} \models_{\mathcal{D}} E \mapsto F &\Leftrightarrow \mathbf{h} \equiv \llbracket E \rrbracket \mathbf{s} \mapsto \llbracket F \rrbracket \mathbf{s} \\ \mathbf{s}, \mathbf{h} \models_{\mathcal{D}} \exists x. P &\Leftrightarrow \exists v \in \mathbb{N}. [\mathbf{s} | x \leftarrow v], \mathbf{h} \models_{\mathcal{D}} P \\ \mathbf{s}, \mathbf{c} \models_{\mathcal{C}} \exists x. K &\Leftrightarrow \exists v \in \mathbb{N}. [\mathbf{s} | x \leftarrow v], \mathbf{c} \models_{\mathcal{C}} K \end{aligned}$$

The same derived formulæ can be defined for heaps as for sequences, and all correspond to standard SL formulæ. Moreover, it is possible to use the \triangleright composition adjoint, together with the empty context I , to derive a formula for expression equality. Expression inequality, likewise, can be expressed using equality and quantification.

Definition 18 (Booleans). *Equality and inequality are defined as follows:*

$$\begin{aligned} (E = F) &\triangleq (I \wedge (E \mapsto F \triangleright F \mapsto F)) \cdot \text{true} \\ (E < F) &\triangleq \exists x. (E + x = F) \wedge \neg(x = 0) \end{aligned}$$

Example 6 (CL_{heap} formulæ). The following are examples of formulæ in CL_{heap} , together with their interpretation on heaps and variable stores:

1. $1 \mapsto 2 * 2 \mapsto \text{nil}$ or $(0 \triangleright (1 \mapsto 2)) \cdot (2 \mapsto \text{nil})$
the 2-cell heap representing a null-terminated list with locations 1 and 2.
2. $x \mapsto y * y \mapsto x$
a 2-cell heap representing a circular list from (the value of) x to (the value of) y and back. Note that x cannot equal y .

3. $x \mapsto y \wedge y \mapsto x$ or $x \mapsto x \wedge x = y$
a 1-cell heap representing a circular list at x , which must equal y .
4. $\Diamond x \mapsto y \wedge \Diamond y \mapsto x$
an arbitrary heap either containing a circular list at x , if $x = y$, or one from x to y and back, if $x \neq y$.
5. $(1 \mapsto 2 \triangleright P) \cdot (\exists x.(1 \mapsto x))$ or $(1 \mapsto 2 \multimap P) \cdot (\exists x.(1 \mapsto x))$
a heap with location 1 that would satisfy P were the value at 1 updated to 2.

4.3 Trees

The next example considered is trees. Trees were the original reasoning target of CL. Indeed, one of the main reasons for introducing CL was that AL was not expressive enough to be used for local Hoare reasoning about tree update [1,10]. In contrast, CL forms a natural specification language for tree update commands. Our original work [1,13] used fairly complicated tree structures including pointers, data and variable. For simplicity, the trees considered here consist of a basic unordered forest structure with unique node identifiers $n \in \mathcal{N}$, and are constructed using a branching operation $n[-]$ and a commutative composition operation $|$. We also consider tree contexts, which are simply trees (forests) with a hole inside.

Definition 19 (Trees and tree contexts). Trees $t \in \mathcal{T}$ and contexts $c \in \mathcal{C}$ are defined as follows, together with the obvious context application operation.

$$\begin{array}{ll}
 & \text{ap}(-, t) \triangleq t \\
 t ::= 0 \mid n[t] \mid t \mid t & \text{ap}(n[c], t) \triangleq n[\text{ap}(c, t)] \\
 c ::= - \mid n[c] \mid t \mid c \mid c \mid t & \text{ap}(t \mid c, t') \triangleq t \mid (\text{ap}(c, t')) \\
 & \text{ap}(c \mid t, t') \triangleq \text{ap}(c, t') \mid t
 \end{array}$$

Trees and contexts satisfy a structural congruence \equiv stating that $|$ is associative, and that 0 is its left and right identity. Well-formed trees and tree contexts are those with unique identifiers, making both the structural constructors and context application partial operations.

It is easy to show that $\text{Tree} = (\mathcal{T}, \mathcal{C}, \text{ap}, \{-\}, \{0\})$ is a model of CL_\emptyset . As before, we introduce new formulæ to the logic that correspond to the tree and context data structure definitions. This can be done at just the context level, with the data connectives derivable using context application. Thus, we introduce a context composition formula $P \mid K$, describing the composition of a tree satisfying P and a context satisfying K , and use it to define tree composition $P \mid P'$ as $(P \mid I) \cdot P'$. Likewise, we define a context branching formula $n[K]$, describing a tree with a root node n and a subcontext satisfying K , and use it to derive tree branching $n[P]$ as $n[I] \cdot P$.

Definition 20 (CL_{tree} formulæ). The formulæ of Context Logic for Trees (CL_{tree}) consist of the same formulæ as CL_\emptyset with the addition of formulæ representing branching contexts and parallel composition:

$$P ::= \dots \quad K ::= \dots \mid n[K] \mid P \mid K$$

Definition 21 (CL_{tree} forcing semantics). *The semantics of CL_{tree} extends that of CL₀ for the model Tree, defined in Defn. 7, by the following cases:*

$$\begin{aligned} s, c \models_C n[K] &\Leftrightarrow \exists n \in \mathcal{N}, c' \in \mathcal{C}. (s, c' \models_C K \wedge c \equiv n[c']) \\ s, c \models_C P \mid K &\Leftrightarrow \exists t \in \mathcal{T}, c' \in \mathcal{C}. (s, t \models_D P \wedge s, c' \models_C K \wedge c \equiv t \mid c') \end{aligned}$$

As before, we define the derived formulæ \Diamond and $*$. The somewhere modality \Diamond allows us to reason about properties satisfied by subtrees arbitrarily deep inside a tree: $\Diamond P$ holds for any tree with a subtree satisfying P . The star connective $*$, meanwhile, describes an interesting way of combining two trees: the embedding $(0 \triangleright P)$ describes adding a hole at some arbitrary location in a tree satisfying P ; thus $P_1 * P_2 \triangleq (0 \triangleright P_1) \cdot P_2$ describes the insertion of a tree satisfying P_2 into an arbitrary location inside a tree satisfying P_1 . Note that this operation is neither commutative nor associative.

Like for heaps, we can also derive a formula expressing value equality. Similarly, we derive the adjoints of both composition and branching, like we did for sequence concatenation. These correspond to standard AL connectives, and illustrate how AL can be neatly expressed inside CL.

Definition 22 (Derived formulæ). *Value equality (left) and the adjoints of composition and branching (right) are defined as follows:*

$$\begin{aligned} n_1 = n_2 &\triangleq (I \wedge (n_1[0] \triangleright n_2[0])) \cdot \text{true} & P_1 \dashv P_2 &\triangleq (P_1 \mid I) \triangleleft P_2 \\ P @ n &\triangleq n[I] \triangleleft P \end{aligned}$$

Example 7 (CL_{tree} formulæ). The following are examples of formulæ in CL_{tree}, together with their interpretations on trees:

1. $n[\text{true}]$ or $n[-] \cdot \text{true}$
a tree with root node n .
2. $\Diamond n[\text{true}]$ or $\text{True} \cdot n[\text{true}]$ or $\text{true} * n[\text{true}]$
a tree containing a node n .
3. $n[0] * \text{true}$ or $(0 \triangleright n[0]) \cdot \text{true}$ or $n[\text{true}] \vee (n[0] \mid \text{true})$
a tree with root node n and either a subforest or siblings.
4. $(n[\text{true}] \blacktriangleright P) \cdot n[0]$
a tree containing an orphan node n to which it is possible to add a subforest and obtain a tree satisfying P ; in other words, the result of deleting the subforest at n from a tree satisfying P .
5. $(n[0] \triangleright P) \cdot n[\text{true}]$
a tree containing n that would satisfy P were the subforest at n removed; in other words, the result of ‘restoring’ a subforest at n to a tree satisfying P .

4.4 Terms

The final example in this paper consists of simple terms. These are constructed from a set of function symbols \mathcal{F} with fixed arities. For us, terms are an interesting example of structured data, as they do not generally contain a zero element in the sense of Section 3. The fixed arities of the function symbols mean that

there can be no ‘invisible’ empty term as for heaps and trees, although there are constant terms of zero arity. Assuming there is more than one such constant, then no set of terms defines a total surjective function from contexts to terms: surjectivity means the set must contain all the zero arity constants; functionality, meanwhile, means that the set must be a singleton. In the case where there is just one constant, then that constant does play the rôle of a zero element.

Definition 23 (Terms). *Given a set $\mathcal{F} = \{f, \dots\}$ of function symbols and a signature $\Sigma: \mathcal{F} \rightarrow \mathbb{N}$ mapping function symbols to arities (where at least one symbol has arity zero), terms $t \in \mathbb{T}$ and term contexts $c \in \mathcal{C}$ are defined by:*

$$\begin{aligned} t &::= f(t_1, \dots, t_k) & k &= \Sigma(f) \\ c &::= - \mid f(t_1, \dots, t_{i-1}, c, t_{i+1}, \dots, t_k) & k &= \Sigma(f) \geq i \geq 1 \end{aligned}$$

The insertion of terms into term contexts is given by:

$$\text{ap}(-, t) = t \quad \text{ap}(f(t_1, \dots, c, \dots, t_k), t) = f(t_1, \dots, \text{ap}(c, t), \dots, t_k)$$

It is easy to show that $\text{Term} = (\mathbb{T}, \mathcal{C}, \text{ap}, \{-\})$ forms a model of CL. As before, we extend CL with special formulæ corresponding to the data structure definitions. We add formulæ of the form $f(P_1, \dots, K, \dots, P_k)$, which enable us to define a term connective $f(P_1, P_2, \dots, P_k)$ as $f(I, P_2, \dots, P_k) \cdot P_1$. We also directly add the zero-arity functions as formulæ.

Definition 24 (CL_{term} formulæ). *The formulæ of Context Logic for Terms (CL_{term}) consist of the same formulæ as CL with the addition of formulæ for zero-arity terms and term contexts: assuming $\Sigma(f) = 0$ and $\Sigma(g) = k \geq i \geq 1$,*

$$P ::= \dots \mid f \quad K ::= \dots \mid g(P_1, \dots, P_{i-1}, K, P_{i+1}, \dots, P_k)$$

Definition 25 (CL_{term} forcing semantics). *The semantics of CL_{term} extends that of CL for the model Term , defined in Defn. 3, by the following cases:*

$$\begin{aligned} s, t &\models_{\mathcal{D}} f \Leftrightarrow t = f \\ s, c &\models_{\mathcal{C}} g(P_1, \dots, P_{i-1}, K, P_{i+1}, \dots, P_k) \\ &\Leftrightarrow \exists \mathbf{t_j} \in \mathbb{T}, c \in \mathcal{C}. (t = g(t_1, \dots, c, \dots, t_k) \wedge s, c \models_{\mathcal{C}} K \wedge \bigwedge_{\substack{1 \leq j \leq k \\ j \neq i}} s, t_j \models_{\mathcal{D}} P_j) \end{aligned}$$

As before, we can derive operators such as the somewhere modality. Unlike for sequences and trees however, we cannot derive the $*$ operator, since this depends on the presence of a zero. This illustrates the fact that there is no general way of combining two terms without adding or removing function symbols.

Example 8 (CL_{term} formulæ). Consider a function symbol set $\mathcal{F} = \mathbb{N} \cup \{-, +, \times\}$, where the naturals \mathbb{N} are constants of arity zero, $-$ represents the unary minus and has arity one, and $+$ and \times have arity two. Then the following are examples of CL_{term} formulæ, together with their interpretations on terms:

1. $\times(2, +(1, 3))$ or $\times(I, +(1, 3)) \cdot 2$ or $\times(2, I) \cdot +(1, 3)$
the multiplication term $\times(2, +(1, 3))$.
2. $+(true, true)$
any addition term.
3. $\Diamond\neg\neg(true)$
a term that contains a double negation.
4. $nat \triangleq \neg((\neg I) \cdot true)$
any zero-arity term: that is, a natural number.
5. $\Box(+ (true, true) \vee (\neg(I) \vee I) \cdot nat)$
a potentially nested sum of positive or negative integers.
6. $(+(1, 1) \triangleright P) \cdot 2$
a term that contains a 2 which would satisfy P were the 2 replaced by the term $+(1, 1)$; in other words, this describes the result of performing a single rewrite on a term satisfying P , folding $+(1, 1)$ into a constant.

5 Additional Work

Whilst this paper provides a gentle introduction to CL, CL has been explored in much more depth elsewhere (for example, see [2] and Zarfaty's thesis [4]). In particular, in [4] there is a detailed account of special classes of formulæ with interesting properties, which is not touched upon here.

A different presentation of CL as a modal logic is given in [10]. Using a general theorem of modal logic due to Salqvist, this paper presents a completeness proof for CL. Perhaps more significantly, the paper provides *parametric* inexpressivity results for CL and SL, demonstrating for example that CL for trees is strictly more expressive than AL, and in addition that SL (CL for heaps) is strictly more expressive than First-order Logic with additional atomic heap formulæ. These inexpressivity results capture our intuition that the structural connectives are a genuine source of additional expressivity. They are in marked contrast with previous results of Lozes [14,15], showing that the structural formulæ for SL and AL can be eliminated using a weaker form of expressivity.

We introduced CL to provide local Hoare reasoning about tree update. This update story has been omitted from this paper. It has been published in a number of places. The original CL paper [1] and the journal paper [2] demonstrate how to apply local Hoare reasoning based on CL to tree update, heap update and local term rewriting. In [13], our reasoning about tree update is extended to incorporate pointers, query languages and updates at multiple locations. This paper raised some particularly interesting questions regarding the rôle of so-called small specifications in local reasoning.

A variety of other work is on-going and as yet unpublished. Calcagno, Dinsdale-Young and Gardner are proving adjoint elimination results for CL applied to trees (in the style of Lozes), which require an extension of CL to include context composition and multi-holed contexts. Gardner and Zarfaty are studying the integration of CL reasoning about high-level tree update with SL reasoning about a low-level implementation of tree update. With Smith, we are also providing a formal specification of DOM [16], a high-level specification language for

XML update written in English. Finally, Gardner and Raza are studying an abstract account of local Hoare reasoning, following Calcagno, O'Hearn and Yang's work on abstract SL [17].

References

1. Calcagno, C., Gardner, P., Zarfaty, U.: Context logic & tree update. In: Proceedings of POPL, pp. 271–282. ACM Press, New York (2005)
2. Calcagno, C., Gardner, P., Zarfaty, U.: Local reasoning about data update (to appear)
3. Gardner, P.: A note on context logic Notes accompanying a lecture course for the APPSEM summer school (2005)
4. Zarfaty, U.: Context Logic and Tree Update. PhD thesis, Imperial College (2007)
5. O'Hearn, P., Reynolds, J., Yang, H.: Local reasoning about programs that alter data structures. In: Fribourg, L. (ed.) CSL 2001 and EACSL 2001. LNCS, vol. 2142, pp. 1–19. Springer, Heidelberg (2001)
6. Berdine, J., Calcagno, C., Cook, B., Distefano, D., O'Hearn, P., Wies, T., Yang, H.: Shape analysis for composite data structures. CAV 2007 (to appear)
7. O'Hearn, P.: Resources, concurrency and local reasoning. Theoretical Computer Science 375(1), 271–307 (2007)
8. Cardelli, L., Gordon, A.: Anytime, anywhere: modal logics for mobile ambients. In: Proceedings of POPL, pp. 365–377. ACM Press, New York (2000)
9. Cardelli, L., Gordon, A.: TQL: a query language for semistructured data based on the ambient logic. Mathematical Structures in Computer Science 14(3), 285–327 (2004)
10. Calcagno, C., Gardner, P., Zarfaty, U.: Context logic as modal logic: Completeness and parametric inexpressivity. In: Proceedings of POPL, ACM Press, New York (2007)
11. O'Hearn, P.W., Pym, D.: The logic of bunched implications. Bulletin of Symbolic Logic 5(2), 215–244 (1999)
12. Pym, D., O'Hearn, P.W., Yang, H.: Possible worlds and resources: the semantics of BI. Theoretical Computer Science 315(1), 257–305 (2004)
13. Zarfaty, U., Gardner, P.: Local reasoning about tree update. In: Proceedings of MFPS. ENTCS, vol. 158, pp. 399–424. Elsevier, Amsterdam (2006)
14. Lozes, É.: Elimination of spatial connectives in static spatial logics. Theoretical Computer Science 330(3), 475–499 (2005)
15. Lozes, E.: Comparing the expressive power of separation logic and classical logic. Short Presentation (2004)
16. W3C: DOM: Document Object Model. W3C recommendation (April 2004) Available at <http://www.w3.org/DOM/DOMTR>
17. Calcagno, C., O'Hearn, P., Yang, H.: Local action and abstract separation logic. LICS 2007 (to appear)

Numerical Constraints for XML^{*}

Sven Hartmann and Sebastian Link

Information Science Research Centre, Massey University, New Zealand
{s.hartmann, s.link}@massey.ac.nz

Abstract. We introduce numerical constraints into the context of XML which restrict the number of nodes within subtrees of an XML tree that contain specific value-equal subnodes. We demonstrate the applicability of numerical constraints by optimising XML queries and predicting the number of XML query answers, updates and encryptions.

In order to effectively unlock the wide range of XML applications decision problems associated with numerical constraints are investigated. The implication problem is *coNP*-hard for several restricted classes of numerical constraints. These sources of intractability direct our attention towards numerical keys that permit the specification of upper bounds. Keys, as introduced by Buneman et al., are numerical keys with upper bound 1. Numerical keys are finitely satisfiable, finitely axiomatisable, and their implication problem is decidable in quadratic time.

1 Introduction

The eXtensible Markup Language (XML,[6]) provides a high degree of syntactic flexibility and is therefore widely used for data exchange and data integration. On the other hand XML shows significant shortcomings to specify the semantics of its data. Consequently, the study of integrity constraints has been recognised as one of the most important yet challenging areas of XML research [12]. The importance of XML constraints is due to a wide range of applications ranging from schema design, query optimisation, efficient storing and updating, data exchange and integration, to data cleaning [12]. Therefore, several classes of integrity constraints have been defined for XML [3,7,9,16,18,19,26]. The complex structure of XML data makes it challenging to balance the tradeoff between the expressiveness of constraint classes and the efficient solution of their associated decision problems [12].

In this paper we will identify numerical constraints as a highly useful and natural class of XML constraints. In order to make effective use of this class in various XML applications we investigate its satisfiability and implication problem. Numerical constraints are defined independently from any specification such as a DTD [6] or XSD [25], and are based on the representation of XML data as trees. This data model is commonly used by DOM [2], XSL [21], and XML Schema [25]. Figure 1 shows such a representation in which nodes are annotated by their type: *E* for element, *A* for attribute, and *S* for text (PCDATA).

^{*} This research is supported by the Marsden Fund Council from Government funding, administered by the Royal Society of New Zealand.

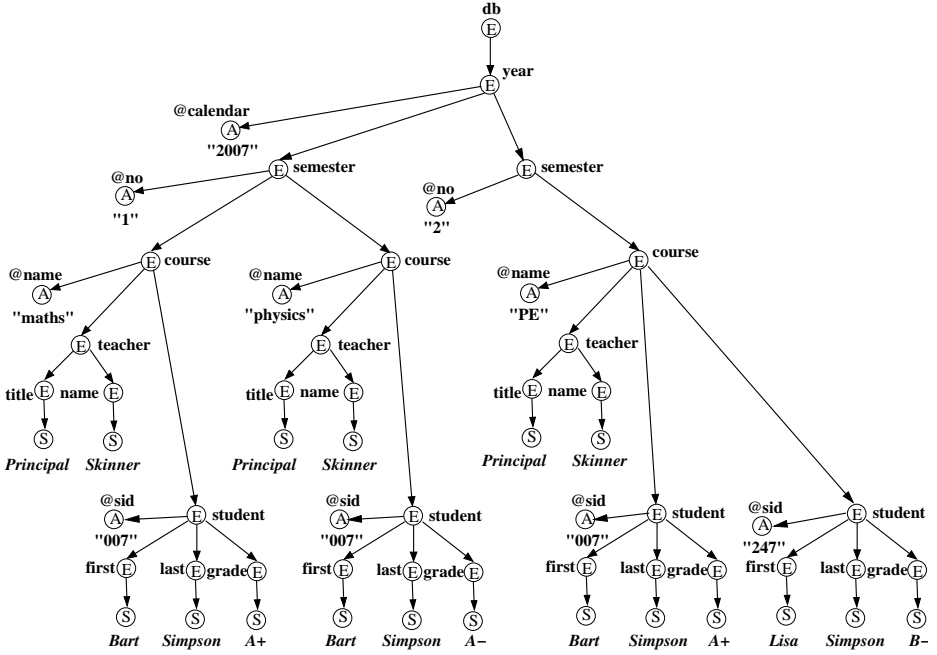


Fig. 1. An XML tree

Numerical constraints are defined in terms of path expressions, and restrict the number of nodes that have the same (complex) values on some selected subnodes. Such restrictions can be specified either for the entire document or relative to selected context nodes. For the XML tree T in Figure 1 the following numerical constraint may be specified for the subtrees rooted at *semester*-nodes: each *student*/*@sid*-value that occurs under some *course*-node actually occurs as a *student*/*@sid*-value under between two and four *course*-nodes in the same *semester*-subtree. In other words, each student who is studying in a certain semester must enrol in at least two and may enrol in up to four courses. Another numerical constraint that may be specified for the subtrees rooted at *year*-nodes is the following: if there are *semester*-children then there must be precisely two. While T violates the first constraint it does satisfy the second.

Students may want to query the database for their grades within a particular year using their cell phone, but may only choose this service in case the costs are reasonable. A database management system capable of reasoning about numerical constraints is able to foresee that the number of answers to the following XQuery [5] query is at most eight:

```

for $s in doc("enrol.xml")/year[@calendar="2007"]//course/student
where $s/@sid="007"
return <grade>{$s/grade}</grade>

```

(1)

Without querying the database itself the student can be informed of the maximal costs that will be charged for this service, and decide accordingly.

Suppose the database designer has specified the additional numerical constraint that in each year a teacher can teach up to three different courses. Applying reasoning techniques an XML query optimiser can transform the XQuery query

```
for $c in doc("enrol.xml")/year[@calendar="2007"]/semester/course
where $c/student/@sid="247" and $c/teacher="Principal Skinner"
return <course>{$c/@name}</course>
```

into the equivalent query

```
for $c in doc("enrol.xml")/year[@calendar="2007"]/semester/course
where $c/teacher="Principal Skinner" and $c/student/@sid="247"
return <course>{$c/@name}</course>
```

which performs better due to the smaller selectivity of course nodes based on teachers rather than students.

The idea of numerical constraints is not new in database practice: they are commonly known as cardinality or participation constraints in conceptual database design, in particular Entity-Relationship modelling [15]. To the best of our knowledge numerical constraints have not yet been considered in XML. This is to some degree surprising since these constraints occur naturally in XML data and are very useful for many XML applications as the examples above already indicate. On the other hand the generalisation of constraints to XML is challenging [12], in particular the identification of classes that can be reasoned about efficiently.

Contributions. We introduce numerical constraints into the context of XML. We believe that numerical constraints are naturally exhibited by XML data and a thorough study of these constraints can significantly improve the semantic capabilities of XML. Moreover, they generalise both the class of XML keys [7,16] and several classes of cardinality constraints studied in the context of Entity-Relationship models [15].

In order to take full advantage of these constraints in various XML applications, it is necessary to reason about them. Similarly to many other classes of XML constraints reasoning about numerical constraints is intractable in general. We show that the finite and unrestricted satisfiability problem are different in the presence of both lower and upper bounds, and locate three different sources of intractability for the finite implication problem of numerical constraints.

We identify *numerical keys* as a large tractable subclass of numerical constraints. They permit the specification of upper bounds on the number of nodes with value-equal subnodes. In particular, numerical keys subsume XML keys [7,16] where the upper bound is always 1.

Unlike many other classes of XML constraints, numerical keys can be reasoned about efficiently. Specifically, any finite set of numerical keys is finitely satisfiable, and the (finite) implication of numerical keys is finitely axiomatisable. Furthermore, numerical key implication is decidable in time quadratic in

the size of the numerical keys given. Therefore, numerical keys are more expressive than XML keys but the time-complexity of their associated implication problems is the same [16].

Finally, we demonstrate the applicability of numerical keys for many XML applications. In the presence of the constraints upper bounds on the number of query answers for XPath [11] and XQuery queries can be predicted. In the same spirit one can obtain upper bounds on the number of updates using the XQuery update facility [10] or on the number of en- and decryptions using XML encryption [20]. Moreover, we outline how the selection of indexes can benefit from their specification.

In summary, we believe that numerical constraints and numerical keys form natural classes of constraints that can be utilised effectively by XML designers and XML applications. Notice that even incomplete sets of sound inference rules for their implication can be of great assistance for various XML applications. In this paper, we shall concentrate on numerical keys as the complexity of their associated decision problems indicates that they can be maintained efficiently by database systems.

Related Work. Constraints have been extensively studied in the context of the relational data model [24]. Dependencies have also been investigated in nested data models, see for instance [14,17]. For a brief survey on recent work on XML constraints see [12] which also includes many more references. To the best of our knowledge numerical constraints, in particular numerical keys, have not been studied for XML before. They extend the class of XML keys [7,16] as well as cardinality constraints studied in the context of Entity-Relationship models [15].

The *minOccurs* and *maxOccurs*-attributes of XML Schema [25] restrict the number of a node's children but do this independently from the data that is actually occurring. In sharp contrast numerical constraints restrict the number of nodes in XML subtrees based on the data that can be found on selected subnodes.

Organisation. We introduce numerical XML constraints in Section 2 based on the common XML tree model. It is shown that reasoning about numerical constraints is computationally intractable in general. Subsequently, we identify numerical keys as a tractable subclass. In Section 3 we provide (i) a finite axiomatisation, and (ii) a quadratic upper time bound for numerical key implication. The applicability of numerical keys to several XML applications is further illustrated in Section 4. Section 5 concludes and briefly looks at future work. Proof techniques and proofs will be given in the full version of the paper.

2 Numerical XML Constraints

In this section we recall the basics of the XML tree model, the notion of value equality, and describe the path language used to locate sets of nodes within an XML document. Subsequently, we introduce numerical constraints for XML and show that reasoning about these constraints is computationally intractable. Finally, we identify numerical keys as an important subclass of numerical constraints.

2.1 The XML Tree Model

XML documents can be modelled as node-labelled trees. We assume that there is a countably infinite set \mathbf{E} denoting element tags, a countably infinite set \mathbf{A} denoting attribute names, and a singleton $\{S\}$ denoting text (PCDATA). We further assume that these sets are pairwise disjoint, and put $\mathcal{L} = \mathbf{E} \cup \mathbf{A} \cup \{S\}$. We refer to the elements of \mathcal{L} as *labels*.

An *XML tree* is a 6-tuple $T = (V, lab, ele, att, val, r)$ where V denotes a set of nodes, and lab is a mapping $V \rightarrow \mathcal{L}$ assigning a label to every node in V . A node $v \in V$ is called an *element node* if $lab(v) \in \mathbf{E}$, an *attribute node* if $lab(v) \in \mathbf{A}$, and a *text node* if $lab(v) = S$. Moreover, ele and att are partial mappings defining the edge relation of T : for any node $v \in V$, if v is an element node, then $ele(v)$ is a list of element and text nodes in V and $att(v)$ is a set of attribute nodes in V . If v is an attribute or text node then $ele(v)$ and $att(v)$ are undefined. val is a partial mapping assigning a string to each attribute and text node: for each node $v \in V$, $val(v)$ is a string if v is an attribute or text node, while $val(v)$ is undefined otherwise. Finally, r is the unique and distinguished root node.

An XML tree is said to be *finite* if V is finite, and is said to be *empty* if V consists of the root node only. For a node $v \in V$, each node w in $ele(v)$ or $att(v)$ is called a *child* of v , and we say that there is an edge (v, w) from v to w in T . An XML tree has a tree structure: for each node $v \in V$, there is a unique (directed) path of edges from the root r to v .

We can now define value equality for pairs of nodes in an XML tree. Informally, two nodes u and v of an XML tree T are value equal if they have the same label and, in addition, either they have the same string value if they are text or attribute nodes, or their children are pairwise value equal if they are element nodes. More formally, two nodes $u, v \in V$ are *value equal*, denoted by $u =_v v$, if and only if the subtrees rooted at u and v are isomorphic by an isomorphism that is the identity on string values. That is, two nodes u and v are value equal when the following conditions are satisfied:

- (a) $lab(u) = lab(v)$,
- (b) if u, v are attribute or text nodes, then $val(u) = val(v)$,
- (c) if u, v are element nodes, then (i) if $att(u) = \{a_1, \dots, a_m\}$, then $att(v) = \{a'_1, \dots, a'_m\}$ and there is a permutation π on $\{1, \dots, m\}$ such that $a_i =_v a'_{\pi(i)}$ for $i = 1, \dots, m$, and (ii) if $ele(u) = [u_1, \dots, u_k]$, then $ele(v) = [v_1, \dots, v_k]$ and $u_i =_v v_i$ for $i = 1, \dots, k$.

For example, all *teacher*-nodes in Figure 1 are value equal.

2.2 Path Languages

In order to define numerical constraints we need a path language that is expressive enough to be practical, yet sufficiently simple to be reasoned about efficiently. This is the case for the path languages PL_s and PL [7,8].

A *path expression* is a (possibly empty) finite list of symbols. In this paper, a *simple path expression* is a path expression that consists of labels from \mathcal{L} . We

Table 1. The path languages PL_s and PL

Path Language	Syntax
PL_s	$P ::= \varepsilon \mid \ell.P$
PL	$Q ::= \varepsilon \mid \ell \mid Q.Q \mid _*$

use the languages PL_s and PL to describe path expressions. Both languages are fragments of regular expressions. PL_s expressions and PL expressions are defined by the grammars in Table 1. Herein, ε denotes the empty path expression, “.” denotes the concatenation of two path expressions, and ℓ denotes any element of \mathcal{L} . The language PL is a generalisation of PL_s that allows the distinguished symbol “ $_*$ ” to occur. We call $_*$ the *don’t care* symbol. It serves as a combination of the wildcard “ $_$ ” and the Kleene star “ $*$ ”. Note that every PL_s expression is a PL expression, too.

We now introduce some more terminology [7,16] used throughout this paper. Note that for PL expressions, the equalities $Q.\varepsilon = \varepsilon.Q = Q$ for all $Q \in PL$, and $_*. _* = _*$ hold. A PL expression Q in *normal form* does not contain consecutive $_*$, and it does not contain ε unless $Q = \varepsilon$. A PL expression can be transformed into normal form in linear time, just by removing superfluous $_*$ and ε symbols. To simplify discussion, we usually assume that PL expressions are already given in normal form. The *length* $|Q|$ of a PL expression Q is the number of labels in Q plus the number of $_*$ in the normal form of Q . The empty path expression ε has length 0.

When replacing all $_*$ in a PL expression Q by simple path expressions, we obtain a simple path expression P and write $P \in Q$. Thus, a PL expression Q gives rise to a regular language of simple path expressions $P \in Q$. In particular, a PL_s expression represents a single simple path expression. For convenience, we will refer to PL_s expressions as simple path expressions, too.

We will use path expressions to describe sets of paths in an XML tree T . Recall that each attribute or text node is a leaf in T . Therefore, a path expression is said to be *valid* if it does not contain a label $\ell \in \mathbf{A}$ or $\ell = S$ in a position other than the last one. In the sequel, we use a valid PL expression to represent only valid simple path expressions. For example, $_*.course$ is a valid PL expression that represents (among others) the valid simple path expression *year.semester.course*.

A path p is a sequence of pairwise distinct nodes v_0, \dots, v_m where (v_{i-1}, v_i) is an edge for $i = 1, \dots, m$. We call p a path from v_0 to v_m , and say that v_m is *reachable* from v_0 following the path p . The path p gives rise to a valid simple path expression $lab(v_1) \dots lab(v_m)$, which we denote by $lab(p)$. Let P be a simple path expression, and Q a PL expression. A path p is called a *P-path* if $lab(p) = P$, and a *Q-path* if $lab(p) \in Q$. If p is a path from v to w , then w is said to be *reachable* from v following a *P-path* or *Q-path*, respectively. We also write $T \models P(v, w)$ and $T \models Q(v, w)$ when w is reachable from v following a *P-path* or *Q-path*, respectively, in an XML tree T . For example, in the XML tree in Figure 1, all *teacher* nodes are reachable from the root following a *year.semester.course.teacher-path*. Consequently, they are also reachable from the root following a $_*.teacher$ -path.

For a node v of an XML tree T , let $v[Q]$ denote the set of nodes in T that are reachable from v by following the PL expression Q , i.e., $v[Q] = \{w \mid T \models Q(v, w)\}$. We shall use $[Q]$ as an abbreviation for $r[Q]$ where r is the root of T . For example, let v be the second *semester* node in Figure 1. Then $v[*.student]$ is the set of all *student* nodes that are children of the second semester. Furthermore, $[*.teacher]$ is the set of all *teacher* nodes in the entire XML tree.

For nodes v and v' of T , the *value intersection* of $v[Q]$ and $v'[Q]$ is given by $v[Q] \cap_v v'[Q] = \{(w, w') \mid w \in v[Q], w' \in v'[Q], w =_v w'\}$ [8]. That is, $v[Q] \cap_v v'[Q]$ consists of all those node pairs in T that are value equal and are reachable from v and v' , respectively, by following Q -paths.

A PL expression Q is said to be *contained* in a PL expression Q' , denoted by $Q \subseteq Q'$, if for any XML tree T and any node v of T we have $v[Q] \subseteq v[Q']$. That is, every node that is reachable from v by following a Q -path is also reachable from v by following a Q' -path. Note that $Q \subseteq Q'$ holds if and only if every valid path expression $P \in Q$ satisfies $P \in Q'$ [8]. The *containment problem* of PL is to decide, given any PL expressions Q and Q' , whether $Q \subseteq Q'$ holds. The containment problem of PL is decidable in $\mathcal{O}(|Q| \times |Q'|)$ time [8].

Note that although we have presented the language PL using the syntax of regular expressions, there is an easy conversion of PL expressions to $XPath$ expressions, just be replacing “ $_*$ ” of a PL expression with “ $/$ ”, and “ $.$ ” with “ $/$ ”. Also, if a PL expression is meant to start from the root, the converted path is preceded with the symbol “ $/$ ”.

The choice of a path language is directly influenced by the complexity of its containment problem. Buneman et al. [7,8] argue that PL is simple yet expressive enough to be adopted by XML designers and maintained by systems for XML applications.

2.3 Numerical Constraints

Let \mathbb{N} denote the positive integers, and $\bar{\mathbb{N}}$ denote the positive integers together with ∞ .

Definition 1. A numerical constraint φ for XML is an expression

$$\text{card}(Q, (Q', \{Q_1, \dots, Q_k\})) = (\min, \max)$$

where Q, Q', Q_1, \dots, Q_k are PL expressions such that $Q.Q'$ is a valid path expression if $k = 0$, and $Q.Q'.Q_i$ are valid path expressions for all $i = 1, \dots, k$ if $k > 0$, where k is a non-negative integer, and where $\min \in \mathbb{N}$ and $\max \in \bar{\mathbb{N}}$ with $\min \leq \max$. Herein, Q is called the context path, Q' is called the target path, Q_1, \dots, Q_k are called key paths, \min is called the lower bound, and \max the upper bound of φ . If $Q = \epsilon$, we call φ absolute; otherwise φ is called relative. \square

If φ denotes a numerical constraint, then Q_φ denotes its context path, Q'_φ its target path, $Q_1^\varphi, \dots, Q_{k_\varphi}^\varphi$ its key paths, \min_φ its lower bound and \max_φ its upper bound. The *size* $|\varphi|$ of a numerical constraint φ is defined as the sum of the lengths of all path expressions in φ , i.e., $|\varphi| = |Q_\varphi| + |Q'_\varphi| + \sum_{i=1}^{k_\varphi} |Q_i^\varphi|$.

Let $\#S$ denote the cardinality of a finite set S , i.e., the number of its elements.

Definition 2. Let $\varphi = \text{card}(Q, (Q', \{Q_1, \dots, Q_k\})) = (\min, \max)$ be a numerical constraint. An XML tree T satisfies φ , denoted by $T \models \varphi$, if and only if for all $q \in \llbracket Q \rrbracket$, for all $q' \in q \llbracket Q' \rrbracket$ such that for all x_1, \dots, x_k with $x_i \in q' \llbracket Q_i \rrbracket$ for $i = 1, \dots, k$, the following holds:

$$\min \leq \#\{q'' \in q \llbracket Q' \rrbracket \mid \exists y_1, \dots, y_k \text{ such that } y_i \in q'' \llbracket Q_i \rrbracket \text{ and } x_i =_v y_i \text{ for } i = 1, \dots, k\} \leq \max. \quad \square$$

To illustrate the definitions above, we formalise the constraints from the introduction. Let T be the XML tree in Figure 1. The constraint

$$\text{card}(_*. \text{semester}, (\text{course}, \{_*. \text{sid}\})) = (2, 4)$$

says that in every semester each student who decides to study must enrol in at least two and at most four courses. T violates this constraint since *Bart Simpson* is only enrolled in one course in the second semester of 2007. However, T does satisfy the second constraint

$$\text{card}(\text{year}, (\text{semester}, \emptyset)) = (2, 2)$$

that states that each *year*-node has either precisely two *semester*-children or none at all. The numerical constraint

$$\text{card}(\text{year}, (_*. \text{course}, \{\text{teacher}\})) = (3, 6)$$

is satisfied while $\text{card}(_*. \text{semester}, (\text{course}, \{\text{teacher}\})) = (3, 3)$ is violated by T since *Principal Skinner* only teaches *maths* and *physics* within semester 1 of 2007. An example of an absolute numerical constraint is

$$\text{card}(_*. \text{course}, \{\text{name}, \text{student.sid}\}) = (1, 3),$$

i.e., a student may attempt to pass the same course up to 3 times. XML keys as introduced by Buneman et al. [7] and further studied in [8,16] are completely covered by numerical constraints. More specifically, the numerical constraint φ is an XML key precisely when $\min_\varphi = \max_\varphi = 1$. Major observations that hold for the definition of XML keys [7,16] therefore also apply to numerical constraints. Some examples of XML keys are $\text{card}(\epsilon, (\text{year}, \{\text{calendar}\})) = (1, 1)$ stating that *year*-nodes can be identified by the value of their *calendar*-child, $\text{card}(\text{year}, (\text{semester}, \{\text{no}\})) = (1, 1)$ stating that *semester*-nodes can be identified by their *no*-child relatively to the *year*, $\text{card}(_*. \text{semester}, (\text{course}, \{\text{name}\})) = (1, 1)$ stating that *course*-nodes can be identified by their *name*-child relatively to the *semester*, and $\text{card}(_*. \text{course}, (\text{student}, \{\text{sid}\})) = (1, 1)$ stating that *student*-nodes can be identified by their *sid*-child relatively to the *course*.

2.4 Satisfiability and Implication

In this section we define fundamental decision problems associated with various classes of constraints. Let Σ be a finite set of constraints in a class \mathcal{C} and T be

an XML tree. We say that T satisfies Σ if and only if $T \models \sigma$ for every $\sigma \in \Sigma$. The (finite) satisfiability problem for \mathcal{C} is to determine, given any finite set Σ of constraints in \mathcal{C} , whether there is a (finite) XML tree satisfying Σ .

Let $\Sigma \cup \{\varphi\}$ be a finite set of constraints in \mathcal{C} . We say that Σ (finitely) implies φ , denoted by $\Sigma \models_{(f)} \varphi$, if and only if every (finite) XML tree T that satisfies Σ also satisfies φ . The (finite) implication problem is to decide, given any finite set of constraints $\Sigma \cup \{\varphi\}$, whether $\Sigma \models_{(f)} \varphi$. If Σ is a finite set of constraints in \mathcal{C} let Σ^* denote its semantic closure, i.e., the set of all constraints implied by Σ . That is, $\Sigma^* = \{\varphi \in \mathcal{C} \mid \Sigma \models \varphi\}$.

2.5 Computational Intractability

For XML keys, the finite and the unrestricted implication problem coincide [8]. For numerical constraints, however, the situation is already different for the satisfiability problem.

Theorem 1. *The satisfiability problem and the finite satisfiability problem for the class of numerical constraints do not coincide.* \square

Still, satisfiability for numerical constraints can be decided easily. This is due to the fact that the empty XML tree satisfies every numerical constraint that contains at least one label from \mathcal{L} . Deciding implication, on the other hand, is not that easy. We now demonstrate that reasoning is likely to be computationally intractable already for very restricted classes of numerical constraints. We call a numerical constraint $\text{card}(P, P', \{P_1, \dots, P_k\}) = (\min, \max)$ simple if P, P', P_1, \dots, P_k are all simple path expressions in PL_s .

Theorem 2. *The finite implication problem for the class of all simple absolute numerical constraints with a non-empty set of key paths is coNP-hard.* \square

The previous result suggests that computational intractability may result from the specification of both lower and upper bounds. The next result indicates that an empty set of key paths may cause computational intractability.

Theorem 3. *The finite implication problem for the class of all simple absolute numerical constraints where the lower bound is fixed to 1 is coNP-hard.* \square

The next theorem suggests another source of computational intractability: the permission to have arbitrary path expressions in both target- and key paths.

Theorem 4. *The finite implication problem for the class of all absolute numerical constraints that have a non-empty set of key paths and where the lower bound is fixed to 1 is coNP-hard.* \square

2.6 Numerical Keys

In order to take advantage of XML applications effectively it becomes necessary to reason about constraints efficiently. The results from the last section motivate the study of a large subclass of numerical constraints that, as we shall show in this paper, turns out to be computationally tractable.

Definition 3. A numerical key for XML is a numerical constraint

$$\text{card}(Q, (Q', \{P_1, \dots, P_k\})) = (1, \max)$$

where k is a positive integer and P_1, \dots, P_k are simple path expressions in PL_s . We will use $\text{card}(Q, (Q', \{P_1, \dots, P_k\})) \leq \max$ to denote numerical keys. Let \mathcal{N} denote the class of all numerical keys. \square

Notice that numerical keys are still much more expressive than XML keys. More specifically, a numerical key φ becomes a key precisely when $\max_\varphi = 1$. The next result states that every finite set of numerical keys can be satisfied by some finite XML tree: we can just choose the empty XML tree. This extends a result for XML keys [8].

Theorem 5. Every finite set of numerical keys is finitely satisfiable. \square

Moreover, the coincidence of finite and unrestricted implication carries over from keys to numerical keys. In what follows we will therefore speak of the implication problem for numerical keys.

Theorem 6. The implication and finite implication problems for numerical keys coincide. \square

3 Numerical Key Implication

The notion of derivability ($\vdash_{\mathfrak{R}}$) with respect to a set \mathfrak{R} of inference rules can be defined analogously to the notion in the relational data model [1, pp. 164-168]. In the first part of this section we will find a set \mathfrak{R} of inference rules which is *sound*, i.e. $\Sigma_{\mathfrak{R}}^+ \subseteq \Sigma^*$, and *complete*, i.e. $\Sigma^* \subseteq \Sigma_{\mathfrak{R}}^+$, for the implication of numerical keys, and where $\Sigma_{\mathfrak{R}}^+ = \{\varphi \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$ denotes the *syntactic closure* of Σ under inference using \mathfrak{R} . In the second part of this section we will provide an upper bound for the time-complexity of deciding numerical key implication.

3.1 Axiomatisation

Table 2 shows a set of inference rules for the implication of numerical keys. The majority of these rules are extensions of the inference rules for keys [8,16]. Moreover, *infinity*, *weakening* and *superkey* rule form an axiomatisation for max-cardinality constraints in the Entity-Relationship model [15].

The *infinity* axiom introduces a numerical key that is satisfied by any XML tree since ∞ does not put any restriction on the tree. The *weakening* rule indicates that a numerical key with upper bound $\max < \infty$ implies *infinitely* many numerical keys with less restrictive upper bounds $\max' > \max$. Therefore, Σ^* is infinite in general.

A significant rule is *multiplication*. The *interaction rule* for keys [8,16] appears as the special case of multiplication where $\max = 1 = \max'$. We will illustrate the multiplication rule by the following example.

Table 2. An axiomatisation for numerical keys

$\frac{}{card(Q, (Q', S)) \leq \infty}$ (infinity)	$\frac{}{card(Q, (\epsilon, S)) \leq 1}$ (epsilon)
$\frac{card(Q, (Q', S)) \leq \max}{card(Q, (Q', S)) \leq \max + 1}$ (weakening)	$\frac{card(Q, (Q', S)) \leq \max}{card(Q, (Q', S \cup \{P\})) \leq \max}$ (superkey)
$\frac{card(Q, (Q'.P, \{P'\})) \leq \max}{card(Q, (Q', \{P.P'\})) \leq \max}$ (subnodes)	$\frac{card(Q, (Q'.Q'', S)) \leq \max}{card(Q.Q', (Q'', S)) \leq \max}$ (context target)
$\frac{card(Q, (Q', S)) \leq \max}{card(Q'', (Q', S)) \leq \max} Q'' \subseteq Q$ (context-path-containment)	$\frac{card(Q, (Q', S)) \leq \max}{card(Q, (Q'', S)) \leq \max} Q'' \subseteq Q'$ (target-path-containment)
$\frac{card(Q, (Q', S \cup \{\epsilon, P\})) \leq \max}{card(Q, (Q', S \cup \{\epsilon, P.P'\})) \leq \max}$ (prefix-epsilon)	$\frac{card(Q, (Q'.P, \{\epsilon, P'\})) \leq \max}{card(Q, (Q', \{\epsilon, P.P'\})) \leq \max}$ (subnodes-epsilon)
$\frac{card(Q, (Q', \{P.P_1, \dots, P.P_k\})) \leq \max, card(Q.Q', (P, \{P_1, \dots, P_k\})) \leq \max'}{card(Q, (Q'.P, \{P_1, \dots, P_k\})) \leq \max \cdot \max'}$ (multiplication)	

Example 1. Imagine a Dodgeball sports league in which teams compete against each other during a season. The numerical key

$$\sigma_1 = card(_*.season, (month, \{match.home, match.away\})) \leq 3$$

indicates that during a season there are at most 3 months in which the same home team and the same away team can both play a match. Furthermore, the same home team can match up against the same away team twice during each month of the season, i.e.,

$$\sigma_2 = card(_*.season.month, (match, \{home, away\})) \leq 2.$$

The soundness of the multiplication rule tells us now that

$$card(_*.season, (month.match, \{home, away\})) \leq 6$$

is implied by these two numerical keys. That is, during every season the same home team can host the same away team up to 6 times. Suppose that the rules of the league suggest that the same home team plays the same away team at most 5 times during a season. This constraint φ , i.e.,

$$card(_*.season, (month.match, \{home, away\})) \leq 5$$

is not implied by σ_1 and σ_2 as the XML tree in Figure 2 shows. Therefore, this numerical key must be specified in addition to σ_1 and σ_2 . \square

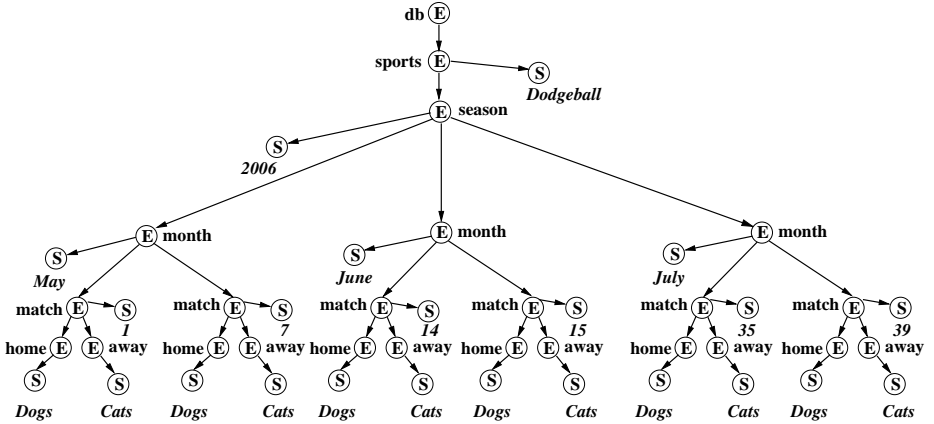


Fig. 2. A counter-example for the implication of φ by $\Sigma = \{\sigma_1, \sigma_2\}$ from Example 1

It is not difficult to show the soundness of the inference rules in Table 2 for the implication of numerical keys. The completeness argument uses shortest path methods and significantly extends our technique applied to XML keys [16].

Theorem 7. *The inference rules from Table 2 are sound and complete for the implication of numerical keys in \mathcal{N} .*

3.2 Time-Complexity of Deciding Implication

The technique of proving completeness can be extended to obtain an algorithm for deciding numerical key implication in time quadratic in the size of the constraints. This significantly generalises the algorithm proposed for deciding key implication [16].

Theorem 8. *Let $\Sigma \cup \{\varphi\}$ be a finite set of numerical keys in \mathcal{N} . The implication problem $\Sigma \models \varphi$ can be decided in time $\mathcal{O}(|\varphi| \cdot (||\Sigma|| + |\varphi|))$.* \square

4 Updating, Encrypting and Indexing

The introduction has already revealed that several XML recommendations would benefit from the specification of numerical keys. In particular, we have seen examples for predicting the number of query answers for XPath and XQuery queries in the presence of numerical keys. In the same way one is able to make predictions for the number of updates, using for instance the XQuery update facility. For example, the XQuery query

```
for $s in doc("enrol.xml")/year[@calendar="2007"]/
  semester[@no="2"]//student[@sid="247"]
return do replace value of $s/last with "Milhouse"
```

will update the lastname of the student with $@sid=247$ by *Milhouse* in all course enrolments of this student in semester two of 2007. If the database management system is able to conclude that $\text{card}(_*.semester, (_*.student, \{sid\})) \leq 4$ is implied by the set Σ of numerical keys specified by the database designer, then the maximal number of updates this query causes is 4.

When XML data is exchanged over the Web it is very common that sensitive information is encrypted, e.g. by XML encryption. In order to evaluate queries on encrypted XML data it may become necessary to decrypt certain data element nodes in order to return the relevant information in the answer. If we recall the example of XQuery query 1 from the introduction and assume that grade elements have been encrypted, then a database management system capable of inferring that both $\text{card}(\text{year}, (_*.student, \{sid\})) \leq 8$ and $\text{card}(_*.student, (\text{grade}, \emptyset)) \leq 1$ are implied by the constraints specified can also predict that the number of necessary decryptions to answer this query is at most 8, and thus also predict the time necessary to deliver the information requested.

As a final application we look at indices that are commonly used to accelerate specific queries. The selection of indexes is an important task when tuning a database. Although already *NP*-complete for relational databases [23] XML queries pose additional challenges to the *index selection problem* since both structure and content need to be covered simultaneously, in general. Suppose the next type of XPath queries are of interest to our application:

`/year[@calendar="2007"]//course[teacher="Principal Skinner" and
student/@sid="007"]/names`

i.e., course names are selected according to a specific year in which the course is taught, a specific teacher who delivers the course and a specific student enrolled in that course. Such a query would call for a multi-key index where the first index is built on the values on `/year/@calendar`. The problem in this scenario is whether the second index will be built on *teacher*-values or on *student/@sid*-values. Reasoning about the numerical keys specified by the database designer may result in the information that for each year there are at most 100 teachers delivering courses in this year and for each of these teachers there are up to 500 students enrolled in courses this teacher delivers. On the other hand one might be able to infer that for each year there are up to 5000 students enrolled in that year's courses and each of these students may be taught by up to 8 different teachers. In this case the second index should be based on *teacher*-values leaving the *student/@sid*-values as the third index. This example illustrates how the specification of numerical keys and the ability to reason about them can potentially reduce the number of choices for the index selection problem.

5 Conclusion and Future Work

We have introduced the class of numerical constraints for XML capable of expressing many important semantic information of XML data. Moreover, we have illustrated that many XML applications can benefit from the specification of

such constraints. While reasoning about numerical constraints is intractable in general we have identified the large subclass of numerical keys that are finitely satisfiable, finitely axiomatisable, and whose implication problem can be decided in quadratic time. Numerical keys form a very natural class of XML constraints that can be utilised effectively by designers, and the complexity of their associated decision problems indicates that they can be maintained efficiently by database systems for XML applications.

One area that warrants future research is the study of (numerical) keys with respect to more expressive path languages [4,22].

Furthermore, (numerical) keys should be investigated in the presence of a schema specification, such as a DTD or an XSD. As already observed for keys [8] numerical keys may behave completely differently under such specifications.

We would like to extend numerical dependencies from relational databases to XML documents. However, since the implication problem for relational numerical dependencies is computationally infeasible [13], future work will focus on identifying useful restricted classes of such XML constraints.

As already indicated one should further explore the impact of numerical constraints on various XML applications, in particular on query optimisation, query rewriting, numbering schemes and indexing techniques. These applications can already benefit greatly from incomplete sets of sound inference rules for the implication of numerical constraints. While reasoning may be intractable in general one may develop efficient decision algorithms for those subclasses syntactically defined by some set of sound inference rules.

References

1. Abiteboul, S., Hull, R., Vianu, V.: *Foundations of Databases*. Addison-Wesley, Reading (1995)
2. Apparao, V., et al.: Document object model (DOM) Level 1 Specification, W3C Recommendation (October 1998), <http://www.w3.org/TR/REC-DOM-Level-1/>
3. Arenas, M., Libkin, L.: A normal form for XML documents. *TODS* 29(1), 195–232 (2004)
4. Benedikt, M., Fan, W., Kuper, G.: Structural properties of XPath fragments. *TCS* 336(1), 3–31 (2005)
5. Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML query language, W3C Recommendation (January 2007), <http://www.w3.org/TR/xquery/>
6. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (XML) 1.0 (Fourth Edition) W3C Recommendation (August 2006), <http://www.w3.org/TR/xml/>
7. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Keys for XML. *Computer Networks* 39(5), 473–487 (2002)
8. Buneman, P., Davidson, S., Fan, W., Hara, C., Tan, W.: Reasoning about keys for XML. *Inf. Syst.* 28(8), 1037–1063 (2003)
9. Buneman, P., Fan, W., Siméon, J., Weinstein, S.: Constraints for semi-structured data and XML. *SIGMOD Record* 30(1), 47–54 (2001)
10. Chamberlin, D., Florescu, D., Robie, J.: XQuery Update Facility, W3C Working Draft, (July 2006), <http://www.w3.org/TR/xqupdate/>

11. Clark, J., DeRose, S.: XML path language (XPath) Version 1.0, W3C Recommendation (November 1999), <http://www.w3.org/TR/xpath>
12. Fan, W.: XML constraints. In: DEXA Workshops, pp. 805–809 (2005)
13. Grant, J., Minker, J.: Inferences for numerical dependencies. TCS 41, 271–287 (1985)
14. Hara, C., Davidson, S.: Reasoning about nested functional dependencies. In: PODS, pp. 91–100 (1999)
15. Hartmann, S.: On the implication problem for cardinality constraints and functional dependencies. Ann. Math. Art. Intell. 33, 253–307 (2001)
16. Hartmann, S., Link, S.: Unlocking keys for XML trees. In: ICDT 2007. LNCS, vol. 4353, pp. 104–118. Springer, Heidelberg (2007)
17. Hartmann, S., Link, S., Schewe, K.-D.: Functional and multivalued dependencies in nested databases generated by record and list constructor. Ann. Math. Art. Intell. 46, 114–164 (2006)
18. Hartmann, S., Link, S., Trinh, T.: Efficient reasoning about XFDs with pre-image semantics. In: DASFAA. LNCS, vol. 4443, pp. 1070–1074. Springer, Heidelberg (2007)
19. Hartmann, S., Trinh, T.: Axiomatising functional dependencies for XML with frequencies. In: Dix, J., Hegner, S.J. (eds.) FoIKS 2006. LNCS, vol. 3861, pp. 159–178. Springer, Heidelberg (2006)
20. Imamura, T., Dillaway, B., Simon, E.: XML encryption syntax and processing, W3C Recommendation (December 2002), <http://www.w3.org/TR/xmlenc-core/>
21. Kay, M.: XSL transformations (XSLT) Version 2.0 W3C Recommendation (January 2007), <http://www.w3.org/TR/xslt20/>
22. Miklau, G., Suciu, D.: Containment and equivalence for a fragment of XPath. J. ACM 51(1), 2–45 (2004)
23. Piatetsky-Shapiro, G.: The optimal selection of secondary indices is NP-complete. SIGMOD Record 13(2), 72–75 (1983)
24. Thalheim, B.: Dependencies in Relational Databases. Teubner (1991)
25. Thompson, H., Beech, D., Maloney, M., Mendelsohn, N.: XML Schema Part 1: Structures second edition, W3C Recommendation (October 2004), <http://www.w3.org/TR/xmlschema-1/>
26. Vincent, M., Liu, J., Liu, C.: Strong functional dependencies and their application to normal forms in XML. TODS 29(3), 445–462 (2004)

Modules over Monads and Linearity

André Hirschowitz¹ and Marco Maggesi²

¹ LJAD, Université de Nice Sophia-Antipolis, CNRS

<http://math.unice.fr/~ah>

² Università degli Studi di Firenze

<http://www.math.unifi.it/~maggesi>

Abstract. Inspired by the classical theory of modules over a monoid, we give a first account of the natural notion of module over a monad. The associated notion of morphism of left modules ("linear" natural transformations) captures an important property of compatibility with substitution, in the heterogeneous case where "terms" and variables therein could be of different types as well as in the homogeneous case. In this paper, we present basic constructions of modules and we show examples concerning in particular abstract syntax and lambda-calculus.

1 Introduction

Substitution is a major operation. Its relevance to computer sciences has been stressed constantly (see e.g. [7]). Mathematicians of the last century have coined two strongly related notions which capture the formal properties of this operation. The first one is the notion of monad, while the second one is the notion of operad. We focus on the notion of monad. A monad in the category \mathbf{C} is a monoid in the category of endofunctors of \mathbf{C} (see 2 below) and as such, has right and left modules. Apriori these are endofunctors (in the same category) equipped with an action of the monad. In fact, we introduce a slightly more general notion of modules over a monad, based on the elementary observation that we can readily extend the notion of a right action of a monad in \mathbf{C} to the case of a functor from any category \mathbf{B} to \mathbf{C} , and symmetrically the notion of a left action of a monad in \mathbf{C} to the case of a functor from \mathbf{C} to any category \mathbf{D} . We are mostly interested in left modules. As usual, the interest of the notion of left module is that it generates a companion notion of morphism. We call morphisms those natural transformations among (left) modules which are compatible with the structure, namely which commute to substitution (we also call these morphisms *linear* natural transformations).

Despite the natural ideas involved, the only mention of modules over monads we have been able to find is on a blog by Urs Schreiber.¹ On the other hand, modules over operads have been introduced by M. Markl ([16,17]) and are commonly used by topologists (see e.g. [9,14,4]). In [8], such modules over operads have been considered, under the name of actions, in the context of semantics.

¹ <http://golem.ph.utexas.edu/string/archives/000715.html>

We think that the notions of module over a monad and linear transformations deserve more attention and propose here a first reference for basic properties of categories of left modules, together with basic examples of morphisms of left modules, hopefully showing the adequacy of the language of left modules for questions concerning in particular (possibly higher-order) syntax and lambda-calculus.

In section 2, we briefly review the theory of monads and their algebras. In section 3, we develop the basic theory of modules. In section 4, we sketch a treatment of syntax (with variable-binding) based on modules. In the remaining sections, we show various linear transformations concerning lists and the lambda-calculus (typed or untyped). The appendix discusses the formal proof in the Coq proof assistant of one of our examples.

2 Monads and Algebras

We briefly recall some standard material about monads and algebras. Experienced readers may want to skip this section or just use it as reference for our notations. Mac Lane's book [15] can be used as reference on this material.

Let \mathbf{C} be a category. A monad over \mathbf{C} is a monoid in the category $\mathbf{C} \rightarrow \mathbf{C}$ of endofunctors of \mathbf{C} . In more concrete terms:

Definition 1 (Monad). A monad $R = \langle R, \mu, \eta \rangle$ is given by a functor $R: \mathbf{C} \rightarrow \mathbf{C}$, and two natural transformations $\mu: R^2 \rightarrow R$ such that the following diagrams commute:

$$\begin{array}{ccc} R^3 & \xrightarrow{R\mu} & R^2 \\ \mu R \downarrow & & \downarrow \mu \\ R^2 & \xrightarrow{\mu} & R \end{array} \qquad \begin{array}{ccccc} I \cdot R & \xrightarrow{\eta R} & R^2 & \xleftarrow{R\eta} & R \cdot I \\ & \searrow 1_R & \downarrow \mu & \swarrow 1_R & \\ & & R & & \end{array}$$

The μ and η natural transformations are often referred as product (or composition) and unit of the monad M . In the programming language Haskell, they are noted `join` and `return` respectively.

Given a monad R and an arrow $f: X \rightarrow RY$, we define the function $\text{bind } f: RX \rightarrow RY$ given by $\text{bind } f := \mu \cdot Rf$. The functoriality and the composition of the monad can be defined alternatively in terms of the unit and the bind operator. More precisely, we have the equations

$$\mu_X = \text{bind } 1_X, \quad Rf = \text{bind}(\eta \cdot f).$$

Moreover, we have the following associativity and unity equations for `bind`

$$\text{bind } g \cdot \text{bind } f = \text{bind}(\text{bind } g \cdot f), \quad \text{bind } \eta_X = 1_{RX}, \quad \text{bind } f \cdot \eta = f \quad (1)$$

for any pair of arrows $f: X \rightarrow RY$ and $g: Y \rightarrow RZ$.

In fact, to give a monad is equivalent to give two operators unit and bind as above which satisfy equations 1.

Example 1 (Lists). To construct the monad of lists L (over \mathbf{Set}), first take the functor $L: \mathbf{Set} \rightarrow \mathbf{Set}$

$$L: X \mapsto \Sigma_{n \in \mathbb{N}} X^n = * + X + X \times X + X \times X \times X + \dots$$

So $L(X)$ is the set of all finite lists with elements in X . Then consider as composition the natural transformation $\mu: L \cdot L \rightarrow L$ given by the *join* (or *flattening*) of lists of lists:

$$\mu[[a_1, \dots], [b_1, \dots], \dots, [z_1, \dots]] = [a_1, \dots, b_1, \dots, \dots, z_1, \dots].$$

The unit $\eta: I \rightarrow L$ is constituted by the singleton map $\eta_X: x \in X \mapsto [x] \in L(X)$.

Example 2 (Lambda Calculus). This example will be worked out with the necessary details in section 5.1, but let us give early some basic ideas (see also [1]). We denote by $FV(M)$ the set of free variables of a λ -term M . For a fixed set X , consider the collection of λ -terms (modulo α -conversion) with free variables in X :

$$\mathbf{LC}(X) := \{M \mid FV(M) \subset X\}.$$

Given a set X we take as unit morphism $\eta_X: X \rightarrow \mathbf{LC}(X)$ the application assigning to an element $x \in X$ the corresponding variable in $\mathbf{LC}(X)$. Every map $f: X \rightarrow Y$ induces a morphism $\mathbf{LC}(f): \mathbf{LC}(X) \rightarrow \mathbf{LC}(Y)$ (“renaming”) which makes \mathbf{LC} a functor. The instantiation (or substitution) of free variables gives us a natural transformation

$$\mu_X: \mathbf{LC}(\mathbf{LC}(X)) \rightarrow \mathbf{LC}(X).$$

With this structure \mathbf{LC} is a monad.

Moreover, by taking the quotient $\Lambda(X)$ of $\mathbf{LC}(X)$ modulo $\beta\eta$ -conversion we still obtain a monad (i.e., the composition and the unit of the monad are compatible with $\beta\eta$ -conversions).

Definition 2 (Maybe monad). *In a category \mathbf{C} with finite sums and a final object (like \mathbf{Set}), the functor $X \mapsto X + *$ which takes an object and “adds one point” has a natural structure of monad on \mathbf{C} . Borrowing from the terminology of the library of the programming language Haskell, we call it the **Maybe monad**.*

Definition 3 (Derivative). *We define the derivative F' of a functor $F: \mathbf{C} \rightarrow \mathbf{C}$ to be the functor $F' = F \cdot \mathbf{Maybe}$. We can iterate the construction and denote by $F^{(n)}$ the n -th derivative.²*

Definition 4 (Morphisms of monads). *A morphism of monads is a natural transformation between two monads $\phi: P \rightarrow R$ which respects composition and unit, i.e., such that the following diagrams commute:*

$$\begin{array}{ccc} P^2 & \xrightarrow{\phi\phi} & R^2 \\ \downarrow \mu_R & & \downarrow \mu_R \\ P & \xrightarrow{\phi} & R \end{array} \qquad \begin{array}{ccc} P & \xrightarrow{\phi} & R \\ \eta_P \swarrow & & \nearrow \eta_R \\ & I & \end{array}$$

² This corresponds to the **MaybeT** monad transformer in Haskell.

It can be easily seen that morphisms of monads form a category.

For our purpose it is relevant to observe that there are a number of natural transformations which arise in the above examples which fail to be morphisms of monads. We take the following as paradigmatic example.

Example 3 (Abstraction is not a morphism of monads). Abstraction on λ -terms gives a natural transformation $\text{abs}: \text{LC}' \rightarrow \text{LC}$ which takes a λ -term $M \in \text{LC}(X+*)$ and binds the “variable” $*$. This fails to be a morphism of monads because it does not respect substitution in the sense of monads: a careful inspection reveals that the transformation

$$\text{LC}(\text{LC}(X + *) + *) \xrightarrow{\mu} \text{LC}(X + *) \xrightarrow{\text{abs}} \text{LC}(X)$$

binds all stars under a single abstraction while

$$\text{LC}(\text{LC}(X + *) + *) \xrightarrow{\text{abs abs}} \text{LC}(\text{LC}(X)) \xrightarrow{\mu} \text{LC}(X)$$

not. In fact, we will see later that LC' is a left module over LC and abs is a LC -linear morphism.

Now let R be a monad over \mathcal{C} .

Definition 5 (Algebra). *An algebra over R is given by an object A and a morphism $\rho: R(A) \rightarrow A$ in \mathcal{C} such that the following diagrams commute:*

$$\begin{array}{ccc} R^2(A) & \xrightarrow{R\rho} & R(A) \\ \mu_A \downarrow & & \downarrow \rho \\ R(A) & \xrightarrow{\rho} & A \end{array} \quad \begin{array}{ccc} A & \xrightarrow{\eta_A} & R(A) \\ & \searrow 1_A & \downarrow \rho \\ & & A \end{array}$$

Definition 6. *Let A, B be two algebras over a monad R . An arrow $f: A \rightarrow B$ in \mathcal{C} is said to be a morphism of algebras if it is compatible with the associated actions, i.e., the two induced morphisms from $R(A)$ to B are equal:*

$$\rho_B \cdot R(f) = f \cdot \rho_A$$

As we will see later, algebras can be regarded as special kind of right modules.

Example 4 (Monoids). In the category of sets, algebras over the monad L of lists are sets equipped with a structure of monoid; given a monoid A , the corresponding action $L(A) \rightarrow A$ is the product (sending a list to the corresponding product).

3 Modules over Monads

Being a monoid in a suitable monoidal category, a monad has associated left and right modules which, a-priori, are objects in the same category, acted upon by the monoid.

Although we are mostly interested in left modules, let us remark that from this classical point of view, algebras over a monad are not (right-)modules. We give a slightly more general definition of modules which is still completely natural. According to this extended definition, algebras turn out to be right-modules.

3.1 Left Modules

We start first by concentrating ourselves on left modules over a given monad R over a category \mathbf{C} .

Definition 7 (Left modules). *A left R -module in \mathbf{D} is given by a functor $M: \mathbf{C} \rightarrow \mathbf{D}$ equipped with a natural transformation $\rho: M \cdot R \rightarrow M$, called action, which is compatible with the monad composition, more precisely, we require that the following diagrams commute*

$$\begin{array}{ccc} M \cdot R^2 & \xrightarrow{M\mu} & M \cdot R \\ \rho R \downarrow & & \downarrow \rho \\ M \cdot R & \xrightarrow{\rho} & M \end{array} \quad \begin{array}{ccc} M \cdot R & \xleftarrow{M\eta} & M \cdot I \\ \downarrow \rho & \swarrow 1_M & \\ M & & \end{array}$$

We will refer to the category \mathbf{D} as the range of M .

Remark 1. The companion definition of modules over an operad (c.f. e.g. [17,9]) follows easily from the observation [19] that operads are monoids in a suitable monoidal category. This monoidal structure is central in [6].

Given a left R -module M , we can introduce the `mbind` operator which, to each arrow $f: X \rightarrow RY$, associates an arrow `mbind` $f := MX \rightarrow MY$ defined by `mbind` $f := \rho \cdot Mf$. The axioms of left module are equivalent to the following equations over `mbind`:

$$\text{mbind } g \cdot \text{mbind } f = \text{mbind}(\text{bind } g \cdot f), \quad \text{mbind } \eta_X = 1_X$$

Example 5. We can see our monad R as a left module over itself (with range \mathbf{C}), which we call the *tautological* module.

Example 6. Any constant functor $\underline{W}: \mathbf{C} \rightarrow \mathbf{D}$, $W \in \mathbf{D}$ is a trivial left R -module.

Example 7. For any functor $F: \mathbf{D} \rightarrow \mathbf{E}$ and any left R -module $M: \mathbf{C} \rightarrow \mathbf{D}$, the composition $F \cdot M$ is a left R -module (in the evident way).

Definition 8 (Derived module). *As for functors and monads, derivation is well-behaved also on left modules: for any left R -module M , the derivative $M' = M \cdot \text{Maybe}$ has a natural structure of left R -module where the action $M' \cdot P \rightarrow M'$ is the composition*

$$M \cdot \text{Maybe} \cdot R \xrightarrow{M\gamma} M \cdot R \cdot \text{Maybe} \xrightarrow{\rho \text{Maybe}} M \cdot \text{Maybe}$$

and γ is the natural arrow $\text{Maybe} \cdot R \rightarrow R \cdot \text{Maybe}$.

Definition 9 (Morphisms of left modules). *We say that a natural transformation of left R -modules $\tau: M \rightarrow N$ is linear if it is compatible with substitution:*

$$\begin{array}{ccc} M \cdot R & \xrightarrow{\tau R} & N \cdot R \\ \rho_M \downarrow & & \downarrow \rho_N \\ M & \xrightarrow{\tau} & N \end{array}$$

We take linear natural transformations as left module morphisms.

Remark 2. Here the term *linear* refers to linear algebra: linear applications between modules over a ring are group morphisms compatible with the action of the ring. It is compatible with the usual flavor of the word linear (no duplication, no junk) as the following example shows.

Example 8. We consider the monad M on **Set** generated by two binary constructions $+$ and $*$ and we build (by recursion) a natural transformation $n: M \rightarrow M$ as follows: for a variable x , $n(x)$ is $x + x$, while for the other two cases we have $n(a + b) = n(a) * n(b)$ and $n(a * b) = n(a) + n(b)$. It is easily verified that n is a non-linear natural transformation (check the diagram against $n(\text{var}(x * x))$).

Example 9. We easily check that the natural transformation of a left module into its derivative is linear. Note that there are two natural inclusions of the derivative M' into the second derivative M'' . Both are linear.

Example 10. Consider again the monad of lists L . The concatenation of two lists is a L -linear morphism $L \times L \rightarrow L$.

Definition 10 (Category of left modules). We check easily that linear morphisms between left R -modules with the same range yield a subcategory of the functor category. We denote by $\text{Mod}^D(R)$ the category of left R -modules with range D .

Definition 11 (Product of left modules). We check easily that the cartesian product of two left R -modules as functors (having as range a cartesian category D) is naturally a left R -module again and is the cartesian product also in the category $\text{Mod}^D(R)$. We also have finite products as usual. The final left module $*$ is the product of the empty family.

Example 11. Given a monad R on **Set** and a left R -module M with range in a fixed cartesian category D , we have a natural “evaluation” morphism $\text{eval}: M' \times R \rightarrow M$, where M' is the derivative of M .

Proposition 1. Derivation yields a cartesian endofunctor on the category of left R -modules with range in a fixed cartesian category D

3.2 Right Modules

Let R be a monad over a category C . The definition of right module is similar to that of left module.

Definition 12 (Right modules). A right R -module (from D) is given by a functor $M: D \rightarrow C$ equipped with a natural transformation $\rho: R \cdot M \rightarrow M$ which makes the following diagrams commutative

$$\begin{array}{ccc}
 R^2 \cdot M & \xrightarrow{\mu^M} & R \cdot M \\
 R\rho \downarrow & & \downarrow \rho \\
 R \cdot M & \xrightarrow{\rho} & M
 \end{array}
 \qquad
 \begin{array}{ccc}
 I \cdot M & \xrightarrow{\eta^M} & R \cdot M \\
 1_M \searrow & & \downarrow \rho \\
 & & M
 \end{array}$$

As for left modules, we will call corange of M the category D .

We remark that for any right R -module M and any object $X \in \mathbf{D}$ the image $M(X)$ is an R -algebra. Then a right R -module is simply a functor from the corange category \mathbf{D} to the category of R -algebras.

Example 12. Our monad R is a right module over itself.

Example 13. If A is an R -algebra, then for any category \mathbf{D} the constant functor $\underline{A}: X \mapsto A$ has a natural structure of right R -module. In particular, we can identify our algebra with the corresponding functor $\underline{A}: * \rightarrow \mathbf{C}$, where $*$ is the category with one object and one arrow.

Example 14. Let $\phi: R \rightarrow P$ be a morphism of monads. Then P is a right and left R -module with actions given respectively by $P \cdot R \xrightarrow{\phi_R} R \cdot R \xrightarrow{\mu_R} R$ and $R \cdot P \xrightarrow{P\phi} R \cdot R \xrightarrow{\mu_R} R$.

Definition 13 (Morphisms of right R -modules). A morphism of right R -modules is a natural transformation $\tau: M \rightarrow N$ which is compatible with substitution, i.e., such that the following diagram commutes:

$$\begin{array}{ccc} R \cdot M & \xrightarrow{R\tau} & R \cdot N \\ \rho_M \downarrow & & \downarrow \rho_N \\ M & \xrightarrow{\tau} & N \end{array}$$

Definition 14 (Category of right R -modules). We check easily that module morphisms among right R -modules with the same corange yield a subcategory of the functor category.

3.3 Limits and Colimits of Left Modules

Limits and colimits in the category of left modules can be constructed pointwise. For instance:

Lemma 1 (Limits and colimits of left modules). If \mathbf{D} is complete (resp. cocomplete), then $\text{Mod}^{\mathbf{D}}(R)$ is complete (resp. cocomplete).

Proof. Suppose first that \mathbf{D} be a complete category and $G: \mathbf{I} \rightarrow \text{Mod}^{\mathbf{D}}(R)$ be a diagram of left modules over the index category \mathbf{I} . For any object $X \in \mathbf{C}$ we have a diagram $G(X): \mathbf{I} \rightarrow \mathbf{D}$ and for any arrow $f: X \rightarrow Y$ in \mathbf{C} we have a morphism of diagrams $G(X) \rightarrow G(Y)$. So define

$$U(X) := \lim G(X)$$

Next, given an arrow $f: X \rightarrow Y$, we have an induced morphism of diagrams $G(X) \rightarrow G(Y)$ by the module structure on each object of the diagram. This induces a morphism $\text{mbind } f: U(X) \rightarrow U(Y)$. It is not hard to prove that mbind satisfies the module axioms and that U is the limit of G . The colimit construction is carried analogously.

3.4 Base Change

Definition 15 (Base change). *Given a morphism $f: A \rightarrow B$ of monads and a left B -module M , we have an A -action on M given by*

$$M \cdot A \xrightarrow{Mf} M \cdot B \xrightarrow{\rho_M} M.$$

We denote by f^*M the resulting A -module and we refer to f^* as the base change operator.

Lemma 2. *The base change of a left module is a left module.*

Proof. Our thesis is the commutativity of the diagram

$$\begin{array}{ccccc}
 M \cdot B \cdot A & \xleftarrow{MfA} & M \cdot A \cdot A & \xrightarrow{M\mu} & M \cdot A \\
 \downarrow \rho A & & \downarrow Mff & & \downarrow Mf \\
 & & M \cdot B \cdot B & \xrightarrow{M\mu} & M \cdot B \\
 & & \downarrow \rho B & & \downarrow \rho \\
 M \cdot A & \xrightarrow{Mf} & M \cdot B & \xrightarrow{\rho} & M
 \end{array}$$

which follows from the commutativity of the three pieces: M is a left B -module, the map from $M(B(_)) \rightarrow M(_)$ is functorial, and f is a morphism.

Definition 16 (Base change (functoriality)). *We upgrade the base change operator into a functor $f^*: \text{Mod}^D(B) \rightarrow \text{Mod}^D(A)$ by checking that if $g: M \rightarrow N$ is a morphism of left B -modules, then so is the natural transformation $f^*g: f^*M \rightarrow f^*N$.*

Proposition 2. *The base change functor commutes with products and with derivation.*

Proposition 3. *Any morphism of monads $f: A \rightarrow B$ yields a morphism of left A -modules, still denoted f , from A to f^*B .*

4 Initial Algebra Semantics

To ease the reading of the forthcoming sections, we collect in this section some classical ideas about Initial Algebra Semantics.

Given a category \mathbb{C} and an endofunctor $T: \mathbb{C} \rightarrow \mathbb{C}$, a T -algebra³ is given by an object $A \in \mathbb{C}$ and an arrow

$$\sigma_A: TA \rightarrow A.$$

³ There is a lexical conflict here with algebra of monads introduced in section 2, which is deeply rooted in the literature anyway. We hope that this will not lead to any confusion.

A morphism of T -algebras is an arrow $f : A \rightarrow B$ which commutes with the structural morphism σ

$$\begin{array}{ccc} TA & \xrightarrow{Tf} & TB \\ \sigma_A \downarrow & & \downarrow \sigma_B \\ A & \xrightarrow{f} & B \end{array}$$

This defines the category of T -algebras. Notice that, for any T -algebra A , there is an induced T -algebra structure on TA given by $T\sigma_A : T(TA) \rightarrow TA$, turning σ_A into a morphism of algebras. An initial T -algebra is called a (*least*) *fixpoint* of T . Given one such fixpoint U and any other T -algebra A we denote by $\text{fold}_A : U \rightarrow A$ the induced initial morphism. We observe that σ_U is an isomorphism whose inverse is fold_{TU} since $\sigma_U \cdot \text{fold}_{TU} = 1_U$ by the universal property of U and from the naturality of fold follows that the diagram

$$\begin{array}{ccc} TU & \xrightarrow{T \text{ fold}_{TU}} & T(TU) \\ \sigma_U \downarrow & \searrow 1_{TU} & \downarrow T\sigma_U \\ U & \xrightarrow{\text{fold}_{TU}} & TU \end{array}$$

is commutative.

Let us show how this general framework can work in the case of (polymorphic) lists.

Example 15. Take $\mathbf{C} = \mathbf{Set} \rightarrow \mathbf{Set}$ the category of endofunctors of \mathbf{Set} and consider the functor $T : (\mathbf{Set} \rightarrow \mathbf{Set}) \rightarrow (\mathbf{Set} \rightarrow \mathbf{Set})$ defined by

$$T(F) := X \mapsto * + X \times FX.$$

The least fix point of T is (the underlying functor of) the monad of lists L introduced in section 2. The T -algebra structure $* + X \times LX = TL \simeq L$ gives the constructors (`nil`, `cons`) and the corresponding destructors. We would like to recognise this structural isomorphism as an L -linear morphism. Unfortunately, we do not have on TL a structure of left L -module corresponding to our expectation (notice that the identity functor is not an L -module in a natural way). We will explain in section 6 how this phenomenon can be considered a consequence of the lack of typing.

5 Monads over Sets

In this section we consider more examples of linear morphisms over monads on the category of sets.

5.1 Untyped Syntactic Lambda Calculus

Consider the functor $T := (\text{Set} \rightarrow \text{Set}) \rightarrow (\text{Set} \rightarrow \text{Set})$ given by

$$TF: X \mapsto X + FX \times FX + F'X$$

where F' denotes the derived functor $X \mapsto F(X + *)$. It can be shown that T possesses a least fixpoint that we denote by LC (LC standing for λ -calculus, cfr. the example in section 2). We consider $\text{LC}(X)$ as the set of λ -terms with free variables taken from X (see also [3]). In fact, the structural morphism $T\text{LC} \rightarrow \text{LC}$ gives the familiar constructors for λ -calculus in the *locally nameless* encoding, namely, the natural transformations

$$\text{var}: I \rightarrow \text{LC}, \quad \text{app}: \text{LC} \times \text{LC} \rightarrow \text{LC}, \quad \text{abs}: \text{LC}' \rightarrow \text{LC}.$$

As already observed, the substitution (instantiation) of free variables gives a monad structure on LC where var is the unit.

We would like to express that these constructors are well behaved with respect to substitution. Again, as in the case of lists, $T\text{LC}$ has no natural structure of left LC -module. However, we can consider the functor T as built of two parts $TF = I + T_0F$ where $T_0F := F \times F + F'$ (in other words we are tackling apart var , the unit of the monad, from the other two constructors app and abs). Now $T_0\text{LC}$ is a left LC -module and we can observe that the algorithm of substitution is carried precisely in such a way that the induced morphism

$$\text{app}, \text{abs}: T_0\text{LC} \rightarrow \text{LC}$$

is LC -linear or, equivalently, the natural transformations $\text{app}: \text{LC} \times \text{LC} \rightarrow \text{LC}$ and $\text{abs}: \text{LC}' \rightarrow \text{LC}$ are LC -linear. To make the idea clearer, we reproduce a short piece of code in the *Haskell* programming language which implements the algorithm of substitution.

```
module LC where
import Monad (liftM)

data LC a = Var a | App (LC a) (LC a) | Abs (LC (Maybe a))

instance Monad LC where
  return = Var
  Var x >>= f = f x
  App x y >>= f = App (x >>= f) (y >>= f)
  Abs x >>= f = Abs (x 'mbind' f)

mbind :: LC (Maybe a) -> (a -> LC b) -> LC (Maybe b)
mbind x f = x >>= maybe (Var Nothing) (liftM Just . f)
```

In the above code, `mbind` constitutes the left LC -module structure on LC' . It is now evident that the recursive calls in the definition of `(>>=)` are exactly those given by the linearity of `app` and `abs`.

We can go further and try to make the linearity more explicit in the syntactic definition of λ -calculus. This can be done as follows.

Theorem 1. *Consider the category Mon^{T_0} where objects are monads R over sets endowed with a R -linear morphism $T_0R \rightarrow R$ while arrows are given by commutative diagrams*

$$\begin{array}{ccc} T_0R & \xrightarrow{T_0f} & f^*T_0P \\ \downarrow & & \downarrow \\ R & \xrightarrow{f} & f^*P \end{array}$$

where all morphisms are R -linear (we are using implicitly the fact that the base change functor commutes with derivation and products). The monad LC is initial in Mon^{T_0} .

In fact, the previous theorem can be generalized as follows (interested readers may also want to look at other works on higher order abstract syntax, e.g., [6,13,10] see also our [12]). Let R be a monad over Set . We define an *arity* to be a list of nonnegative integers. We denote by \mathbb{N}^* the set of arities. For each arity (a_1, \dots, a_r) , and for any R -module M , we define the R -module $T^a M$ by

$$T^a M = M^{(a_1)} \times \dots \times M^{(a_r)},$$

where $M^{(n)}$ denotes the n -th derivative of M , and we say that a linear morphism $T^a R \rightarrow R$ is a R -representation of a (or a representation of a in R). For instance, the **app** and **abs** constructors are LC -representations of the arities $(0, 0)$ and (1) respectively.

Next, we consider *signatures* which are family of arities. For each signature $\Sigma: I \rightarrow \mathbb{N}^*$, and for any R -module M , we define the R -module $T^\Sigma M$ by

$$T^\Sigma M = \sum_{i \in I} T^{\Sigma_i} M$$

and we say that a linear morphism $T^\Sigma R \rightarrow R$ is a R -representation of Σ (or a representation of Σ in R). Altogether **app** and **abs** give a LC -representation of the signature $((0, 0), (1))$.

As in the special case of the λ -calculus, representations of a given signature Σ form a category.

Theorem 2. *For any signature Σ , the category of Σ -representations has an initial object.*

5.2 Untyped Semantic Lambda Calculus

For any set X , consider the equivalence relation $\equiv_{\beta\eta}$ on $\text{LC}(X)$ given by the reflexive symmetric transitive closure of β and η conversions and define $\Lambda(X) := \text{LC}(X) / \equiv_{\beta\eta}$. It can be shown that $\equiv_{\beta\eta}$ is compatible with the structure of LC so Λ has a structure of monad, the projection $\text{LC} \rightarrow \Lambda$ is a morphism of monads, and we have an induced morphism $T_0\Lambda \rightarrow \Lambda$ which is Λ -linear.

Now the key fact is that the abstraction $\text{abs}: \Lambda' \rightarrow \Lambda$ is a linear isomorphism! In fact, it is easy to construct its inverse $\text{app}_1: \Lambda \rightarrow \Lambda'$:

$$\text{app}_1 x = \text{app}(\hat{x}, *)$$

where $x \mapsto \hat{x}$ denotes the natural inclusion $\Lambda \rightarrow \Lambda'$. The equation

$$\text{abs} \cdot \text{app}_1 = 1_\Lambda$$

clearly corresponds to the η -rule while the other equation

$$\text{app}_1 \cdot \text{abs} = 1_{\Lambda'}$$

can be considered the ultimate formulation of the β -rule. In fact, there is a more classical formulation of the β -rule which can be stated as the commutativity of the diagram

$$\begin{array}{ccc} \Lambda' \times \Lambda & \xrightarrow{\text{abs} \times \Lambda} & \Lambda \times \Lambda \\ & \searrow \text{subst} & \downarrow \text{app} \\ & & \Lambda \end{array}$$

Again, we can present this situation from a more syntactical point of view. For this, consider the category of *exponential monads*: an exponential monad is a monad R endowed with a R -linear isomorphism with its derivative $\text{exp}_R: R' \rightarrow R$. An arrow is a monad morphism f such that

$$\begin{array}{ccc} R' & \xrightarrow{f'} & f^* P' \\ \text{exp}_R \downarrow & & \downarrow \text{exp}_P \\ R & \xrightarrow{f} & f^* P \end{array}$$

is a commutative diagram of R -modules (we are implicitly using the commutativity of base change with derivation).

Theorem 3. *The monad Λ is initial in the category of exponential monads.*

We have developed a formal proof of the above theorem in the Coq proof assistant [5] which is discussed in the appendix.

6 Monads over Types

So far we mostly considered examples of monads and modules on the category $\mathbf{C} = \mathbf{Set}$ of small sets. Other interesting phenomena can be captured by taking into account monads and modules on other categories. In this section we consider the case $\mathbf{C} = \mathbf{Set}_\tau$ the category of sets fibered over a fixed set τ . This is the category given by maps $\phi_X: X \rightarrow \tau$, called τ -sets, where arrows $\langle X, \phi_X \rangle \rightarrow \langle Y, \phi_Y \rangle$ are given by maps $f: X \rightarrow Y$ which commute with the structural morphisms, i.e., $\phi_Y \cdot f = \phi_X$. For each $t \in \tau$ and each τ -set X , we denote by $X_t := \phi_X^{-1}(t)$ the preimage of t in X . We regard τ as a “set of types” and the fibers X_t as a set of “terms of type t ”.

6.1 Typed Lists

Here we show how, in the context of typed lists, the constructors `nil` and `cons` may appear as linear. To this effect, we introduce a distinction between the base type $*$, the type of lists $\text{list } *$, the type of lists of lists, etc. Thus we take $\tau = \mathbb{N}$ the inductive set of types generated by the grammar $\tau = * \mid \text{list } \tau$, and consider the category \mathbf{Set}_τ .

For each $t \in \tau$ we define $\mathcal{L}_t : \mathbf{Set}_\tau \rightarrow \mathbf{Set}$ by setting $\mathcal{L}_t(X)$ to be the set of terms of type t built out from (typed) variables in X by adding, as usual, terms obtained through the `nil` and `cons` constructions. By glueing these \mathcal{L}_t together, we obtain an endofunctor \mathcal{L} in \mathbf{Set}_τ . It is easily seen to be a monad (the present structure of monad has nothing to do with flattening).

For each $t \in \tau$, \mathcal{L}_t is a left \mathcal{L} -module (by Example 7). The `nil` and `cons` constructors constitute a family of natural transformations parametrized by $t \in \tau$

$$\text{nil}_t : * \longrightarrow \mathcal{L}_{\text{list } t}, \quad \text{cons}_t : \mathcal{L}_t \times \mathcal{L}_{\text{list } t} \longrightarrow \mathcal{L}_{\text{list } t}.$$

Hence we have here examples of heterogeneous modules since $*$, \mathcal{L}_t and $\mathcal{L}_{\text{list } t}$ are \mathcal{L} -modules in \mathbf{Set} . And nil_t and cons_t are easily seen to be morphisms among these modules.

We may also want to glue for instance these cons_t into a single `cons`. For this, we need shifts. Given $X \in \mathbf{Set}_\tau$ we have the associated *shifts* $X[n]$ which are obtained by adding n to the structural map $X \rightarrow \mathbb{N}$. The shift $(\cdot)[n] : X \mapsto X[n]$ gives an endofunctor over \mathbf{Set}_τ . Given a functor F from any category to the category \mathbf{Set}_τ , we consider the *shifted functors* $F[n]$ obtained as composition of F followed by $(\cdot)[n]$. From the remarks of section 3, it follows at once that if F is an \mathcal{L} -module, then so is $F[n]$. With these notations, glueing yields

$$\text{nil} : * [1] \longrightarrow \mathcal{L}, \quad \text{cons} : \mathcal{L}[1] \times \mathcal{L} \longrightarrow \mathcal{L}$$

where $*$ denotes the final functor in the category of endofunctors of \mathbf{Set}_τ . Again `nil` and `cons` are easily checked to be \mathcal{L} -linear.

6.2 Simply Typed Lambda Calculus

Our second example of typed monad is the simply-typed λ -calculus. We denote by τ the set of simple types $\tau := * \mid \tau \rightarrow \tau$. Following [22], we consider the syntactic typed λ -calculus as an assignment $V \mapsto \mathbf{LC}_\tau(V)$, where $V = \sum_{t \in \tau} (V_t)$ is a (variable) set (of typed variables) while

$$\mathbf{LC}_\tau(V) = \sum_{t \in \tau} (\mathbf{LC}_\tau(V))_t$$

is the set of typed λ -terms (modulo α -conversion) built on free variables taken in V .

Given a type t we set $\mathbf{LC}_t(X) := (\mathbf{LC}_\tau(X))_t$ which gives a functor over τ -sets, which is equipped with substitution, turning it into a (heterogeneous) left module over \mathbf{LC}_τ . And given two types s, t , we have

$$\text{app}_{s,t} : \mathbf{LC}_{s \rightarrow t} \times \mathbf{LC}_s \longrightarrow \mathbf{LC}_t$$

which is linear.

For the **abs** construction, we need a notion of partial derivative for a module. For a left module M over τ -sets, and a type $t \in \tau$, we set

$$\delta_t M(V) := M(V + *_t)$$

where $V + *_t$ is obtained from V by adding one element with type t . It is easily checked how $\delta_t M$ is again a left module. Now, given two types s and t , it turns out that

$$\mathbf{abs}_{s,t} : \delta_s \mathbf{LC}_t \longrightarrow \mathbf{LC}_{s \rightarrow t}$$

is linear.

As in the untyped case, we can consider the functor Λ_τ obtained by quotienting modulo $\beta\eta$ conversion. This is again a monad over the category of τ -sets and the natural quotient transformation $\mathbf{LC}_\tau \rightarrow \Lambda_\tau$ is a morphism of monads. For this semantic monad, the above left module morphisms induce semantic counterparts: $\mathbf{app}_{s,t} : \Lambda_{s \rightarrow t} \times \Lambda_s \longrightarrow \Lambda_t$ and $\mathbf{abs}_{s,t} : \delta_s \Lambda_t \longrightarrow \Lambda_{s \rightarrow t}$.

Here we need a new notion of arity and signature, which we will introduce in some future work, in order to state and prove a typed counterpart of our theorem 2. For a typed counterpart of our theorem 3, see [22].

6.3 Typed Lambda Calculus

Our final example of typed monad is just a glance to more general typed λ -calculi. The point here is that the set of types is no more fixed. Thus our monads take place in the category **Fam** of set families: an object in **Fam** is an application $p : I \rightarrow \mathbf{Set}$ while a morphism $m : p \rightarrow p'$ is a pair (m_0, m_1) with $m_0 : I \rightarrow I'$, and $m_1 : (i : I)p(i) \rightarrow p'(m_0(i))$. We say that I is the set of types of $p : I \rightarrow \mathbf{Set}$. From **Fam** there are two forgetful functors $T, Tot : \mathbf{Fam} \rightarrow \mathbf{Set}$ where $T(p : I \rightarrow \mathbf{Set}) := I$ and $Tot(p) := \coprod_{i \in T(p)} p(i)$, and a natural transformation $proj : Tot \rightarrow T$, defined by $proj(p) = p$ in the obvious sense. Given a monad R on **Fam**, we thus have a morphism of R -modules $proj_R : Tot \circ R \rightarrow T \circ R$.

We need also two *may-be* monads on **Fam**: the first one $F \mapsto F^*$ adds one (empty) type (*tnew*) to F , while the second one, $F \mapsto F^{*/}$ adds one type (*tnew*) with one element (*new*). Given a monad R on **Fam**, we thus have two “derived” R -modules: $R^* := F \mapsto R(F^*)$ and $R^{*/} := F \mapsto R(F^{*/})$.

Now when should we say that R is a lambda-calculus in this context? At least we should have a module morphism $arrow : (T \circ R)^2 \rightarrow T \circ R$. and a module morphism for abstraction, $abs : R^{*/} \rightarrow R^*$ (the “arity” for application is not so simple). We hope this example shows the need for new notions of arity and signature, as well as the new room opened by modules for such concepts.

7 Monads over Preordered Sets

Our last example is about monads and modules over the category of preordered sets (sets with a reflexive and transitive binary relation). Preordering is used here to model the relation $\xrightarrow{\beta\eta}_*$ generated by the reflexive and transitive closure

of the β and η conversions. In fact, the construction given in this section can be considered a refinement of those of section 5.1 where we used the reflexive, symmetric and transitive closure $\equiv_{\beta\eta}$.

Let us consider again the monad \mathbf{LC} of λ -terms. Given a preordered set X , we consider the preordering on $\mathbf{LC}(X)$ given by the rules

$$\begin{aligned} x \leq y &\implies \mathbf{var} \, x \leq \mathbf{var} \, y, \\ S \leq S' \wedge T \leq T' &\implies \mathbf{app}(S, T) \leq \mathbf{app}(S', T'), \\ T \leq T' &\implies \mathbf{abs}(T) \leq \mathbf{abs}(T'), \\ T \longrightarrow_{\beta\eta} T' &\implies T \leq T'. \end{aligned}$$

It is not hard to verify that with this new structure \mathbf{LC} is now a monad over preordered sets. It turns out that the \mathbf{app} and \mathbf{abs} constructions are still \mathbf{LC} -linear with respect to this richer structure.

8 Conclusions and Related Works

We have introduced the notion of module over a monad, and more importantly the notion of linearity for transformations among such modules and we have tried to show that this notion is ubiquitous as soon as syntax and semantics are concerned. Our thesis is that the point of view of modules opens some new room for initial algebra semantics, as we sketched for typed λ -calculus (see also [12]).

The idea that the notion of monad is suited for modelling substitution concerning syntax (and semantics) has been retained by many recent contributions concerned with syntax (see e.g. [2,11,18]) although some other settings have been considered. Notably in [6] the authors work within a setting roughly based on operads (although they do not write this word down; the definition of operad is on Wikipedia; operads and monads are not too far from each other). As they mention, their approach is, to some extent, equivalent to an approach through monads. It has been both applied e.g. in [21] and generalized e.g. in [20]. Another approach to syntax with bindings, initiated by Gabbay and Pitts [10], relies on a systematic consideration of freshness, an issue which is definitely ignored in the monadic or operadic approach.

While the notion of module over a monad has been essentially ignored till now, the notion of module over an operad has been introduced more than ten years ago, and has been incidentally considered in the context of semantics, as we already mentioned in our introduction.

References

1. Altenkirch, T., Reus, B.: Monadic presentations of lambda terms using generalized inductive types. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 453–468. Springer, Heidelberg (1999)
2. Bird, R., Paterson, R.: Generalised folds for nested datatypes. *Formal Aspects of Computing* 11(2), 200–222 (1999)

3. Bird, R.S., Paterson, R.: De Bruijn notation as a nested datatype. *Journal of Functional Programming* 9(1), 77–91 (1999)
4. Ching, M.: Bar constructions for topological operads and the Goodwillie derivatives of the identity. *Geom. Topol.* 9, 833–933 (2005)
5. The Coq Proof Assistant. <http://coq.inria.fr>
6. Fiore, M., Plotkin, G., Turi, D.: Abstract syntax and variable binding (extended abstract). In: 14th Symposium on Logic in Computer Science, Trento, 1999, pp. 193–202. IEEE Computer Society Press, Los Alamitos (1999)
7. Fiore, M.P.: On the structure of substitution. In: Invited address for the 22nd Mathematical Foundations of Programming Semantics Conf (MFPS XXII), DISI, University of Genova, Italy (2006)
8. Fiore, M.P., Turi, D.: Semantics of name and value passing. In: *Logic in Computer Science*, pp. 93–104 (2001)
9. Fresse, B.: Koszul duality of operads and homology of partition posets. In: *Homotopy theory: relations with algebraic geometry, group cohomology, and algebraic K-theory*, vol. 346 of *Contemp. Math.*, pp. 115–215. Amer. Math. Soc., Providence, RI (2004)
10. Gabbay, M., Pitts, A.: A new approach to abstract syntax involving binders. In: 14th Symposium on Logic in Computer Science, Trento, 1999, pp. 214–224. IEEE Computer Society Press, Los Alamitos (1999)
11. Ghani, N., Uustalu, T.: Explicit substitutions and higher-order syntax. In: *MERLIN '03. Proceedings of the 2003 ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding*, Uppsala, Sweden, pp. 1–7. ACM Press, New York, NY, USA (2003)
12. Hirschowitz, A., Maggesi, M.: The algebraicity of the lambda-calculus. [arXiv:math/0607427v1](https://arxiv.org/abs/math/0607427v1) (2007)
13. Hofmann, M.: Semantical analysis of higher-order abstract syntax. In: 14th Symposium on Logic in Computer Science, Trento, 1999, pp. 204–213. IEEE Computer Society Press, Los Alamitos (1999)
14. Livernet, M.: From left modules to algebras over an operad: application to combinatorial Hopf algebras. [arXiv:math/0607427v1](https://arxiv.org/abs/math/0607427v1) (2006)
15. Lane, S.M.: Categories for the working mathematician. In: *Graduate Texts in Mathematics*, 2nd edn., vol. 5, Springer-Verlag, New York (1998)
16. Markl, M.: Models for operads. [arXiv:hep-th/9411208v1](https://arxiv.org/abs/hep-th/9411208v1) (1994)
17. Markl, M.: A compactification of the real configuration space as an operadic completion. [arXiv:hep-th/9608067v1](https://arxiv.org/abs/hep-th/9608067v1) (1996)
18. Matthes, R., Uustalu, T.: Substitution in non-wellfounded syntax with variable binding. *Theor. Comput. Sci.* 327(1–2), 155–174 (2004)
19. Smirnov, V.A.: Homotopy theory of coalgebras. *Math. USSR Izv.* 27, 575–592 (1986)
20. Tanaka, M., Power, J.: A unified category-theoretic formulation of typed binding signatures. In: *MERLIN '05. Proceedings of the 3rd ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding*, Tallinn, Estonia, pp. 13–24. ACM Press, New York, NY, USA (2005)
21. Tanaka, M., Power, J.: Pseudo-distributive laws and axiomatics for variable binding. *Higher Order Symbol. Comput.* 19(2–3), 305–337 (2006)
22. Zsidó, J.: Le lambda calcul vu comme monade initiale. Master's thesis, Université de Nice – Laboratoire J. A. Dieudonné, 2005/06. Mémoire de Recherche – master 2.

A Appendix: Formal Proof of Theorem 3

In this section we present our formal proof of theorem 3 in the Coq proof assistant [5]. We recall the statement of the theorem

The monad Λ of semantic untyped λ -calculus is an initial object in the category of exponential monads.

We include here only a small fraction of the code without proofs. The full sources can be found at <http://www.math.unifi.it/~maggesi>.

A.1 Structure of the Formalisation

The structure of our proof can be outlined in the following four major parts: (1) axioms and support library; (2) formalisation of monads, modules and exponential monads; (3) formalisation of syntactic and semantic λ -calculus; (4) the main theorem.

The second and third part are independent of each other. As for what this paper is concerned, the first part (files `Misc.v`, `Congr.v`) can be considered as an extension of the Coq system for practical purposes. This part contains some meta-logical material (tactics and notations) and declares the following axioms: functional choice, proof irrelevance, dependent extensionality. We include here their declarations:

```
Axiom functional_choice : forall (A B : Type) (R : A -> B -> Prop),
  (forall x : A, exists y : B, R x y) -> exists f : A -> B, (forall x : A, R x (f x)).
Axiom proof_irrelevance : forall (A : Prop) (H1 H2 : A), H1 = H2.
Axiom extens_dep : forall (X : Type) (T : X -> Type) (f g : forall x : X, T x),
  (forall x : X, f x = g x) -> f = g.
```

Moreover, we use an axiomatic definition of quotient types (file `Quot.v`) to construct semantic λ -calculus as quotient of syntactic λ -calculus.

A.2 Formalisation of Monads and Modules

After the preliminary material, our formalisation opens the theory of monads and (left) modules (files `Monad.v`, `Mod.v`, `Derived_Mod.v`). This is constructed starting from a rather straightforward translation of the Haskell monad library. As an example, we report here our definitions of monads and modules in the Coq syntax.

```
Record Monad : Type := {
  monad_carrier :> Set -> Set;
  bind : forall X Y : Set, (X -> monad_carrier Y) -> monad_carrier X -> monad_carrier Y;
  unit : forall X : Set, X -> monad_carrier X;
  bind_bind : forall (X Y Z : Set) (f : X -> monad_carrier Y) (g : Y -> monad_carrier Z)
    (x : monad_carrier X),
    bind Y Z g (bind X Y f x) = bind X Z (fun u => bind Y Z g (f u)) x;
  bind_unit : forall (X Y : Set) (f : X -> monad_carrier Y) (x : X),
    bind X Y f (unit X x) = f x;
  unit_bind : forall (X : Set) (x : monad_carrier X), bind X X (unit X) x = x
}.
Notation "x >=> f" := (@bind _ _ _ f x).
```

```

Record Mod (U : Monad) : Type := {
  mod_carrier :> Set -> Set;
  mbind : forall (X Y : Set) (f : X -> U Y) (x : mod_carrier X), mod_carrier Y;
  mbind_mbind : forall (X Y Z : Set) (f : X -> U Y) (g : Y -> U Z) (x : mod_carrier X),
    mbind Y Z g (mbind X Y f x) = mbind X Z (fun u => f u >>= g) x;
  unit_mbind : forall (X : Set) (x : mod_carrier X), mbind X X (@unit U X) x = x
}.
Notation "x >>= f" := (@mbind _ _ _ f x).

```

The library also includes the definition of morphism of monads and modules and other related categorical material. Other definitions which are specific to our objective are those of derived module and exponential monad. The latter reads as follows:

```

Record ExpMonad : Type := {
  exp_monad :> Monad;
  exp_abs : Mod_Hom (Derived_Mod exp_monad) exp_monad;
  exp_app : Mod_Hom exp_monad (Derived_Mod exp_monad);
  exp_eta : forall X (x : exp_monad X), exp_abs _ (exp_app _ x) = x;
  exp_beta : forall X (x : Derived_Mod exp_monad X), exp_app _ (exp_abs _ x) = x
}.

```

and comes with its associated notion of morphism:

```

Record ExpMonad_Hom (M N : ExpMonad) : Type := {
  expmonad_hom :> Monad_Hom M N;
  expmonad_hom_app : forall X (x : M X),
    expmonad_hom _ (exp_app M _ x) = exp_app N _ (expmonad_hom _ x);
  expmonad_hom_abs : forall X (x : Derived_Mod M X),
    expmonad_hom _ (exp_abs M _ x) = exp_abs N _ (expmonad_hom _ x)
}.

```

A.3 Formalisation of the λ -Calculus

This part contains the definition of syntactic and semantic λ -calculus (files Slc.v and Lc.v respectively). We use nested datatypes to encode λ -terms in the Calculus of (Co)Inductive Constructions as already shown in the Haskell fragment of section 5.1 for which we report below the equivalent Coq code. Notice that this encoding can be considered a typeful variant of the well-known de Bruijn encoding [3]. As the de Bruijn encoding, it represents λ -terms modulo α -conversion.

```

Inductive term (X : Set) : Set := var : X -> term X
| app : term X -> term X -> term X
| abs : term (option X) -> term X.

Fixpoint fct (X Y : Set) (f : X -> Y) (x : term X) { struct x } : term Y :=
  match x with var a => var (f a)
| app x y => app (x //- f) (y //- f)
| abs x => abs (x //- (optmap f)) end
where "x //- f" := (@fct _ _ f x).

Definition shift X (x : term X) : term (option X) := x //- @Some X.

```

```
Definition comm (X Y : Set) (f : X -> term Y) (x : option X) : term (option Y) :=
  match x with Some a => shift (f a) | None => var None end.
```

```
Fixpoint subst (X Y : Set) (f : X -> term Y) (x : term X) { struct x } : term Y :=
  match x with var x => f x
    | app x y => app (x // f) (y // f)
    | abs x => abs (x // comm f) end
where "x // f" := (@subst _ _ f x).
```

Once the basic definitions are settled, we prove a series of basic lemmas which includes the associativity of substitution

```
Lemma subst_subst : forall (X Y Z : Set) (f : X -> term Y) (g : Y -> term Z) (x : term X),
  x // f // g = x // fun u => f u // g.
```

which is the most important ingredient to prove that the λ -calculus is a monad. Finally, we introduce the beta-eta equivalence relation on lambda terms

```
Inductive lcr (X : Set) : term X -> term X -> Prop :=
| lcr_var : forall a : X, var a == var a
| lcr_app : forall x1 x2 y1 y2 : term X, x1 == x2 -> y1 == y2 -> app x1 y1 == app x2 y2
| lcr_abs : forall x y : term (option X), x == y -> abs x == abs y
| lcr_beta : forall x y : term X, Beta x y -> x == y
| lcr_eta : forall x : term X, abs (app1 x) == x
| lcr_sym : forall x y : term X, y == x -> x == y
| lcr_trs : forall x y z : term X, lcr x y -> lcr y z -> lcr x z
where "x == y" := (@lcr _ _ x y).
```

and prove some compatibility lemmas for constructors and other operations. The compatibility of substitution is stated as follows:

```
Lemma lcr_subst : forall (X Y : Set) (f g : X -> term Y) (x y : term X),
  (forall u, f u == g u) -> x == y -> x // f == y // g.
```

A.4 Proof of the Main Theorem

The fourth and last part summarises the results proved in the other parts and proves the main theorem. It starts by glueing together the previous two sections by proving that our definitions of syntactic and semantic lambda calculus provides indeed two monads, denoted SLC and LC respectively, and by showing that the two morphisms `abs` and `app1` constitute morphisms of modules:

```
Definition SLC : Monad := Build_Monad term subst var subst_subst subst_var var_subst.
```

```
Definition LC : Monad :=
  Build_Monad lc lc_subst lc_var lc_subst_assoc lc_subst_var lc_var_subst.
```

```
Let abs_hom : Mod_Hom (Derived_Mod LC) LC :=
  Build_Mod_Hom (Derived_Mod LC) LC lc_abs lc_abs_hom.
```

```
Let app1_hom : Mod_Hom LC (Derived_Mod LC) :=
  Build_Mod_Hom LC (Derived_Mod LC) lc_app1 lc_app1_hom.
```

One more glueing step is the proof that LC is an exponential monad, which is stated in Coq through the following definition:

```
Definition ELC : ExpMonad := Build_ExpMonad abs_hom app1_hom lc_eta lc_beta.
```

Next comes the construction of the initial morphism which is initially defined as a fixpoint on terms.

```

Variable M : ExpMonad.
Fixpoint iota_fix X (x : term X) { struct x } : M X :=
  match x with var a => unit M a
    | app x y => exp_app M _ (iota_fix x) >>= default (@unit M X) (iota_fix y)
    | abs x => exp_abs M _ (iota_fix x) end.

```

Then we prove that `iota_fix` is compatible with the $\beta\eta$ equivalence relation and thus it factors through the monad `LC`.

```

Let iota X : lc X -> M X := lc_factor (@iota_fix X) (@iota_fix_wd X).

```

The construction of the initial morphism ends with the verification that it is actually a morphism of exponential monads.

```

Let iota_monad : Monad_Hom LC M := Build_Monad_Hom LC M iota iota_subst iota_var.

```

```

Let exp_iota : ExpMonad_Hom ELC M :=
  Build_ExpMonad_Hom ELC M iota_monad iota_app1 iota_abs.

```

Finally, we prove that `iota_monad` is unique.

```

Theorem iota_unique : forall j : ExpMonad_Hom ELC M, j = exp_iota.

```

The Coq terms `ELC`, `iota_monad` and `iota_unique` altogether form the formal proof of the initiality of the monad `LC` in the category of exponential monads.

Hydra Games and Tree Ordinals

Ariya Isihara

Department of Computer Science, Vrije Universiteit, Amsterdam

`ariya@few.vu.nl`

Abstract. Hydra games were introduced by Kirby and Paris, for the formulation of a result which is independent from Peano arithmetic but depends on the transfinite structure of ϵ_0 . Tree ordinals are a well-known simple way to represent countable ordinals. In this paper we study the relation between these concepts; an ordinal less than ϵ_0 is canonically translated into both a hydra and a tree ordinal term, and the reduction graph of the hydra and the normal form of the term syntactically correspond to each other.

1 Introduction

Tree ordinals are one of the simplest ways known for representing countable ordinals. The main concept of tree ordinals is to represent limit ordinals by fundamental sequences themselves. The notation of tree ordinals naturally fits to Dedekind's TRS [2], which gives a simple definition for basic arithmetics for ordinals — addition, multiplication, and exponentiation. With these arithmetic functions, we can represent ordinals up to ϵ_0 by finite expressions.

On the other hand, Hydra games were invented by Kirby and Paris [4] for the formulation of an undecidable statement. The termination of hydra games cannot be proved within Peano arithmetic, but under the assumption that the ordinal ϵ_0 is well-ordered. There, nondeterministic behaviour of hydrae takes place to represent infiniteness of transfinite ordinals.

In the present work we translate infinite sequences in tree ordinals and non-deterministic reductions in hydra games into each other. Thereby we will see a syntactical correspondence between tree ordinals and hydra games.

Overview

In Section 2 we present Dedekind's TRS on tree ordinals. In Section 3 we recall the definition of hydra games. Section 4 shows the correspondence between hydra games and tree ordinals. Section 5 gives a concluding remark.

2 Tree Ordinals

Definition 1. The set \mathcal{T} of *tree ordinal terms* consists of the possibly infinite terms generated by the following symbols

arity	symbol(s)
0	$0, \omega$
1	succ, nats
2	$\text{cons}, \text{Add}, \text{Mul}, \text{Exp}$

with the rewrite rules

$$\begin{aligned}
& \text{Add}(x, 0) \rightarrow x \\
& \text{Add}(x, \text{succ}(y)) \rightarrow \text{succ}(\text{Add}(x, y)) \\
& \text{Add}(x, \text{cons}(y, z)) \rightarrow \text{cons}(\text{Add}(x, y), \text{Add}(x, z)) \\
& \text{Mul}(x, 0) \rightarrow 0 \\
& \text{Mul}(x, \text{succ}(y)) \rightarrow \text{Add}(\text{Mul}(x, y), x) \\
& \text{Mul}(x, \text{cons}(y, z)) \rightarrow \text{cons}(\text{Mul}(x, y), \text{Mul}(x, z)) \\
& \text{Exp}(x, 0) \rightarrow \text{succ}(0) \\
& \text{Exp}(x, \text{succ}(y)) \rightarrow \text{Mul}(\text{Exp}(x, y), x) \\
& \text{Exp}(x, \text{cons}(y, z)) \rightarrow \text{cons}(\text{Exp}(x, y), \text{Exp}(x, z)) \\
& \omega \rightarrow \text{nats}(0) \\
& \text{nats}(x) \rightarrow x : \text{nats}(\text{succ}(x)).
\end{aligned}$$

Observe that the rules are orthogonal and that there exists only one collapsing rule. The system is thus CR^∞ and UN^∞ (See [7] or [5]). We write $\text{nf}(t)$ to denote the (possibly infinite) normal form of t , if it exists.

We write $t : s$ for $\text{cons}(t, s)$, where ‘:’ is right-associative; $t_0 : t_1 : t_2 : \dots$ represents a infinite sequence of terms t_0, t_1, t_2, \dots .

Definition 2. The set $\mathcal{TO} \subset \mathcal{T}$ of *tree ordinals* is given as the smallest set that satisfies the following conditions:

$$\begin{aligned}
0 & \in \mathcal{TO} \\
\text{succ}(t) & \in \mathcal{TO} & (t \in \mathcal{TO}) \\
t_0 : t_1 : t_2 : \dots & \in \mathcal{TO} & (t_0, t_1, t_2, \dots \in \mathcal{TO})
\end{aligned}$$

with *semantics* $\llbracket - \rrbracket : \mathcal{TO} \rightarrow \Omega$ inductively given by

$$\begin{aligned}
\llbracket 0 \rrbracket &= 0 \\
\llbracket \text{succ}(t) \rrbracket &= \llbracket t \rrbracket + 1 \\
\llbracket t_0 : t_1 : t_2 : \dots \rrbracket &= \limsup_{i \in \mathbb{N}} \llbracket t_i \rrbracket
\end{aligned}$$

where Ω denotes the set of countable ordinals.

Definition 3. We define a certain subset \mathcal{E}_0 of \mathcal{T} given by the following BNF:

$$\mathcal{E}_0 ::= 0 \mid \text{Add}(\mathcal{E}_0, \mathcal{E}_0) \mid \text{Exp}(\omega, \mathcal{E}_0).$$

Let \sim be the equivalence relation generated by

$$\begin{aligned}\text{Add}(\text{Add}(t, s), u) &\sim \text{Add}(t, \text{Add}(s, u)) \\ \text{Add}(t, 0) &\sim t \\ \text{Add}(0, t) &\sim t\end{aligned}$$

and let $[\mathcal{E}_0]$ be the set of equivalence classes of \mathcal{E}_0 modulo \sim .

We write $\sum_{i=1}^n t_i$ and ω^t for $\text{Add}(t_1, \dots, \text{Add}(t_{n-1}, t_n))$ and $\text{Exp}(\omega, t)$, respectively. We regard $\sum_{i=1}^0 t_i$ as 0.

Proposition 4 (Induction on $[\mathcal{E}_0]$). *For any $t \in \mathcal{E}_0$, there exist $n \in \mathbb{N}$ and $t_1, \dots, t_n \in \mathcal{E}_0$ such that $t \sim \sum_{i=1}^n \omega^{t_i}$. Moreover, the equivalence classes in $[\mathcal{E}_0]$ are enumerated by this construction. Namely, the following induction principle holds: Let $P \subset [\mathcal{E}_0]$. If $\sum_{i=1}^n \omega^{t_i} \in P$ for all $n \in \mathbb{N}$ and $t_1, \dots, t_n \in P$, then $P = [\mathcal{E}_0]$. \square*

In the remainder of this section we state some propositions on the productivity of our system, without proofs. Since the system forms a subclass of the system presented in [6], please see *ibid.* for the proofs.

Lemma 5. *For any $t \in \mathcal{E}_0$, $\text{nf}(t)$ does exist. \square*

Lemma 6. *If $t \sim s$, then $\text{nf}(t) = \text{nf}(s)$. The function $\text{nf} : [\mathcal{E}_0] \rightarrow \mathcal{T}$ is thus canonically defined. \square*

Let $\mathcal{TO}_{\mathcal{E}_0}$ be the image of $[\mathcal{E}_0]$ via nf .

Proposition 7. *We have $\mathcal{TO}_{\mathcal{E}_0} \subset \mathcal{TO}$. \square*

Theorem 8. *Let $t, s \in \mathcal{E}_0$. Then the following equations hold:*

$$\begin{aligned}[\text{nf}(0)] &= 0 \\ [\text{nf}(\text{Add}(t, s))] &= [\text{nf}(t)] + [\text{nf}(s)] \\ [\text{nf}(\text{Exp}(\omega, t))] &= \omega^{[\text{nf}(t)]}.\end{aligned}$$

And therefore the image of $\mathcal{TO}_{\mathcal{E}_0}$ via $[-]$ is ϵ_0 . \square

Definition 9. For any α in ϵ_0 , the *nested Cantor normal form* of α , written $\text{ncn}(\alpha)$, is inductively defined by

- (i) if $\alpha = 0$, then $\text{ncn}(\alpha) = 0$.
- (ii) if $\alpha = \beta + 1$, then $\text{ncn}(\alpha) = \text{ncn}(\beta) + \text{Exp}(\omega, 0)$.
- (iii) if α is a limit ordinal, then there exist unique $\beta, \gamma < \alpha$ such that $\alpha = \omega^\beta + \gamma$.
Let $\text{ncn}(\alpha) = \text{Exp}(\omega, \text{ncn}(\beta)) + \text{ncn}(\gamma)$.

For example, $ncn(\omega \times 2)$ is computed as follows:

$$\begin{aligned}
 ncn(\omega \times 2) &= ncn(\omega^1 + \omega^1) \\
 &= \omega^{ncn(1)} + ncn(\omega^1) \\
 &= \omega^{ncn(0+1)} + \omega^{ncn(0+1)} \\
 &= \omega^{ncn(0)+\omega^0} + \omega^{ncn(0)+\omega^0} \\
 &\sim \omega^{\omega^0} + \omega^{\omega^0}.
 \end{aligned}$$

Proposition 10. *We have $\llbracket - \rrbracket \circ nf \circ ncn = \mathbf{id}_{\epsilon_0}$. Thus, the system $[\mathcal{E}_0]$ can compute every ordinal less than ϵ_0 . \square*

3 Hydra Games

In this section, we present hydra games [4] with a minor change.

Definition 11. A *hydra* is an unlabeled finite tree with arbitrary finite branches. As a hydra, a leaf node is called a *head*. A head is *short* if the immediate parent of the head is the root; *long* if it is neither short nor the root (See Figure 1). The empty hydra is called *dead* and written \bigcirc .

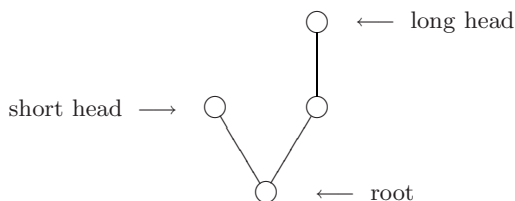


Fig. 1. A hydra

Thus, formally, the set \mathcal{H} of hydrae is inductively defined as follows:

$$(h_1, \dots, h_n) \in \mathcal{H} \quad \text{if} \quad h_1, \dots, h_n \in \mathcal{H}$$

where $()$, in the case $n = 0$, is regarded as \bigcirc .

As in the original paper, Herakles chops a head of a hydra. However, in this paper Herakles chops only the rightmost-head of the hydra. Hence, we are mainly interested in the rightmost structure of a hydra so that we take the notation

$$(h_1(h_2(\dots(h_n \bigcirc))))$$

to describe a given hydra, where h_i denotes the juxtaposition of n_i many hydrae $h_{i1} \dots h_{in_i}$. This hydra is depicted by Figure 2. Observe that, for any hydra, there exists a unique representation of this form.

Now we define the *chopping-relation* on the hydrae.

and

$$\begin{aligned}
 (\omega^0 + \widehat{\omega^{\omega^{\omega^0} + \omega^0}}) &= (\widehat{0} (\widehat{\omega^{\omega^0} + \omega^z})) \\
 &= (\widehat{0} (\widehat{\omega^z} \widehat{0})) \\
 &= (\widehat{0} ((\widehat{0}) \widehat{0})) \\
 &= (\widehat{0} ((\widehat{0}) \widehat{0})).
 \end{aligned}$$

Before we state our main result, we show some lemmas.

Lemma 17. *The following propositions hold:*

- (i) Let $t, s \in \mathcal{E}_0$. Then $nf(t + s) = nf(s)[0 := nf(t)]$.
- (ii) Let $h_1, \dots, h_n \in \mathcal{H}$ where $n > 0$. Then we have

$$gr(h_1 \dots h_n) = gr(h_n)[0 := gr(h_1 \dots h_{n-1})].$$

Proof. (i) By induction on $nf(s)$.

- (ii) Notice that for any $h \in \mathcal{H}$ such that $h \xrightarrow{m} h'$ where $m \in \{!\} \cup \mathbb{N}$, we have

$$(h_1 \dots h_{n-1} h) \xrightarrow{m} (h_1 \dots h_{n-1} h').$$

Thus, the claim follows by induction on $gr(h_n)$.

Lemma 18. *Let $t \in [\mathcal{E}_0]$. Then the following holds:*

- (i) If $\llbracket nf(t) \rrbracket = 0$, then $t = 0$.
- (ii) If $\llbracket nf(t) \rrbracket$ is a successor ordinal, then t is of the form $\sum_{i=1}^n \omega^{t_i}$ where $t_n = 0$.

Proof. (i) Suppose $t = \sum_{i=1}^n \omega^{t_i}$. Then from Theorem 8 we have

$$\llbracket nf(t) \rrbracket = \omega^{\llbracket nf(t_1) \rrbracket} + \dots + \omega^{\llbracket nf(t_n) \rrbracket}.$$

Since $\omega^\alpha > 0$ for every ordinal α , we have $n = 0$.

- (ii) Suppose $t = \sum_{i=1}^n \omega^{t_i}$. Similarly, we have $n > 0$ and that $\omega^{\llbracket nf(t_n) \rrbracket}$ is a successor ordinal. Hence, $\llbracket nf(t_n) \rrbracket = 0$. From the above result, we have $t_n = 0$.

Lemma 19. *Let $t \in [\mathcal{E}_0]$. Then one of the following conditions holds:*

- (i) $\llbracket nf(t) \rrbracket = 0$.
- (ii) There exists $t' \in \mathcal{TO}_{\mathcal{E}_0}$ such that $nf(t) = \text{succ}(t')$ and $\llbracket nf(t) \rrbracket = \llbracket nf(t') \rrbracket + 1$.
- (iii) There exist $t_0, t_1, \dots \in \mathcal{TO}_{\mathcal{E}_0}$ such that $nf(t) = t_0 : t_1 : \dots$ and $\llbracket t_i \rrbracket < \llbracket nf(t) \rrbracket$ for all $i \in \mathbb{N}$.

Proof. Using Theorem 8, it easily follows by induction on t .

Now we set $P = \{t \in [\mathcal{E}_0] \mid nf(t) = gr(\widehat{t})\}$.

Lemma 20. *Let $t_1, \dots, t_n \in \mathcal{E}_0$. If $t_i \in P$ for all applicable i , then $\sum_{i=1}^n t_i \in P$.*

Proof. By mathematical induction on n . It is trivial for the case $n = 0$. For induction step, we assume $\sum_{i=1}^{n-1} t_i \in P$. We have

$$nf\left(\sum_{i=1}^n t_i\right) = nf(t_n) \left[0 := nf\left(\sum_{i=1}^{n-1} t_i\right) \right]$$

and

$$gr\left(\widehat{\sum_{i=1}^n t_i}\right) = gr(\widehat{t_n}) \left[0 := gr\left(\widehat{\sum_{i=1}^{n-1} t_i}\right) \right]$$

by Lemma 17. Thus, by induction hypothesis, we have

$$nf\left(\sum_{i=1}^n t_i\right) = gr\left(\widehat{\sum_{i=1}^n t_i}\right),$$

implying $\sum_{i=1}^n t_i \in P$.

Lemma 21. *If $t \in P$, then $\omega^t \in P$.*

Proof. Transfinite induction on $\llbracket nf(t) \rrbracket$. We suppose that $\llbracket nf(s) \rrbracket < \llbracket nf(t) \rrbracket$ implies $s \in P$. Case analysis by Lemma 19.

(i) If $\llbracket nf(t) \rrbracket = 0$, then from Lemma 18 we have $t = 0$ and thus

$$nf(\omega^t) = gr(\widehat{t}) = \text{succ}(0).$$

Hence, $\omega^t \in P$.

(ii) If $nf(t) = \text{succ}(s)$ where $\llbracket nf(t) \rrbracket = \llbracket s \rrbracket + 1$, then from Lemma 18 we have $t = \sum_{i=1}^{n-1} \omega^{t_i} + \omega^0$ with $nf\left(\sum_{i=1}^{n-1} t_i\right) = s$. Let $t' = \sum_{i=1}^{n-1} t_i$. Then

$$\begin{aligned} nf(\omega^t) &= nf(\omega^{t' + \text{succ}(0)}) \\ &= nf(\text{Mul}(\omega^{t'}, \omega)) \\ &= 0 : nf(\omega^{t'}) : nf(\text{Add}(\omega^{t'} + \omega^{t'})) : \dots \end{aligned}$$

and

$$\begin{aligned} gr(\widehat{\omega^t}) &= gr(\widehat{t'} (\bigcirc)) \\ &= gr(\bigcirc) : gr(\widehat{t'}) : gr(\widehat{t'} \widehat{t'}) : \dots \end{aligned}$$

We have $nf(\sum_{i=1}^m t') = gr(\widehat{\sum_{i=1}^m t'}) = gr(\widehat{t'} \dots \widehat{t'})$ for all $m \in \mathbb{N}$, by induction hypothesis and Lemma 20. Therefore, we have $nf(\omega^t) = gr(\widehat{\omega^t})$, implying $\omega^t \in P_x$.

Theorem 22. *We have $nf \circ \overline{(-)} = gr$ and $gr \circ \widehat{(-)} = nf$.*

Proof. The latter equation follows from Proposition 4 and the above two lemmas. The other one then immediately follows, using Proposition 16.

5 Conclusion

We have presented the syntactical correspondence between hydra games and a certain subclass of tree ordinals. The contribution of the paper is depicted by Figure 5, where the upper-left triangle commutes.

Since Buchholz [1] gives a variation of hydra games which is related to a much larger ordinal, it is expected that there exists an extension of this TRS which is related to Buchholz's hydra games.

In addition, a visualisation tool for the original hydra games due to Kirby and Paris is available at [3]; Figure 6 is a screenshot of the tool.

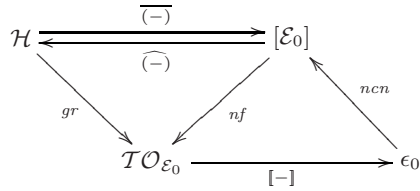


Fig. 5. The diagram which relates hydra games and tree ordinals

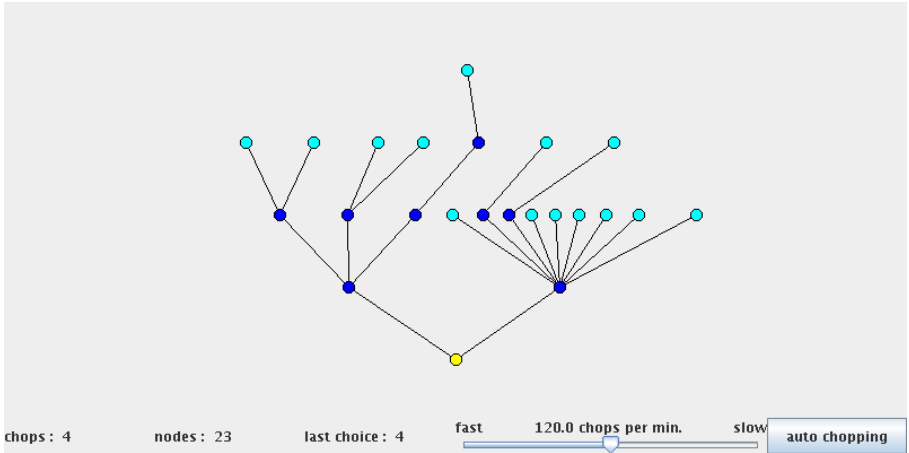


Fig. 6. Screenshot of hydra (JavaApplet)

Acknowledgement

I would thank Andreas Weiermann, Jan Willem Klop and Vincent van Oostrom. Andreas noticed the underlying similarity on hydra games and tree ordinals. Jan Willem showed me interest on my draft idea. Vincent gave me a chance to present my draft idea in TeReSe meeting in Utrecht. I also acknowledge Roel de Vrijer and Clemens Grabmayer for their helpful comments.

References

1. Buchholz, W.: An independence result for $(\Pi_1^1\text{-CA}) + \text{BI}$. *Annals of Pure and Applied Logic* 33(2), 131–155 (1987)
2. Dedekind, R.: *Was sind und was sollen die Zahlen?* Vieweg (1888)
3. Ishihara, A.: hydra (JavaApplet). Available at <http://www.few.vu.nl/~ariya/app/hydra/hydra.html>
4. Kirby, L., Paris, J.: Accessible independence results for Peano Arithmetic. *Bulletin of the London Mathematical Society* 14, 285–293 (1982)
5. Klop, J.W., de Vrijer, R.: Infinitary normalization. In: Artemov, S., Barringer, H., d’Avila Garcez, A., Lamb, L., Woods, J. (eds.) *We Will Show Them: Essays in Honour of Dov Gabbay*, vol. 2, pp. 169–192. College Publications (2005)
6. Kwiatkowski, M.: Ordinal arithmetic through infinitary term rewriting. Master’s thesis, Vrije Universiteit, Amsterdam, The Netherlands (2006)
7. Terese, J.: *Term Rewriting Systems*. Cambridge University Press, Cambridge (2003)

Spin Networks, Quantum Topology and Quantum Computation

Louis H. Kauffman¹ and Samuel J. Lomonaco Jr.²

¹ Department of Mathematics, Statistics and Computer Science (m/c 249), 851 South Morgan Street, University of Illinois at Chicago, Chicago, Illinois 60607-7045, USA

`kauffman@uic.edu`

² Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, USA

`lomonaco@umbc.edu`

Abstract. We review the q -deformed spin network approach to Topological Quantum Field Theory and apply these methods to produce unitary representations of the braid groups that are dense in the unitary groups. The results are applied to the quantum computation of colored Jones polynomials.

Keywords: braiding, knotting, linking, spin network, Temperley – Lieb algebra, unitary representation.

1 Introduction

This paper, a short version of [13], describes the background for topological quantum computing in terms of Temperley – Lieb Recoupling Theory. This is a recoupling theory that generalizes standard angular momentum recoupling theory, generalizes the Penrose theory of spin networks and is inherently topological. Temperley – Lieb Recoupling Theory is based on the bracket polynomial model for the Jones polynomial. It is built in terms of diagrammatic combinatorial topology. The same structure can be explained in terms of the $SU(2)_q$ quantum group, and has relationships with functional integration and Witten’s approach to topological quantum field theory. Nevertheless, the approach given here will be unrelentingly elementary. Elementary, does not necessarily mean simple. In this case an architecture is built from simple beginnings and this architecture and its recoupling language can be applied to many things including: colored Jones polynomials, Witten–Reshetikhin–Turaev invariants of three manifolds, topological quantum field theory and quantum computing.

In quantum computing, the application is most interesting because the recoupling theory yields representations of the Artin Braid group into unitary groups $U(n)$. These representations are *dense* in the unitary group, and can be used to model quantum computation universally in terms of representations of the braid group. Hence the term: topological quantum computation.

In this paper, we outline the basics of the Temperley – Lieb Recoupling Theory, and show explicitly how unitary representations of the braid group arise

from it. We will return to this subject in more detail in subsequent papers. In particular, we do not describe the context of anyonic models for quantum computation in this paper. Rather, we concentrate here on showing how naturally unitary representations of the braid group arise in the context of the Temperley – Lieb Theory. For the reader interested in the relevant background in anyonic topological quantum computing we recommend the following references [3,4,5,6,7,14,15,17,18].

Here is a very condensed presentation of how unitary representations of the braid group are constructed via topological quantum field theoretic methods. For simplicity assume that one has a single (mathematical) particle with label P that can interact with itself to produce either itself labeled P , or itself with the null label $*$. When $*$ interacts with P the result is always P . When $*$ interacts with $*$ the result is always $*$. One considers process spaces where a row of particles labeled P can successively interact, subject to the restriction that the end result is P . For example the space $V[(ab)c]$ denotes the space of interactions of three particles labeled P . The particles are placed in the positions a, b, c . Thus we begin with $(PP)P$. In a typical sequence of interactions, the first two P 's interact to produce a $*$, and the $*$ interacts with P to produce P .

$$(PP)P \longrightarrow (*)P \longrightarrow P.$$

In another possibility, the first two P 's interact to produce a P , and the P interacts with P to produce P .

$$(PP)P \longrightarrow (P)P \longrightarrow P.$$

It follows from this analysis that the space of linear combinations of processes $V[(ab)c]$ is two dimensional. The two processes we have just described can be taken to be the qubit basis for this space. One obtains a representation of the three strand Artin braid group on $V[(ab)c]$ by assigning appropriate phase changes to each of the generating processes. One can think of these phases as corresponding to the interchange of the particles labeled a and b in the association $(ab)c$. The other operator for this representation corresponds to the interchange of b and c . This interchange is accomplished by a *unitary change of basis mapping*

$$F : V[(ab)c] \longrightarrow V[a(bc)].$$

If

$$A : V[(ab)c] \longrightarrow V[(ba)c]$$

is the first braiding operator (corresponding to an interchange of the first two particles in the association) then the second operator

$$B : V[(ab)c] \longrightarrow V[(ac)b]$$

is accomplished via the formula $B = F^{-1}AF$ where the A in this formula acts in the second vector space $V[a(bc)]$ to apply the phases for the interchange of b and c .

In this scheme, vector spaces corresponding to associated strings of particle interactions are interrelated by *recoupling transformations* that generalize the mapping F indicated above. A full representation of the Artin braid group on each space is defined in terms of the local interchange phase gates and the recoupling transformations. These gates and transformations have to satisfy a number of identities in order to produce a well-defined representation of the braid group. These identities were discovered originally in relation to topological quantum field theory. In our approach the structure of phase gates and recoupling transformations arise naturally from the structure of the bracket model for the Jones polynomial [8]. Thus we obtain a knot-theoretic basis for topological quantum computing.

2 Spin Networks and Temperley – Lieb Recoupling Theory

In this section we discuss a combinatorial construction for spin networks that generalizes the original construction of Roger Penrose [16]. The result of this generalization is a structure that satisfies all the properties of a graphical *TQFT* as described in our paper on braiding and universal quantum gates [12], and specializes to classical angular momentum recoupling theory in the limit of its basic variable. The construction is based on the properties of the bracket polynomial [9]. A complete description of this theory can be found in the book “Temperley – Lieb Recoupling Theory and Invariants of Three-Manifolds” by Kauffman and Lins [11].

The “ q -deformed” spin networks that we construct here are based on the bracket polynomial relation. View Figure 1 and Figure 2.

In Figure 1 we indicate how the basic projector (symmetrizer, Jones-Wenzl projector) is constructed on the basis of the bracket polynomial expansion [9]. In this technology, a symmetrizer is a sum of tangles on n strands (for a chosen integer n). The tangles are made by summing over braid lifts of permutations in the symmetric group on n letters, as indicated in Figure 1. Each elementary braid is then expanded by the bracket polynomial relation, as indicated in Figure 1, so that the resulting sum consists of flat tangles without any crossings (these can be viewed as elements in the Temperley – Lieb algebra). The projectors have the property that the concatenation of a projector with itself is just that projector, and if you tie two lines on the top or the bottom of a projector together, then the evaluation is zero. This general definition of projectors is very useful for this theory. The two-strand projector is shown in Figure 2. Here the formula for that projector is particularly simple. It is the sum of two parallel arcs and two turn-around arcs (with coefficient $-1/d$, with $d = -A^2 - A^{-2}$ is the loop value for the bracket polynomial. Figure 2 also shows the recursion formula for the general projector. This recursion formula is due to Jones and Wenzl and the projector in this form, developed as a sum in the Temperley – Lieb algebra (see Section 5 of this paper), is usually known as the *Jones–Wenzl projector*.

The projectors are combinatorial analogs of irreducible representations of a group (the original spin nets were based on $SU(2)$ and these deformed nets are

$$\begin{aligned}
 \tilde{\times} &= \times \quad \bigcirc = -A^2 \cdot A^{-2} = d \\
 \times &= A \quad \text{cup} + A^{-1} \quad \text{cap} \\
 \begin{array}{|c} \hline n \\ \hline \end{array} &= \begin{array}{|c} \hline \vdots \\ \hline \end{array} \quad \begin{array}{|c} \hline n \\ \hline \end{array} = \begin{array}{|c} \hline n \\ \hline \end{array} \\
 &\quad \text{n strands} \\
 \{n\}! &= \sum_{\sigma \in S_n} (A^{-4})^{t(\sigma)} \quad \begin{array}{|c} \hline \vdots \\ \hline \end{array} = 0 \\
 \begin{array}{|c} \hline n \\ \hline \end{array} &= (1/\{n\}!) \sum_{\sigma \in S_n} (A^{-3})^{t(\sigma)} \quad \begin{array}{|c} \hline \tilde{\sigma} \\ \hline \end{array}
 \end{aligned}$$

Fig. 1. Basic Projectors

$$\begin{aligned}
 \begin{array}{|c} \hline 2 \\ \hline \end{array} &= \begin{array}{|c} \hline \vdots \\ \hline \end{array} = \begin{array}{|c} \hline \vdots \\ \hline \end{array} - 1/\delta \quad \begin{array}{|c} \hline \cup \\ \hline \end{array} \\
 \begin{array}{|c} \hline n \quad 1 \quad 1 \\ \hline \end{array} &= \begin{array}{|c} \hline n \quad 1 \quad 1 \\ \hline \end{array} - \Delta_{n+1} \quad \begin{array}{|c} \hline \cup \\ \hline \end{array} \\
 \Delta_{-1} &= 0 \quad \Delta_0 = 1 \\
 \Delta_{n+1} &= \delta \Delta_n - \Delta_{n-1}
 \end{aligned}$$

Fig. 2. Two Strand Projector

based on the quantum group corresponding to $SU(2)$). As such the reader can think of them as “particles”. The interactions of these particles are governed by how they can be tied together into three-vertices. See Figure 3. In Figure 3 we show how to tie three projectors, of a, b, c strands respectively, together to form a three-vertex. In order to accomplish this interaction, we must share lines between them as shown in that Figure so that there are non-negative integers i, j, k so that $a = i + j, b = j + k, c = i + k$. This is equivalent to the condition

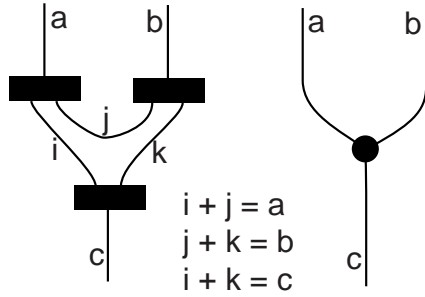


Fig. 3. Trivalent Vertex

that $a + b + c$ is even and that the sum of any two of a, b, c is greater than or equal to the third. For example $a + b \geq c$. One can think of the vertex as a possible particle interaction where $[a]$ and $[b]$ interact to produce $[c]$. That is, any two of the legs of the vertex can be regarded as interacting to produce the third leg.

There is a basic orthogonality of three vertices as shown in Figure 4. Here if we tie two three-vertices together so that they form a “bubble” in the middle, then the resulting network with labels a and b on its free ends is a multiple of an a -line (meaning a line with an a -projector on it) or zero (if a is not equal to b). The multiple is compatible with the results of closing the diagram in the equation of Figure 4 so the the two free ends are identified with one another. On closure, as shown in the Figure, the left hand side of the equation becomes a Theta graph and the right hand side becomes a multiple of a “delta” where Δ_a denotes the bracket polynomial evaluation of the a -strand loop with a projector on it. The $\Theta(a, b, c)$ denotes the bracket evaluation of a theta graph made from three trivalent vertices and labeled with a, b, c on its edges.

There is a recoupling formula in this theory in the form shown in Figure 5. Here there are “6-j symbols”, recoupling coefficients that can be expressed, as shown in Figure 7, in terms of tetrahedral graph evaluations and theta graph evaluations. The tetrahedral graph is shown in Figure 6. One derives the formulas for these coefficients directly from the orthogonality relations for the trivalent vertices by closing the left hand side of the recoupling formula and using orthogonality to evaluate the right hand side. This is illustrated in Figure 7.

Finally, there is the braiding relation, as illustrated in Figure 8.

With the braiding relation in place, this q -deformed spin network theory satisfies the pentagon, hexagon and braiding naturality identities needed for a topological quantum field theory. All these identities follow naturally from the basic underlying topological construction of the bracket polynomial. One can apply the theory to many different situations.

2.1 Evaluations

In this section we discuss the structure of the evaluations for Δ_n and the theta and tetrahedral networks. We refer to [11] for the details behind these formulas.

$$\begin{aligned}
 a \mid &= \boxed{a} \\
 a \bigcirc &= a \boxed{} = \Delta_a \\
 \begin{array}{c} \bullet \\ \text{c} \end{array} \begin{array}{c} \bullet \\ \text{d} \end{array} \begin{array}{c} \bullet \\ \text{a} \end{array} &= \Theta(a, c, d) \\
 \boxed{\begin{array}{c} \bullet \\ \text{a} \end{array} \begin{array}{c} \bullet \\ \text{c} \end{array} \begin{array}{c} \bullet \\ \text{d} \end{array} \begin{array}{c} \bullet \\ \text{b} \end{array}} &= \frac{\Theta(a, c, d)}{\Delta_a} \mid \begin{array}{c} \text{a} \\ \delta_b^a \end{array}
 \end{aligned}$$

Fig. 4. Orthogonality of Trivalent Vertices

$$\begin{array}{c} \text{a} \\ \bullet \end{array} \begin{array}{c} \text{b} \\ \bullet \end{array} \begin{array}{c} \text{c} \\ \bullet \end{array} \begin{array}{c} \text{d} \\ \bullet \end{array} \begin{array}{c} \text{i} \\ \text{---} \end{array} = \sum_j \left\{ \begin{array}{ccc} \text{a} & \text{b} & \text{i} \\ \text{c} & \text{d} & \text{j} \end{array} \right\} \begin{array}{c} \text{a} \\ \bullet \end{array} \begin{array}{c} \text{b} \\ \bullet \end{array} \begin{array}{c} \text{c} \\ \bullet \end{array} \begin{array}{c} \text{d} \\ \bullet \end{array} \begin{array}{c} \text{j} \\ \text{---} \end{array}$$

Fig. 5. Recoupling Formula

$$\begin{array}{c} \bullet \\ \text{a} \end{array} \begin{array}{c} \bullet \\ \text{b} \end{array} \begin{array}{c} \bullet \\ \text{c} \end{array} \begin{array}{c} \bullet \\ \text{d} \end{array} \begin{array}{c} \text{i} \\ \text{---} \end{array} \begin{array}{c} \text{k} \\ \text{---} \end{array} = \text{Tet} \left[\begin{array}{ccc} \text{a} & \text{b} & \text{i} \\ \text{c} & \text{d} & \text{k} \end{array} \right]$$

Fig. 6. Tetrahedron Network

Recall that Δ_n is the bracket evaluation of the closure of the n -strand projector, as illustrated in Figure 4. For the bracket variable A , one finds that

$$\Delta_n = (-1)^n \frac{A^{2n+2} - A^{-2n-2}}{A^2 - A^{-2}}.$$

$$\begin{aligned} & \left(\text{tetrahedron with vertices } a, b, c, d \text{ and internal lines } i, j, k \right) = \sum_j \left\{ \begin{matrix} a & b & i \\ c & d & j \end{matrix} \right\} \left(\text{tetrahedron with internal line } j \right)_k \\ &= \sum_j \left\{ \begin{matrix} a & b & i \\ c & d & j \end{matrix} \right\} \frac{\Theta(a, b, j)}{\Delta_j} \frac{\Theta(c, d, j)}{\Delta_j} \Delta_j \delta_j^k \\ &= \left\{ \begin{matrix} a & b & i \\ c & d & k \end{matrix} \right\} \frac{\Theta(a, b, k) \Theta(c, d, k)}{\Delta_k} \end{aligned}$$

$$\left\{ \begin{matrix} a & b & i \\ c & d & k \end{matrix} \right\} = \frac{\text{Tet} \left[\begin{matrix} a & b & i \\ c & d & k \end{matrix} \right] \Delta_k}{\Theta(a, b, k) \Theta(c, d, k)}$$

Fig. 7. Tetrahedron Formula for Recoupling Coefficients

$$\lambda_c^{ab} = (-1)^{(a+b-c)/2} A^{(a'+b'-c')/2}$$
$$x' = x(x+2)$$

Fig. 8. Local Braiding Formula

One sometimes writes the *quantum integer*

$$[n] = (-1)^{n-1} \Delta_{n-1} = \frac{A^{2n} - A^{-2n}}{A^2 - A^{-2}}.$$

If

$$A = e^{i\pi/2r}$$

where r is a positive integer, then

$$\Delta_n = (-1)^n \frac{\sin((n+1)\pi/r)}{\sin(\pi/r)}.$$

Here the corresponding quantum integer is

$$[n] = \frac{\sin(n\pi/r)}{\sin(\pi/r)}.$$

Note that $[n+1]$ is a positive real number for $n = 0, 1, 2, \dots, r-2$ and that $[r-1] = 0$.

The evaluation of the theta net is expressed in terms of quantum integers by the formula

$$\Theta(a, b, c) = (-1)^{m+n+p} \frac{[m+n+p+1]![n]![m]![p]!}{[m+n]![n+p]![p+m]!}$$

where

$$a = m + p, b = m + n, c = n + p.$$

Note that

$$(a + b + c)/2 = m + n + p.$$

When $A = e^{i\pi/2r}$, the recoupling theory becomes finite with the restriction that only three-vertices (labeled with a, b, c) are *admissible* when $a+b+c \leq 2r-4$. All the summations in the formulas for recoupling are restricted to admissible triples of this form.

2.2 Symmetry and Unitarity

The formula for the recoupling coefficients given in Figure 7 has less symmetry than is actually inherent in the structure of the situation. By multiplying all the vertices by an appropriate factor, we can reconfigure the formulas in this theory so that the revised recoupling transformation is orthogonal, in the sense that its transpose is equal to its inverse. This is a very useful fact. It means that when the resulting matrices are real, then the recoupling transformations are unitary.

Figure 9 illustrates this modification of the three-vertex. Let $Vert[a, b, c]$ denote the original 3-vertex of the Temperley – Lieb recoupling theory. Let $ModVert[a, b, c]$ denote the modified vertex. Then we have the formula

$$ModVert[a, b, c] = \frac{\sqrt{\sqrt{\Delta_a \Delta_b \Delta_c}}}{\sqrt{\Theta(a, b, c)}} Vert[a, b, c].$$

Lemma. For the bracket evaluation at the root of unity $A = e^{i\pi/2r}$ the factor

$$f(a, b, c) = \frac{\sqrt{\sqrt{\Delta_a \Delta_b \Delta_c}}}{\sqrt{\Theta(a, b, c)}}$$

$$\begin{array}{c} a \quad b \\ \diagdown \quad \diagup \\ \circ \\ \diagup \\ c \end{array} = \frac{\sqrt{\sqrt{\Delta_a \Delta_b \Delta_c}}}{\sqrt{\Theta(a, b, c)}} \begin{array}{c} a \quad b \\ \diagdown \quad \diagup \\ \bullet \\ \diagup \\ c \end{array}$$

Fig. 9. Modified Three Vertex

$$\begin{array}{c} a \\ | \\ \bullet \\ \circ \\ b \quad c \\ | \\ a \end{array} = \frac{\Theta(a, b, c)}{\Delta_a} \begin{array}{c} a \\ | \\ \bullet \\ | \\ a \end{array}$$

$$\begin{array}{c} a \\ | \\ \circ \\ \circ \\ b \quad c \\ | \\ a \end{array} = \frac{\sqrt{\Delta_a \Delta_b \Delta_c}}{\Theta(a, b, c)} \begin{array}{c} a \\ | \\ \bullet \\ \bullet \\ b \quad c \\ | \\ a \end{array}$$

$$\begin{array}{c} a \\ | \\ \circ \\ \bullet \\ b \quad c \\ | \\ a \end{array} = \sqrt{\frac{\Delta_b \Delta_c}{\Delta_a}} \begin{array}{c} a \\ | \\ \bullet \\ | \\ a \end{array}$$

Fig. 10. Modified Bubble Identity

is real, and can be taken to be a positive real number for (a, b, c) admissible (i.e. with $a + b + c \leq 2r - 4$).

Proof. By the results from the previous subsection,

$$\Theta(a, b, c) = (-1)^{(a+b+c)/2} \hat{\Theta}(a, b, c)$$

where $\hat{\Theta}(a, b, c)$ is positive real, and

$$\Delta_a \Delta_b \Delta_c = (-1)^{(a+b+c)} [a + 1][b + 1][c + 1]$$

where the quantum integers in this formula can be taken to be positive real. It follows from this that

$$f(a, b, c) = \sqrt{\frac{\sqrt{[a + 1][b + 1][c + 1]}}{\hat{\Theta}(a, b, c)}},$$

showing that this factor can be taken to be positive real. This completes the proof.

In Figure 10 we show how this modification of the vertex affects the non-zero term of the orthogonality of trivalent vertices (compare with Figure 4). We refer to this as the “modified bubble identity.” The coefficient in the modified bubble identity is

$$\sqrt{\frac{\Delta_b \Delta_c}{\Delta_a}} = (-1)^{(b+c-a)/2} \sqrt{\frac{[b+1][c+1]}{[a+1]}}$$

where (a, b, c) form an admissible triple. In particular $b + c - a$ is even and hence this factor can be taken to be real.

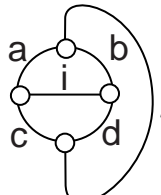
We rewrite the recoupling formula in this new basis and emphasize that the recoupling coefficients can be seen (for fixed external labels a, b, c, d) as a matrix transforming the horizontal “double-Y” basis to a vertically disposed double-Y basis. In Figures 11, 12 and 13 we have shown the form of this transformation, using the matrix notation

$$\begin{aligned} M[a, b, c, d]_{ij} &= \sum_k \begin{bmatrix} a & b \\ c & d \end{bmatrix}_{ik} \begin{bmatrix} a & b \\ c & d \end{bmatrix}_{kj} \\ &= \sum_k \begin{bmatrix} a & b \\ c & d \end{bmatrix}_{ik} \sqrt{\frac{\Delta_a \Delta_b}{\Delta_j}} \sqrt{\frac{\Delta_c \Delta_d}{\Delta_j}} \Delta_j \delta_j^k \\ &= \begin{bmatrix} a & b \\ c & d \end{bmatrix}_{ij} \sqrt{\frac{\Delta_a \Delta_b}{\Delta_j}} \sqrt{\frac{\Delta_c \Delta_d}{\Delta_j}} \Delta_j \\ &= \frac{\begin{bmatrix} a & b \\ c & d \end{bmatrix}_{ij} \sqrt{\frac{\Delta_a \Delta_b}{\Delta_j}} \sqrt{\frac{\Delta_c \Delta_d}{\Delta_j}} \Delta_j}{\sqrt{\frac{\Delta_a \Delta_b}{\Delta_j}} \sqrt{\frac{\Delta_c \Delta_d}{\Delta_j}} \Delta_j} = \frac{\text{ModTet} \left[\begin{bmatrix} a & b & i \\ c & d & j \end{bmatrix} \right]}{\sqrt{\frac{\Delta_a \Delta_b}{\Delta_j}} \sqrt{\frac{\Delta_c \Delta_d}{\Delta_j}} \Delta_j} \end{aligned}$$

Fig. 11. Derivation of Modified Recoupling Coefficients

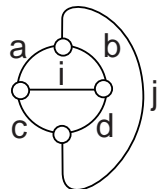
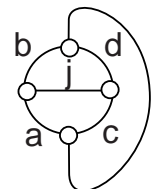
$$\begin{array}{c} a \quad b \\ \diagdown \quad \diagup \\ \bigcirc \text{---} i \text{---} \bigcirc \\ \diagup \quad \diagdown \\ c \quad d \end{array} = \sum_j \begin{bmatrix} a & b \\ c & d \end{bmatrix}_{ij} \begin{array}{c} a \quad b \\ \diagdown \quad \diagup \\ \bigcirc \\ | \quad j \\ \bigcirc \\ \diagup \quad \diagdown \\ c \quad d \end{array}$$

Fig. 12. Modified Recoupling Formula

$$\begin{array}{|c|c|} \hline a & b \\ \hline c & d \\ \hline \end{array} \Big|_{ij} = \frac{\text{Diagram}}{\sqrt{\Delta_a \Delta_b \Delta_c \Delta_d}}$$


$$M[a,b,c,d]_{ij} = \begin{array}{|c|c|} \hline a & b \\ \hline c & d \\ \hline \end{array} \Big|_{ij}$$

Fig. 13. Modified Recoupling Matrix

$$\frac{\text{Diagram 1}}{\sqrt{\Delta_a \Delta_b \Delta_c \Delta_d}} = \frac{\text{Diagram 2}}{\sqrt{\Delta_a \Delta_b \Delta_c \Delta_d}}$$



$$\Rightarrow \begin{array}{|c|c|} \hline a & b \\ \hline c & d \\ \hline \end{array}^T = \begin{array}{|c|c|} \hline a & b \\ \hline c & d \\ \hline \end{array}^{-1}$$

Fig. 14. Modified Matrix Transpose

for the modified recoupling coefficients. In Figure 11 we derive an explicit formula for these matrix elements. The proof of this formula follows directly from trivalent-vertex orthogonality (See Figures 4 and 7.), and is given in Figure 11. The result shown in Figure 11 and Figure 12 is the following formula for the recoupling matrix elements.

$$M[a,b,c,d]_{ij} = \text{ModTet} \left(\begin{array}{cc} a & b \\ c & d \end{array} \right) / \sqrt{\Delta_a \Delta_b \Delta_c \Delta_d}$$

where $\sqrt{\Delta_a \Delta_b \Delta_c \Delta_d}$ is short-hand for the product

$$\begin{aligned} & \sqrt{\frac{\Delta_a \Delta_b}{\Delta_j}} \sqrt{\frac{\Delta_c \Delta_d}{\Delta_j}} \Delta_j \\ &= (-1)^{(a+b-j)/2} (-1)^{(c+d-j)/2} (-1)^j \sqrt{\frac{[a+1][b+1]}{[j+1]}} \sqrt{\frac{[c+1][d+1]}{[j+1]}} [j+1] \\ &= (-1)^{(a+b+c+d)/2} \sqrt{[a+1][b+1][c+1][d+1]} \end{aligned}$$

In this form, since (a, b, j) and (c, d, j) are admissible triples, we see that this coefficient can be taken to be real, and its value is independent of the choice of i and j . The matrix $M[a, b, c, d]$ is real-valued.

It follows from Figure 12 (turn the diagrams by ninety degrees) that

$$M[a, b, c, d]^{-1} = M[b, d, a, c].$$

In Figure 14 we illustrate the formula

$$M[a, b, c, d]^T = M[b, d, a, c].$$

It follows from this formula that

$$M[a, b, c, d]^T = M[a, b, c, d]^{-1}.$$

Hence $M[a, b, c, d]$ is an orthogonal, real-valued matrix.

Theorem. In the Temperley – Lieb theory we obtain unitary (in fact real orthogonal) recoupling transformations when the bracket variable A has the form $A = e^{i\pi/2r}$. Thus we obtain families of unitary representations of the Artin braid group from the recoupling theory at these roots of unity.

Proof. The proof is given in the discussion above.

Remark. The density of these representations will be taken up in a subsequent paper.

3 Quantum Computation of Colored Jones Polynomials and the Witten-Reshetikhin-Turaev Invariant

In this section we make some brief comments on the quantum computation of colored Jones polynomials. This material will be expanded in a subsequent publication.

First, consider Figure 15. In that figure we illustrate the calculation of the evaluation of the (a) - colored bracket polynomial for the plat closure $P(B)$ of a braid B . The reader can infer the definition of the plat closure from Figure 15. One takes a braid on an even number of strands and closes the top strands with

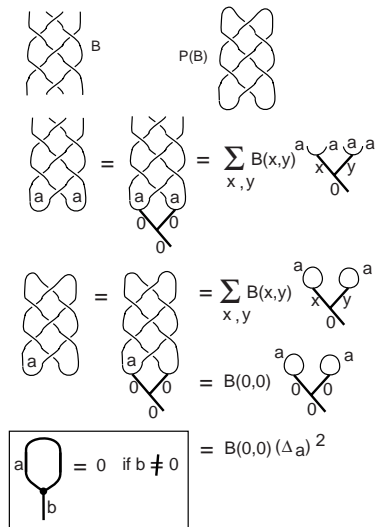


Fig. 15. Evaluation of the Plat Closure of a Braid

each other in a row of maxima. Similarly, the bottom strands are closed with a row of minima. It is not hard to see that any knot or link can be represented as the plat closure of some braid. Note that in this figure we indicate the action of the braid group on the process spaces corresponding to the small trees attached below the braids.

The (a) - colored bracket polynomial of a link L , denoted $\langle L \rangle_a$, is the evaluation of that link where each single strand has been replaced by a parallel strands and the insertion of Jones-Wenzl projector. We then see that we can use our discussion of the Temperley-Lieb recoupling theory to compute the value of the colored bracket polynomial for the plat closure PB . As shown in Figure 15, we regard the braid as acting on a process space $V_0^{a,a,\dots,a}$ and take the case of the action on the vector v whose process space coordinates are all zero. Then the action of the braid takes the form

$$Bv(0, \dots, 0) = \sum_{x_1, \dots, x_n} B(x_1, \dots, x_n) v(x_1, \dots, x_n)$$

where $B(x_1, \dots, x_n)$ denotes the matrix entries for this recoupling transformation and $v(x_1, \dots, x_n)$ runs over a basis for the space $V_0^{a,a,\dots,a}$. Here n is even and equal to the number of braid strands. In the figure we illustrate with $n = 4$. Then, as the figure shows, when we close the top of the braid action to form PB , we cut the sum down to the evaluation of just one term. In the general case we will get

$$\langle PB \rangle_a = B(0, \dots, 0) \Delta_a^{n/2}.$$

The calculation simplifies to this degree because of the vanishing of loops in the recoupling graphs.

The *colored Jones polynomials* are normalized versions of the colored bracket polynomials, differing just by a normalization factor.

In order to consider quantum computation of the colored bracket or colored Jones polynomials, we therefore can consider quantum computation of the matrix entries $B(0, \dots, 0)$. These matrix entries in the case of the roots of unity $A = e^{i\pi/2r}$ and for the $a = 2$ Fibonacci model with $A = e^{3i\pi/5}$ are parts of the diagonal entries of the unitary transformation that represents the braid group on the process space $V_0^{a,a,\dots,a}$. We can obtain these matrix entries by using the *Hadamard test*. As a result we get relatively efficient quantum algorithms for the colored Jones polynomials at these roots of unity, in essentially the same framework as we described in section 4, but for braids of arbitrary size. The computational complexity of these models is essentially the same as the models for the Jones polynomial discussed in [1].

Remark on the Hadamard Test. In order to (quantum) compute the trace of a unitary matrix U , one can use the *Hadamard test* to obtain the diagonal matrix elements $\langle \psi | U | \psi \rangle$ of U . The trace is then the sum of these matrix elements as $|\psi\rangle$ runs over an orthonormal basis for the vector space. We first obtain

$$\frac{1}{2} + \frac{1}{2} \text{Re} \langle \psi | U | \psi \rangle$$

as an expectation by applying the Hadamard gate H

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

to the first qubit of

$$C_U \circ (H \otimes 1) |0\rangle |\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle \otimes |\psi\rangle + |1\rangle \otimes U|\psi\rangle).$$

Here C_U denotes controlled U , acting as U when the control bit is $|1\rangle$ and the identity mapping when the control bit is $|0\rangle$. We measure the expectation for the first qubit $|0\rangle$ of the resulting state

$$\begin{aligned} \frac{1}{2} (H|0\rangle \otimes |\psi\rangle + H|1\rangle \otimes U|\psi\rangle) &= \frac{1}{2} ((|0\rangle + |1\rangle) \otimes |\psi\rangle + (|0\rangle - |1\rangle) \otimes U|\psi\rangle) \\ &= \frac{1}{2} (|0\rangle \otimes (|\psi\rangle + U|\psi\rangle) + |1\rangle \otimes (|\psi\rangle - U|\psi\rangle)). \end{aligned}$$

This expectation is

$$\frac{1}{2} (\langle \psi | + \langle \psi | U^\dagger) (|\psi\rangle + U|\psi\rangle) = \frac{1}{2} + \frac{1}{2} \text{Re} \langle \psi | U | \psi \rangle.$$

The imaginary part is obtained by applying the same procedure to

$$\frac{1}{\sqrt{2}} (|0\rangle \otimes |\psi\rangle - i|1\rangle \otimes U|\psi\rangle)$$

This is the method used in [1], and the reader may wish to contemplate its efficiency in the context of this model. Note that the Hadamard test enables this quantum computation to estimate the trace of any unitary matrix U by repeated trials that estimate individual matrix entries $\langle \psi | U | \psi \rangle$. In the case of the quantum computation of a colored Jones polynomial evaluation for the plat closure of a link, we need only obtain a single entry of the unitary matrix that represents the corresponding braiding transformation. Thus, if one is given the quantum computer hardware for the braiding, already provided, the actual computation is quite efficient.

Remark on the WRT invariant. It is worth remarking here that these algorithms give not only quantum algorithms for computing the colored bracket and Jones polynomials, but also for computing the Witten-Reshetikhin-Turaev (WRT) invariants at the above roots of unity. The reason for this is that the WRT invariant, in unnormalized form is given as a finite sum of colored bracket polynomials:

$$WRT(L) = \sum_{a=0}^{r-2} \Delta_a \langle L \rangle_a,$$

and so the same computation as shown in Figure 60 applies to the WRT. This means that we have, in principle, a quantum algorithm for the computation of the Witten functional integral [19] via this knot-theoretic combinatorial topology. It would be very interesting to understand a more direct approach to such a computation via quantum field theory and functional integration.

Finally, we note that in the case of the Fibonacci model, the (2)-colored bracket polynomial is a special case of the Dubrovnik version of the Kauffman polynomial [10]. See Figure 16 for diagrammatics that resolve this fact. The skein

$$\begin{aligned}
 & \text{Crossing} = A^4 \text{Parallel} + A^{-4} \text{AntiParallel} + \delta \text{Square} \\
 & \text{Crossing} = A^{-4} \text{Parallel} + A^4 \text{AntiParallel} + \delta \text{Square} \\
 & \text{Crossing} - \text{Crossing} = (A^4 - A^{-4}) (\text{Parallel} - \text{AntiParallel}) \\
 & \boxed{
 \begin{aligned}
 & \text{Crossing} - \text{Crossing} = (A^4 - A^{-4}) (\text{Parallel} - \text{AntiParallel}) \\
 & \text{Twist} = A^8 \text{Parallel}
 \end{aligned}
 }
 \end{aligned}$$

Fig. 16. Dubrovnik Polynomial Specialization at Two Strands

relation for the Dubrovnik polynomial is boxed in this figure. Above the box, we show how the double strands with projectors reproduce this relation. This observation means that in the Fibonacci model, the natural underlying knot polynomial is a special evaluation of the Dubrovnik polynomial, and the Fibonacci model can be used to perform quantum computation for the values of this invariant.

References

1. Aharonov, D., Jones, V., Landau, Z.: A polynomial quantum algorithm for approximating the Jones polynomial, quant-ph/0511096.
2. Aharonov, D., Arad, I.: The BQP-hardness of approximating the Jones polynomial, quant-ph/0605181.
3. Freedman, M.: A magnetic model with a possible Chern-Simons phase, quant-ph/0110060v1 9 October 2001 (preprint)
4. Freedman, M.: Topological Views on Computational Complexity. Documenta Mathematica - Extra Volume ICM, 453–464 (1998)
5. Freedman, M., Larsen, M., Wang, Z.: A modular functor which is universal for quantum computation, quant-ph/0001108v2 (February 1 2000)
6. Freedman, M.H., Kitaev, A., Wang, Z.: Simulation of topological field theories by quantum computers. Commun. Math. Phys. 227, 587–603 (2002) quant-ph/0001071.
7. Freedman, M.: Quantum computation and the localization of modular functors, quant-ph/0003128.
8. Jones, V.F.R.: A polynomial invariant for links via von Neumann algebras, Bull. Amer. Math. Soc. 129, 103–112 (1985)
9. Kauffman, L.H.: State models and the Jones polynomial. Topology 26, 395–407 (1987)
10. Kauffman, L.H.: An invariant of regular isotopy. Trans. Amer. Math. Soc. 318(2), 417–471 (1990)
11. Kauffman, L.H.: Temperley – Lieb Recoupling Theory and Invariants of Three-Manifolds, Princeton University Press, Annals Studies, 114 (1994)
12. Kauffman, L.H., Lomonaco Jr., S.J.: Braiding Operators are Universal Quantum Gates. New Journal of Physics 6(134), 1–39 (2004)
13. Kauffman, L.H., Lomonaco Jr., S.J.: q - deformed spin networks, knot polynomials and anyonic topological computation, quant-ph/0606114 (to appear in JKTR)
14. Kitaev, A.: Anyons in an exactly solved model and beyond, arXiv.cond-mat/0506438 v1 (17 June 2005)
15. Marzuoli, A., Rasetti, M.: Spin network quantum simulator. Physics Letters A 306, 79–87 (2002)
16. Penrose, R.: Angular momentum: An approach to Combinatorial Spacetime. In: Bastin, T. (ed.) Quantum Theory and Beyond, Cambridge University Press, Bastin (1969)
17. Preskill, J.: Topological computing for beginners, (slide presentation), Lecture Notes for Chapter 9 - Physics 219 - Quantum Computation.
<http://www.iqi.caltech.edu/preskill/ph219>
18. Wilczek, F.: Fractional Statistics and Anyon Superconductivity. World Scientific Publishing Company, Singapore (1990)
19. Witten, E.: Quantum field Theory and the Jones Polynomial. Commun. Math. Phys. 1989, 351–399 (1989)

Symmetries in Natural Language Syntax and Semantics: The Lambek-Grishin Calculus

Michael Moortgat*

Utrecht Institute of Linguistics OTS, The Netherlands
moortgat@let.uu.nl

Abstract. In this paper, we explore the Lambek-Grishin calculus **LG**: a symmetric version of categorial grammar based on the generalizations of Lambek calculus studied in Grishin [1]. The vocabulary of **LG** complements the Lambek product and its left and right residuals with a dual family of type-forming operations: coproduct, left and right difference. The two families interact by means of structure-preserving distributivity principles. We present an axiomatization of **LG** in the style of Curry’s combinatory logic and establish its decidability. We discuss Kripke models and Curry-Howard interpretation for **LG** and characterize its notion of type similarity in comparison with the other categorial systems. From the linguistic point of view, we show that **LG** naturally accommodates non-local semantic construal and displacement — phenomena that are problematic for the original Lambek calculi.

1 Background

The basic Lambek calculus [2] is a logic without *any* structural rules: grammatical material cannot be duplicated or erased without affecting well-formedness (absence of Contraction and Weakening); moreover, structural rules affecting word order and constituent structure (Commutativity and Associativity) are unavailable. What remains (in addition to the preorder axioms for derivability) is the pure logic of residuation of (1).

$$\text{RESIDUATED TRIPLE} \quad A \rightarrow C/B \quad \text{iff} \quad A \otimes B \rightarrow C \quad \text{iff} \quad B \rightarrow A \backslash C \quad (1)$$

The type-forming operations have two kinds of semantics. One is a *structural* semantics, where they are interpreted with respect to a ternary composition relation (or ‘Merge’, as it is called in generative grammar). The truth conditions for

* Linguistic exploration of Lambek-Grishin calculus started at ESSLLI’04 (Nancy) in discussion with Raffaella Bernardi and Rajeev Goré; I thank them, and Natasha Kurtonina, for the stimulating exchange of ideas throughout the period reported on here. The material of this paper was the subject of a series of lectures at Moscow State University (Sept 2006, April 2007). I thank Mati Pentus and Barbara Partee for accommodating these talks in their seminars and for providing such perceptive audiences. The support of the Dutch-Russian cooperation program NWO/RFBR, project No. 047.017.014 “Logical models of human and mechanical reasoning” is gratefully acknowledged.

this interpretation are given in (2); one finds the basic soundness/completeness results in [3]. The second interpretation is a *computational* one, along the lines of the Curry-Howard formulas-as-types program. Under this second interpretation, Lambek derivations are associated with a linear (and structure-sensitive) sublanguage of the lambda terms one obtains for proofs in positive Intuitionistic logic. The slashes $/, \backslash$ here are seen as directional implications; elimination of these operations corresponds to function application, introduction to lambda abstraction.

$$\begin{aligned} x \Vdash A \otimes B &\text{ iff } \exists yz. R_{\otimes}xyz \text{ and } y \Vdash A \text{ and } z \Vdash B \\ y \Vdash C/B &\text{ iff } \forall xz. (R_{\otimes}xyz \text{ and } z \Vdash B) \text{ implies } x \Vdash C \\ z \Vdash A \backslash C &\text{ iff } \forall xy. (R_{\otimes}xyz \text{ and } y \Vdash A) \text{ implies } x \Vdash C \end{aligned} \quad (2)$$

The original Lambek calculus, like its predecessors the Ajdukiewicz/Bar Hillel (AB) calculi, and later systems such as Combinatory Categorical Grammar (CCG), adequately deals with linguistic subcategorization or valency. It greatly improves on AB and CCG systems in fully supporting hypothetical reasoning: the bidirectional implications of the residuation laws are fully symmetric with respect to *putting together* larger phrases out of their subphrases, and *taking apart* compound phrases in their constituent parts. AB systems lack the second feature completely; the combinatory schemata of CCG provide only a weak approximation. Consequences of hypothetical reasoning are the theorems of type lifting and argument lowering of (3) below; type transitions of this kind have played an important role in our understanding of natural language semantics.

$$A \rightarrow B/(A \backslash B) \quad (B/(A \backslash B)) \backslash B \rightarrow A \backslash B \quad (3)$$

It is ironic that precisely in the hypothetical reasoning component the Lambek grammars turn out to be deficient. As one sees in (3), hypothetical reasoning typically involves higher order types, where a slash occurs in a negative environment as in the schema (4) below. Given Curry-Howard assumptions, the associated instruction for meaning assembly has an application, corresponding to the elimination of the main connective $/$, and an abstraction, corresponding to the introduction of the embedded \backslash .

$$C/(A \backslash B) \quad (M \lambda x^A. N^B)^C \quad (4)$$

The minimal Lambek calculus falls short in its characterization of which A -type hypotheses are ‘visible’ for the slash introduction rule: for the residuation rules to be applicable, the hypothesis has to be structurally *peripheral* (left peripheral for \backslash , right peripheral for $/$). One can distinguish two kinds of problems.

Displacement. The A -type hypothesis occurs *internally* within the domain of type B . Metaphorically, the functor $C/(A \backslash B)$ seems to be displaced from the site of the hypothesis. Example: *wh* ‘movement’.

Non-local semantic construal. The functor (e.g. $C/(A \backslash B)$) occupies the structural position where in fact the A -type hypothesis is needed, and realizes its semantic effect at a higher structural level. (The converse of the above.) Example: quantifying expressions.

The initial reaction to these problems has been to extend the basic system with structural rules of Associativity and/or Commutativity, resulting in the hierarchy of Fig 1. From a linguistic point of view, a *global* implementation of these structural options is undesirable, as it entails a complete loss of sensitivity for word order and/or constituent structure.

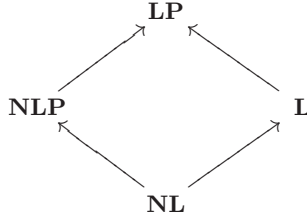


Fig. 1. The Lambek hierarchy

A significant enhancement of the Lambek systems occurred in the mid Nineties of the last century, when the vocabulary was extended with a pair of *unary* type-forming operations (Moortgat [4]). Like their binary relatives, \diamond, \square form a residuated pair, satisfying (5), and with interpretation (6).

$$\text{RESIDUATED PAIR} \quad \diamond A \rightarrow B \quad \text{iff} \quad A \rightarrow \square B \quad (5)$$

$$\begin{aligned} x \Vdash \diamond A & \text{ iff } \exists y (R_{\diamond}xy \text{ and } y \Vdash A) \\ y \Vdash \square A & \text{ iff } \forall x (R_{\diamond}xy \text{ implies } x \Vdash A) \end{aligned} \quad (6)$$

From a purely logic point of view, the unary modalities introduce facilities for *subtyping*, in the sense that any type A is now derivationally related to a more specific type $\diamond\square A$ and a more general type $\square\diamond A$.

$$\diamond\square A \rightarrow A \rightarrow \square\diamond A \quad (7)$$

Bernardi [5] makes good use of the patterns in (7) to fine-tune the construal of scope-bearing expressions, and to capture the selectional restrictions governing the distribution of polarity sensitive items. In addition to this logical use of \diamond, \square , one can also use them to provide *controlled* versions of structural rules that, in a global form, would be deleterious. The set of postulates in (8) make left branches ($P1, P2$) or right branches ($P3, P4$) accessible for hypothetical reasoning.

$$\begin{aligned} (P1) \quad \diamond A \otimes (B \otimes C) & \rightarrow (\diamond A \otimes B) \otimes C & (C \otimes B) \otimes \diamond A & \rightarrow C \otimes (B \otimes \diamond A) & (P3) \\ (P2) \quad \diamond A \otimes (B \otimes C) & \rightarrow B \otimes (\diamond A \otimes C) & (C \otimes B) \otimes \diamond A & \rightarrow (C \otimes \diamond A) \otimes B & (P4) \end{aligned} \quad (8)$$

Vermaat [6] uses these postulates in a cross-linguistic study of *wh* extraction constructions, relating the choice between $P1, P2$ and $P3, P4$ to the typological distinction between head-initial versus head-final languages.

Structural control. The general situation with respect to the expressivity of modally controlled structural rules is captured by Thm 1. We consider a source logic \mathcal{L} and a target logic, which is either an upward (\mathcal{L}^\uparrow) or a downward (\mathcal{L}^\downarrow) neighbor in the Lambek hierarchy. The target logic has control modalities \Diamond, \Box which are lacking in the source logic. In terms of these modalities, one defines translations from the formulas of the source logic to the formulas of the target logic. The \cdot^\downarrow translations impose the structure-sensitivity of the source logic in a logic with a more liberal structural regime; the \cdot^\uparrow translations recover the flexibility of an upward neighbor by adding \mathcal{R}_\Diamond — the image under \cdot^\uparrow of the structural rules that discriminate source from target. An example is the translation $(A \otimes B)^\downarrow = \Diamond(A^\downarrow \otimes B^\downarrow)$, which blocks associativity by removing the structural conditions for its application.

Theorem 1. *Structural control (Kurtonina and Moortgat [7]). For logics $\mathcal{L}, \mathcal{L}^\uparrow, \mathcal{L}^\downarrow$ and translations \cdot^\downarrow and \cdot^\uparrow as defined above,*

$$\begin{array}{lll} \mathcal{L} \vdash A \rightarrow B & \text{iff} & \mathcal{L}_\Diamond^\uparrow \vdash A^\downarrow \rightarrow B^\downarrow & \text{CONSTRAINING} \\ \mathcal{L} \vdash A \rightarrow B & \text{iff} & \mathcal{L}_\Diamond^\downarrow + \mathcal{R}_\Diamond \vdash A^\uparrow \rightarrow B^\uparrow & \text{LICENSING} \end{array}$$

Summarizing. Viewed from a foundational point of view, one can see Lambek-style categorical grammars as modal logics of natural language resources. In such logics, the vocabulary for analyzing the assembly of form and meaning consists of n -ary type-forming operations (or connectives, under Curry’s formulas-as-types view); these operations are given a Kripke-style interpretation in terms of $(n+1)$ -ary ‘merge’/composition relations. Grammatical *invariants*, in this approach, are laws that do not impose restrictions on the interpreting composition relations; language *diversity* results from the combination of the invariant base logic with a set of non-logical axioms (and the corresponding frame constraints). These axioms (possibly language-specific) characterize the structural deformations under which the basic form-meaning correspondences are preserved. The reader is referred to [8] for an overview of this line of work.

2 Lambek-Grishin Calculus

Assessing the merits of the above approach, one can identify two problematic aspects. One is of a computational nature, the other relates to the cognitive implications of the Lambek framework.

Complexity. Structural rules (whether implemented globally or under modal control) are computationally expensive. Whereas the basic Lambek calculus has a polynomial recognition problem [9], already the simplest extension with an associative regime is known to be NP complete [10]; one reaches a PSPACE upper bound for the extension with a \Diamond, \Box controlled structural module consisting of resource-respecting (i.e. linear, non-expanding) axioms [11].

Invariants versus structural postulates. On the cognitive level, the limited expressivity of the standard vocabulary means one is forced to accept that a considerable part of grammatical organization is beyond the reach of the type-forming constants. By considering a broader vocabulary of connectives it becomes possible to characterize larger portions of a language’s grammar in terms of linguistic *invariants*, rather than through non-logical postulates.

In a remarkable paper written in 1983, V.N. Grishin [1] has proposed a framework for generalizing the Lambek calculi that provides an alternative to the structural rule approach. The starting point for Grishin’s generalization is a symmetric extension of the vocabulary of type-forming operations: in addition to the familiar $\otimes, \backslash, /$ (product, left and right division), one also considers a dual family \oplus, \oslash, \ominus : coproduct, right and left difference.¹

$$\begin{array}{ll}
 A, B ::= p \mid & \text{atoms: } s \text{ sentence, } np \text{ noun phrases, } \dots \\
 A \otimes B \mid B \backslash A \mid A / B \mid & \text{product, left vs right division} \\
 A \oplus B \mid A \oslash B \mid B \ominus A & \text{coproduct, right vs left difference}
 \end{array} \tag{9}$$

We saw that algebraically, the Lambek operators form a residuated triple; likewise, the \oplus family forms a dual residuated triple.

$$\begin{array}{llll}
 \text{RESIDUATED TRIPLE} & A \rightarrow C / B & \text{iff} & A \otimes B \rightarrow C & \text{iff} & B \rightarrow A \backslash C \\
 \text{DUAL RESIDUATED TRIPLE} & C \oslash B \rightarrow A & \text{iff} & C \rightarrow A \oplus B & \text{iff} & A \oslash C \rightarrow B
 \end{array} \tag{10}$$

Dunn’s [12] framework of gaggle theory brings out the underlying algebraic structure in a particularly clear way. In Fig 2, we consider ordered sets $(X, \leq), (Y, \leq')$ with mappings $f : X \longrightarrow Y, g : Y \longrightarrow X$. In the categorial setting, we have $X = Y = \mathcal{F}$ (the set of types/formulas). The pair of operations (f, g) is called residuated if it satisfies the defining biconditionals of the upper left cell; the lower right cell characterizes dual residuated pairs. Whereas the concept of residuation pairs the (co)product with a (co)implication, the closely related concept of (dual) Galois connected pairs links the (co)implications among themselves. The defining biconditionals fill the lower left and upper right cells. For $* \in \{/, \otimes, \backslash, \oslash, \oplus, \ominus\}$, we write $-*$ ($*-$) for the operation that suffixes (prefixes) a fixed type to its operand, for example: $A \rightarrow C / B$ iff $B \rightarrow A \backslash C$ instantiates the pattern $(/ -, - \backslash)$.

iff	$x \leq gy$	$gy \leq x$
$fx \leq' y$	$(-\otimes, -/)$ $(\otimes -, \backslash -)$	$(-\oslash, \oslash -)$ $(\oslash -, -\ominus)$
$y \leq' fx$	$(-\backslash, /-)$ $(/-, -\backslash)$	$(-\oplus, -\oslash)$ $(\oplus -, \ominus -)$

Fig. 2. Residuated and Galois connected pairs and their duals

¹ A little pronunciation dictionary: read $B \backslash A$ as ‘ B under A ’, A / B as ‘ A over B ’, $B \oslash A$ as ‘ B from A ’ and $A \ominus B$ as ‘ A less B ’.

The patterns of Fig 2 reveal that on the level of types and derivability the Lambek-Grishin system exhibits two kinds of mirror symmetry characterized by the bidirectional translation tables in (11): \bowtie is order-preserving, ∞ order-reversing: $A^{\bowtie} \rightarrow B^{\bowtie}$ iff $A \rightarrow B$ iff $B^{\infty} \rightarrow A^{\infty}$.

$$\bowtie \frac{C/D \quad A \otimes B \quad B \oplus A \quad D \oslash C}{D \setminus C \quad B \otimes A \quad A \oplus B \quad C \oslash D} \quad \infty \frac{C/B \quad A \otimes B \quad A \setminus C}{B \oslash C \quad B \oplus A \quad C \oslash A} \quad (11)$$

Interaction principles. The minimal symmetric categorial grammar (which we will refer to as \mathbf{LG}_\emptyset) is given by the preorder axioms for the derivability relation, together with the residuation and dual residuation principles of (10). \mathbf{LG}_\emptyset by itself does not offer us the kind of expressivity needed to address the problems discussed in §1. The real attraction of Grishin's work derives from the *interaction principles* he proposes for structure-preserving communication between the \otimes and the \oplus families. In all, the type system allows eight such principles, configured in two groups of four. Consider first the group in (12) which we will collectively refer to as \mathcal{G}^\uparrow .²

$$\begin{aligned} (G1) \quad & (A \otimes B) \otimes C \rightarrow A \otimes (B \otimes C) & C \otimes (B \otimes A) \rightarrow (C \otimes B) \otimes A & (G3) \\ (G2) \quad & C \otimes (A \otimes B) \rightarrow A \otimes (C \otimes B) & (B \otimes A) \otimes C \rightarrow (B \otimes C) \otimes A & (G4) \end{aligned} \quad (12)$$

On the lefthand side of the derivability arrow, one finds a \otimes formula which has a formula with the difference operation ($A \otimes B$ or $B \otimes A$) in its first or second coordinate. The Grishin principles rewrite this configuration in such a way that the difference operations \otimes , \oslash become the main connective. Combined with transitivity, the principles take the form of the inference rules in (13).

$$\begin{aligned} & \frac{A \otimes (B \otimes C) \rightarrow D}{(A \otimes B) \otimes C \rightarrow D} G1 & \frac{(A \otimes B) \otimes C \rightarrow D}{A \otimes (B \otimes C) \rightarrow D} G3 \\ & \frac{B \otimes (A \otimes C) \rightarrow D}{A \otimes (B \otimes C) \rightarrow D} G2 & \frac{(A \otimes C) \otimes B \rightarrow D}{(A \otimes B) \otimes C \rightarrow D} G4 \end{aligned} \quad (13)$$

These rules, from a backward-chaining perspective, have the effect of bringing the A subformula to a position where it can be shifted to the righthand side of \rightarrow by means of the dual residuation principles. The images of (12) under \cdot^∞ are given in (14): in rule form, they rewrite a configuration where a left or right slash is trapped within a \oplus context into a configuration where the A subformula can be shifted to the lefthand side of \rightarrow by means of the residuation principles. One easily checks that the forms in (14) are derivable from (12) — the derivation of $G1'$ from $G1$ is given as an example in (19).

$$\begin{aligned} (G1') \quad & (C \oplus B)/A \rightarrow C \oplus (B/A) & A \setminus (B \oplus C) \rightarrow (A \setminus B) \oplus C & (G3') \\ (G2') \quad & (B \oplus C)/A \rightarrow (B/A) \oplus C & A \setminus (C \oplus B) \rightarrow C \oplus (A \setminus B) & (G4') \end{aligned} \quad (14)$$

² In Grishin's original paper, only one representative of each group is discussed; computation of the remaining three is left to the reader. Earlier presentations of [1] such as [13,14] omit $G2$ and $G4$. The full set of (12) is essential for the intended linguistic applications.

An alternative direction for generalizing the Lambek calculus is given by the *converses* of the \mathcal{G}^\dagger principles obtained by turning around the derivability arrow. We refer to these individually as Gn^{-1} , and to the group as \mathcal{G}^\downarrow . The general picture that emerges is a landscape where the minimal symmetric Lambek calculus \mathbf{LG}_\emptyset can be extended either with $G1$ – $G4$ or with their converses, or with the combination of the two. We discuss potential linguistic applications for \mathcal{G}^\dagger and \mathcal{G}^\downarrow in §3. First, we review some prooftheoretic and modeltheoretic results relating the Lambek-Grishin calculus to the original Lambek systems. These results have focused on the combination $\mathbf{LG}_\emptyset + \mathcal{G}^\dagger$, which in the remainder we will refer to simply as \mathbf{LG} .

2.1 Decidable Proof Search

The axiomatization we have considered so far contains the rule of transitivity (from $A \rightarrow B$ and $B \rightarrow C$ conclude $A \rightarrow C$) which, in the presence of complexity-increasing type transitions, is an unpleasant rule from a proof search perspective. For decidable proof search, we are interested in an axiomatization which has transitivity as an admissible rule. Such an axiomatization for \mathbf{LG} can be given in terms of the identity axiom $1_A : A \rightarrow A$ together with the residuation principles (10), the Grishin axioms in rule form (16), and the monotonicity rules of (17).³ We give these rules with combinator proof terms, so that in the remainder we can succinctly refer to derivations by their combinator. First the residuation rules of (15).

$$\begin{array}{c} \frac{f : A \otimes B \rightarrow C}{\triangleleft f : B \rightarrow A \setminus C} \quad \frac{f : C \rightarrow A \oplus B}{\blacktriangleleft f : A \otimes C \rightarrow B} \\[1em] \frac{f : A \otimes B \rightarrow C}{\triangleright f : A \rightarrow C / B} \quad \frac{f : C \rightarrow A \oplus B}{\blacktriangleright f : C \otimes B \rightarrow A} \end{array} \quad (15)$$

These rules are invertible; we write $\triangleleft', \triangleright', \blacktriangleleft', \blacktriangleright'$ for the reverse direction. Next the Grishin axioms in rule form, for use in the lhs of derivability statements.

$$\begin{array}{c} \frac{f : A \otimes (B \otimes C) \rightarrow D}{\otimes f : (A \otimes B) \otimes C \rightarrow D} \quad \frac{f : (A \otimes B) \otimes C \rightarrow D}{\otimes f : A \otimes (B \otimes C) \rightarrow D} \\[1em] \frac{f : B \otimes (A \otimes C) \rightarrow D}{\otimes^* f : A \otimes (B \otimes C) \rightarrow D} \quad \frac{f : (A \otimes C) \otimes B \rightarrow D}{\otimes^* f : (A \otimes B) \otimes C \rightarrow D} \end{array} \quad (16)$$

Finally, (17) gives the monotonicity rules. As is well-known, the monotonicity rules are *derivable* rules of inference in an axiomatization with residuation and transitivity (cut). The purpose of the present axiomatization is to show that the

³ The axiomatization presented here is a close relative of Display Logic, see [14] for a comprehensive view on the substructural landscape. In Display Logic, the Grishin rules and residuation principles are expressed at the *structural* level; structural connectives are introduced by explicit rewriting steps. Our combinator presentation is entirely formula-based, i.e. the distinction between a ‘logical’ and a ‘structural’ occurrence of a type-forming operation is implicit.

combination monotonicity plus residuation effectively *absorbs* cut. Admissibility of cut is established in Appendix A, extending the earlier result of [15] to the case of symmetric **LG**.

$$\begin{array}{c}
\frac{f : A \rightarrow B \quad g : C \rightarrow D}{f \otimes g : A \otimes C \rightarrow B \otimes D} \quad \frac{f : A \rightarrow B \quad g : C \rightarrow D}{f \oplus g : A \oplus C \rightarrow B \oplus D} \\
\frac{f : A \rightarrow B \quad g : C \rightarrow D}{f / g : A / D \rightarrow B / C} \quad \frac{f : A \rightarrow B \quad g : C \rightarrow D}{f \oslash g : A \oslash D \rightarrow B \oslash C} \\
\frac{f : A \rightarrow B \quad g : C \rightarrow D}{g \backslash f : D \backslash A \rightarrow C \backslash B} \quad \frac{f : A \rightarrow B \quad g : C \rightarrow D}{g \oslash f : D \oslash A \rightarrow C \oslash B}
\end{array} \quad (17)$$

The symmetries we have studied before on the level of types and theorems now manifest themselves on the level of proofs.

$$\infty \frac{h/g \quad f \otimes g \quad f \backslash h}{g \oslash h \quad g \oplus f \quad h \oslash f} \infty \frac{\triangleleft f \quad \triangleright f \quad \triangleleft' f \quad \triangleright' f}{\blacktriangleright f \quad \blacktriangleleft f \quad \blacktriangleright' f \quad \blacktriangleleft' f} \quad (18)$$

As an example, consider (19). On the left, we derive $G1' = G1^\infty$ from $G1$; on the right we derive $G1$ from $G1'$. Notice that these derivations provide the motivation for our choice to posit ∞ as the basic order-reversing duality, rather than \natural which we define as $\bowtie \infty$.

$$\begin{array}{c}
\frac{(c \oplus b)/a \rightarrow (c \oplus b)/a}{((c \oplus b)/a) \otimes a \rightarrow c \oplus b} \triangleright' \quad \frac{a \oslash (b \otimes c) \rightarrow a \oslash (b \otimes c)}{b \otimes c \rightarrow a \oplus ((a \oslash (b \otimes c)))} \blacktriangleleft' \\
\frac{c \oslash (((c \oplus b)/a) \otimes a) \rightarrow b}{(c \oslash ((c \oplus b)/a)) \otimes a \rightarrow b} \blacktriangleleft \quad \frac{b \rightarrow (a \oplus ((a \oslash (b \otimes c))))/c}{b \rightarrow a \oplus ((a \oslash (b \otimes c))/c)} \triangleright \\
\frac{(c \oslash ((c \oplus b)/a)) \otimes a \rightarrow b}{c \oslash ((c \oplus b)/a) \rightarrow b/a} \triangleright \quad \frac{b \rightarrow a \oplus ((a \oslash (b \otimes c))/c)}{a \oslash b \rightarrow (a \oslash (b \otimes c))/c} \blacktriangleleft \\
\frac{c \oslash ((c \oplus b)/a) \rightarrow b/a}{(c \oplus b)/a \rightarrow c \oplus (b/a)} \blacktriangleleft' \quad \frac{a \oslash b \rightarrow (a \oslash (b \otimes c))/c}{(a \oslash b) \otimes c \rightarrow a \oslash (b \otimes c)} \triangleright'
\end{array} \quad (19)$$

$$(\blacktriangleleft' \triangleright \otimes \blacktriangleleft \triangleright' 1_{(c \oplus b)/a})^\infty = \triangleright' \blacktriangleleft \otimes^\infty \triangleright \blacktriangleleft' 1_{a \oslash (b \otimes c)}$$

We close this section with an open question on complexity. The minimal symmetric system **LG**₀ is proved to preserve the polynomiality of the asymmetric **NL** in [16]. Capelletti [17] provides a constructive polynomial algorithm for a combinator-style axiomatization of **LG**₀ that allows agenda-driven chart-based parsing. Whether the methods developed in [17] can be extended to include the Grishin interaction principles remains to be investigated.

2.2 The Group of Grishin Interactions

The \bowtie and ∞ dualities deserve closer scrutiny: as a matter of fact, they hide some interesting grouptheoretic structure.⁴ First of all, from \bowtie and the identity

⁴ This section reports on work in progress with Lutz Strassburger (Ecole Polytechnique, Paris) and with Anna Chernilovskaya (Moscow State University).

transformation 1 we obtain two further order-preserving symmetries \sharp and \flat : on the \otimes family \sharp acts like \bowtie , on the \oplus family it is the identity; \flat acts like \bowtie on the \oplus family, and as the identity on the \otimes family.

$$\begin{array}{c|cc} & \sharp & \flat \\ \hline \text{Frm}(/, \otimes, \backslash) & \bowtie & 1 \\ \text{Frm}(\odot, \oplus, \oslash) & 1 & \bowtie \end{array} \quad (20)$$

One easily checks that the Grishin postulates (12) are related horizontally by \bowtie , vertically by \sharp and diagonally by \flat . Together with the identity transformation, \bowtie , \sharp and \flat constitute D_2 — the dihedral group of order 4 (also known as the Klein group). This is the smallest non-cyclic abelian group. Its Cayley table is given in (21) below.

$$\begin{array}{c|cccc} \circ & 1 & \bowtie & \sharp & \flat \\ \hline 1 & 1 & \bowtie & \sharp & \flat \\ \bowtie & \bowtie & 1 & \flat & \sharp \\ \sharp & \sharp & \flat & 1 & \bowtie \\ \flat & \flat & \sharp & \bowtie & 1 \end{array} \quad (21)$$

Similarly, from ∞ we obtain three further order-reversing symmetries: $\natural = \bowtie \infty$, and the mixed forms \dagger and \ddagger in (22) below.

$$\begin{array}{c|cc} & \dagger & \ddagger \\ \hline \text{Frm}(/, \otimes, \backslash) & \natural & \infty \\ \text{Frm}(\odot, \oplus, \oslash) & \infty & \natural \end{array} \quad (22)$$

Together with the order-preserving transformations $\{1, \bowtie, \sharp, \flat\}$, the set $\{\infty, \natural, \dagger, \ddagger\}$ constitutes a group of order 8. We can now consider a *cube* of Grishin interactions.

$$\begin{array}{ccccc} & G1' & \text{---} & G3' & \\ & \diagdown & & \diagup & \\ G2' & \text{---} & G4' & & \infty \\ & \diagup & & \diagdown & \\ & G1 & \text{---} & G3 & \\ & \diagdown & & \diagup & \\ G2 & \text{---} & G4 & & \sharp \end{array} \quad (23)$$

The lower plane is the square (12) which we saw is characterized by D_2 . The transformation ∞ reflects (12) to the upper plane (14). The remaining transformations connect vertices of the lower plane to those of the upper plane via the diagonals. It will not come as a surprise that we have D_4 , which contains $2 \times D_2$ as subgroups: $\{1, \bowtie, \sharp, \flat\}$ and $\{1, \infty, \natural, \bowtie\}$. Notice that D_4 , unlike its D_2 subgroups, is not abelian. Consider for example $\dagger \infty = \flat \neq \sharp = \infty \dagger$. The composition $\dagger \infty$ relates $G1$ to $G4$ via $G1'$; the composition $\infty \dagger$ brings us from $G1$ to $G2$ via $G2'$.

2.3 Type Similarity

The notion of type similarity is a powerful tool to study the expressivity of categorial logics with respect to derivational relationships. The similarity

relation \sim is introduced in the algebraic remarks at the end of [18] as the reflexive, transitive, symmetric closure of the derivability relation: $A \sim B$ iff there exists a sequence $C_1 \dots C_n$ ($1 \leq n$) such that $C_1 = A$, $C_n = B$ and $C_i \rightarrow C_{i+1}$ or $C_{i+1} \rightarrow C_i$ ($1 \leq i < n$). Lambek proves that $A \sim B$ if and only if one of the following equivalent statements hold (the so-called diamond property): (i) $\exists C$ such that $A \rightarrow C$ and $B \rightarrow C$; (ii) $\exists D$ such that $D \rightarrow A$ and $D \rightarrow B$. In other words, given a join type C for A and B , one can compute a meet type D , and vice versa. The solutions for D and C in [18] are given in (24). It is shown in [19] that these solutions are in fact adequate for the pure logic of residuation, i.e. the non-associative calculus **NL**.

$$\mathbf{NL} : D = (A / ((C / C) \backslash C)) \otimes ((C / C) \backslash B), \quad C = (A \otimes (D \backslash D)) / (B \backslash (D \otimes (D \backslash D))) \quad (24)$$

For associative **L**, [20] has the shorter solution in (25). The possibility of re-bracketing the types for D and C is what makes this solution work. In **LG** also a length 5 solution is available, this time dependent on the Grishin interaction principles, see (26).

$$\mathbf{L} : D = (A / C) \otimes (C \otimes (C \backslash B)), \quad C = (D / A) \backslash (D / (B \backslash D)) \quad (25)$$

$$\mathbf{LG} : D = (A / C) \otimes (C \odot (B \odot C)), \quad C = (A \odot D) \oplus (D / (B \backslash D)) \quad (26)$$

Table 1. Models for type equivalence

CALCULUS	INTERPRETATION
NL	free quasigroup (Foret [19])
L	free group (Pentus [20])
LP	free Abelian group (Pentus [20])
LG	free Abelian group (Moortgat and Pentus [21])

The similarity relation for various calculi in the categorial hierarchy has been characterized in terms of an algebraic interpretation of the types $\llbracket \cdot \rrbracket$, in the sense that $A \sim B$ iff $\llbracket A \rrbracket =_{\mathcal{S}} \llbracket B \rrbracket$ in the relevant algebraic structures \mathcal{S} . Table 1 gives an overview of the results. For the pure residuation logic **NL**, \mathcal{S} is the free quasigroup generated by the atomic types, with $\llbracket \cdot \rrbracket$ defined in the obvious way: $\llbracket p \rrbracket = p$, $\llbracket A / B \rrbracket = \llbracket A \rrbracket / \llbracket B \rrbracket$, $\llbracket B \backslash A \rrbracket = \llbracket B \rrbracket \backslash \llbracket A \rrbracket$, $\llbracket A \otimes B \rrbracket = \llbracket A \rrbracket \cdot \llbracket B \rrbracket$.⁵ In associative **L**, type similarity coincides with equality in the free group generated by the atomic types (free Abelian group for associative/commutative **LP**). The group interpretation is (27).

$$\llbracket p \rrbracket = p, \llbracket A \otimes B \rrbracket = \llbracket A \rrbracket \cdot \llbracket B \rrbracket, \llbracket A / B \rrbracket = \llbracket A \rrbracket \cdot \llbracket B \rrbracket^{-1}, \llbracket B \backslash A \rrbracket = \llbracket B \rrbracket^{-1} \cdot \llbracket A \rrbracket \quad (27)$$

We see in Table 1 that for the systems in the Lambek hierarchy, expressivity for \sim is inversely proportional to structural discrimination: the structural rules

⁵ Recall that a quasigroups is a set equipped with operations $/, \cdot, \backslash$ satisfying the equations $(x / y) \cdot y = x$, $y \cdot (y \backslash x) = x$, $(x \cdot y) / y = x$, $y \backslash (y \cdot x) = x$.

of associativity and commutativity destroy sensitivity for constituent structure and word order. The result below shows that **LG** achieves the same level of expressivity with respect to \sim as the associative/commutative calculus **LP** and does so *without* inducing loss of structural discrimination.

Theorem 2. (Moortgat and Pentus [21]) In **LG** $A \sim B$ iff $\llbracket A \rrbracket = \llbracket B \rrbracket$ in the free Abelian group generated by $\mathbf{Atm} \cup \{\star\}$.

The group interpretation proposed in [21] for **LG** is a variant of that in (27); for **LG** one adds an extra element \star to keep track of the *operator count*, defined as in (28). It is well known that Abelian group equality can be expressed in terms of a balancing count of (input/output) occurrences of literals. The operator count, together with the count of literals, is sufficient to characterize **LG** type similarity. It is in fact enough to keep track of only one of the operator counts since the equalities $|A|_{\otimes} = |B|_{\otimes}$ and $|A|_{\oplus} = |B|_{\oplus}$ are equivalent provided that $|A|_p = |B|_p$ for all p . The equality $\sum_p |A|_p - |A|_{\otimes} - |A|_{\oplus} = 1$ has an easy proof by induction on the structure of A .

$$\begin{aligned}
 |p|_{\otimes} &= |p|_{\oplus} = 0 \\
 \begin{array}{ll}
 |A \otimes B|_{\otimes} &= |A|_{\otimes} + |B|_{\otimes} + 1 & |A \otimes B|_{\oplus} &= |A|_{\oplus} + |B|_{\oplus} \\
 |A \oplus B|_{\otimes} &= |A|_{\otimes} + |B|_{\otimes} & |A \oplus B|_{\oplus} &= |A|_{\oplus} + |B|_{\oplus} + 1 \\
 |A / B|_{\otimes} &= |A|_{\otimes} - |B|_{\otimes} - 1 & |A / B|_{\oplus} &= |A|_{\oplus} - |B|_{\oplus} \\
 |B \setminus A|_{\otimes} &= |A|_{\otimes} - |B|_{\otimes} - 1 & |B \setminus A|_{\oplus} &= |A|_{\oplus} - |B|_{\oplus} \\
 |A \odot B|_{\otimes} &= |A|_{\otimes} - |B|_{\otimes} & |A \odot B|_{\oplus} &= |A|_{\oplus} - |B|_{\oplus} - 1 \\
 |B \oslash A|_{\otimes} &= |A|_{\otimes} - |B|_{\otimes} & |B \oslash A|_{\oplus} &= |A|_{\oplus} - |B|_{\oplus} - 1
 \end{array}
 \end{array} \quad (28)$$

Some examples: operator balance fails for a/b vs $a \odot b$ (these formulas have balancing literal counts), and holds for pairs of formulas such as $(b \odot c) \oslash a$, $c/(a \setminus b)$ and a/b , $(a \odot c)/(b \odot c)$ — pairs which are indeed in the \sim relation.

We will discuss a possible use of \sim in linguistic analysis in §3. Below we reproduce a step in the proof of Thm 2 which highlights the fact that **LG** has the kind of expressivity we expect for **LP**.

Claim. For arbitrary **LG** types A, B we have $B \setminus A \sim A/B$. To prove this claim, we provide a meet type, i.e. a type X such that $X \rightarrow B \setminus A$ and $X \rightarrow A/B$, which by residuation means

$$B \otimes X \rightarrow A \quad \text{and} \quad X \otimes B \rightarrow A$$

Let us put $X := Y \odot Z$ and solve for

$$B \otimes (Y \odot Z) \rightarrow A \quad \text{and} \quad (Y \odot Z) \otimes B \rightarrow A$$

which by Grishin mixed associativity or commutativity follows from

$$B \otimes Y \rightarrow A \oplus Z \quad \text{and} \quad Y \otimes B \rightarrow A \oplus Z$$

We have a solution with $Z := (A \odot B)$ and Y the meet for C the join of $B \setminus B$ and B/B , i.e.

$$C := ((b \setminus ((b \otimes b) \odot b)) \oplus (b/b)) \quad \text{and} \quad Y := ((b/b)/C) \otimes (C \odot ((b \setminus b) \oslash C)).$$

2.4 Relational Semantics

Let us turn now to the frame semantics for **LG**. We have seen in (2) that from the modal logic perspective, the binary type-forming operation \otimes is interpreted as an existential modality with ternary accessibility relation R_\otimes . The residual $/$ and \backslash operations are the corresponding universal modalities for the rotations of R_\otimes . For the coproduct \oplus and its residuals, the dual situation obtains: \oplus here is the universal modality interpreted w.r.t. an accessibility relation R_\oplus ; the coimplications are the existential modalities for the rotations of R_\oplus . Notice that, in the minimal symmetric logic, R_\oplus and R_\otimes are *distinct* accessibility relations. Frame constraints corresponding to the Grishin interaction postulates will determine how their interpretation is related.

$$\begin{aligned} x \Vdash A \oplus B &\text{ iff } \forall yz. R_\oplus xyz \text{ implies } (y \Vdash A \text{ or } z \Vdash B) \\ y \Vdash C \oslash B &\text{ iff } \exists xz. R_\oplus xyz \text{ and } z \nVdash B \text{ and } x \Vdash C \\ z \Vdash A \oslash C &\text{ iff } \exists xy. R_\oplus xyz \text{ and } y \nVdash A \text{ and } x \Vdash C \end{aligned} \quad (29)$$

Completeness for **NL** and its extension with \Diamond, \Box can be established on the basis of a canonical model construction where the worlds are simply formulas from $\text{Frm}(/, \otimes, \backslash, \Diamond, \Box)$. For systems with richer vocabulary, Kurtonina [22] unfolds a systematic approach towards completeness in terms of *filter*-based canonical models. In the absence of the lattice operations⁶, for **LG** we can do with the simplest filter-based construction, which construes the worlds as *weak filters*, i.e. sets of formulas closed under derivability. Let us write \mathcal{F}_\uparrow for the set of filters over the **LG** formula language $\text{Frm}(/, \otimes, \backslash, \oslash, \oplus, \odot)$. The set of filters \mathcal{F}_\uparrow is closed under the operations $\cdot \hat{\otimes} \cdot, \cdot \hat{\oslash} \cdot$ defined in (30).

$$\begin{aligned} X \hat{\otimes} Y &= \{C \mid \exists A, B (A \in X \text{ and } B \in Y \text{ and } A \otimes B \rightarrow C)\} \\ X \hat{\oslash} Y &= \{B \mid \exists A, C (A \notin X \text{ and } C \in Y \text{ and } A \oslash C \rightarrow B)\}, \text{ alternatively} \\ &= \{B \mid \exists A, C (A \notin X \text{ and } C \in Y \text{ and } C \rightarrow A \oplus B)\} \end{aligned} \quad (30)$$

To lift the type-forming operations to the corresponding operations in \mathcal{F}_\uparrow , let $\lfloor A \rfloor$ be the principal filter generated by A , i.e. $\lfloor A \rfloor = \{B \mid A \rightarrow B\}$ and $\lceil A \rceil$ its principal ideal, i.e. $\lceil A \rceil = \{B \mid B \rightarrow A\}$. Writing X^\sim for the complement of X , we have

$$(\dagger) \quad \lfloor A \otimes B \rfloor = \lfloor A \rfloor \hat{\otimes} \lfloor B \rfloor \quad (\ddagger) \quad \lfloor A \oslash C \rfloor = \lceil A \rceil^\sim \hat{\oslash} \lceil C \rceil \quad (31)$$

The equations of (31) can then be used to prove the usual truth lemma that for any formula $A \in \mathcal{F}$ and filter $X \in \mathcal{F}_\uparrow$, $X \Vdash A$ iff $A \in X$. The proof is by induction on the complexity of A . The base case is handled by the canonical valuation V^c in the model below.

⁶ Allwein and Dunn [23] develop a richer theory of Kripke models for a hierarchy of substructural logics, accommodating both the lattice operations and (co)product and (co)implications.

Canonical model. Consider $\mathcal{M}^c = \langle W^c, R_{\otimes}^c, R_{\oplus}^c, V^c \rangle$ with

$$\begin{aligned} W^c &= \mathcal{F}_{\uparrow} \\ R_{\otimes}^c XYZ &\text{ iff } Y \hat{\otimes} Z \subseteq X \\ R_{\oplus}^c XYZ &\text{ iff } Y \hat{\odot} X \subseteq Z \\ V^c(p) &= \{X \in W^c \mid p \in X\} \end{aligned}$$

Theorem 3. *Soundness and completeness (Kurtonina and Moortgat [24]).*

$$\mathbf{LG}_{\emptyset} \vdash A \rightarrow B \text{ iff } \models A \rightarrow B$$

For \mathbf{LG}_{\emptyset} extended with the Grishin interaction principles, one imposes the frame constraints corresponding to the set of postulates one wishes to adopt (\mathcal{G}^{\uparrow} , \mathcal{G}^{\downarrow} or both), and one shows that in the canonical model these constraints are satisfied. For example, for (G1) we have the constraint in (32) (where $R^{(-2)}xyz = Rzyx$).

$$\forall xyzwv \ (R_{\otimes}xyz \wedge R_{\oplus}^{(-2)}yvw) \Rightarrow \exists t (R_{\oplus}^{(-2)}xwt \wedge R_{\otimes}tvz) \quad (32)$$

Observe that the construction employed is neutral with respect to the direction of the Grishin interaction principles: it accommodates G1–G4 and the converse G1^{−1}–G4^{−1} in an entirely similar way. Further research would have to show whether more concrete models exist with a bias towards either G1–G4 or the converse principles, and whether one could relate these models to the distinction between ‘overt’ and ‘covert’ forms of displacement which we illustrate in §3.

2.5 Computational Semantics

The second type of semantics we want to consider is the Curry-Howard interpretation of \mathbf{LG} derivations. In Bernardi and Moortgat [25], one finds a continuation semantics based on (a directional refinement of) Curien and Herbelin’s [26] work on the $\lambda\mu$ calculus. The source language for this interpretation is a term language coding \mathbf{LG} sequent proofs. The sequent presentation of \mathbf{LG} presented in [25] essentially relies on the Cut rule, which cannot be eliminated without losing completeness. In the present section, we give a semantics in the continuation-passing style (CPS) for the axiomatization of §2.1 which, as we have demonstrated, has an admissible cut rule.

Functions, under the CPS interpretation, rather than simply returning a value as in a direct interpretation, are provided with an extra argument for the continuation of the computation. We distinguish a call-by-value (cbv) and a call-by-name (cbn) interpretation regime. On the type level, the call-by-value transformation $[\cdot]$ is defined as in (33), where R is the distinguished type of responses. For p atomic, $[p] = p$. Notice that the $[\cdot]$ translations for the (co)implications are related vertically by left/right symmetry \cdot^{\bowtie} and in the horizontal dimension by arrow reversal \cdot^{∞} . Under cbv, for every type A of the source language, one has values $[A]$, continuations (functions from values into R) and computations (functions from continuations into R).

$$\begin{aligned} [A \setminus B] &= R^{[A] \times R^{[B]}} & [B \odot A] &= [B] \times R^{[A]}; \\ [B / A] &= R^{R^{[B]} \times [A]} & [A \odot B] &= R^{[A]} \times [B]. \end{aligned} \quad (33)$$

The call-by-name interpretation $[\cdot]$ is obtained by duality: $[A] \triangleq [A^\infty]$. Under cbn, for every type A of the source language, we have continuations $[A]$ and computations (functions from continuations into R).

Let us turn then to the interpretation of proofs, i.e. arrows $f : A \rightarrow B$. Assuming countably infinite sets of variables x_i and covariables α_i , we inductively associate each arrow $f : A \rightarrow B$ with two closed terms, f^\triangleright and f^\triangleleft , depending on whether we focus on A or B as the active formula. The induction is set up in such a way that we obtain the mappings of (34).

$$[f^\triangleright] : [A] \mapsto R^{R^{[B]}} \quad \text{and} \quad [f^\triangleleft] : R^{[B]} \mapsto R^{[A]} \quad (34)$$

i.e. call-by-value $[f^\triangleright]$ maps from A values to B computations; $[f^\triangleleft]$ from B continuations to A continuations; call-by-name is dual, with $(f^\triangleright)^\infty = (f^\infty)^\triangleleft$, $(f^\triangleleft)^\infty = (f^\infty)^\triangleright$: $[f^\triangleright] = [f^\triangleleft]$ maps B computations to A computations. The basis of the induction is given by the interpretation of the identity arrow $1_A : A \rightarrow A$ in (35).⁷ For the recursive cases, we define either the f^\triangleright or the f^\triangleleft version. The missing case is obtained by swapping the lhs and rhs arguments.

$$[(1_A)^\triangleright] x = \lambda k.(k x) \quad [(1_A)^\triangleleft] \alpha = \lambda x.(\alpha x) \quad (35)$$

Monotonicity. Given $f : A \rightarrow B$ and $g : C \rightarrow D$ we have the monotonicity rules of (36). We use curly brackets to distinguish the meta-application of the function definition from the actual target language lambda term computed.

$$\begin{aligned} [(g \setminus f)^\triangleright] y &= \lambda k.(k \lambda \langle x, \beta \rangle.(\{[g^\triangleright] x\} \lambda m.(y \langle m, \{[f^\triangleleft] \beta\} \rangle))) \\ [(f/g)^\triangleright] x &= \lambda k.(k \lambda \langle \alpha, y \rangle.(\{[g^\triangleright] y\} \lambda m.(x \langle \{[f^\triangleleft] \alpha\}, m \rangle))) \\ [(f \otimes g)^\triangleleft] \alpha &= \lambda \langle x, \delta \rangle.(\{[f^\triangleright] x\} \lambda y.(\alpha \langle y, \{[g^\triangleleft] \delta\} \rangle)) \\ [(g \otimes f)^\triangleleft] \beta &= \lambda \langle \delta, x \rangle.(\{[f^\triangleright] x\} \lambda y.(\beta \langle \{[g^\triangleleft] \delta\}, y \rangle)) \end{aligned} \quad (36)$$

Residuation. For the typing of the arrows f , see (15).

$$\begin{aligned} [(\triangleleft f)^\triangleright] y &= \lambda k.(k \lambda \langle x, \gamma \rangle.(\{[f^\triangleright] \langle x, y \rangle\} \gamma)) & [(\blacktriangleright f)^\triangleleft] \alpha &= \lambda \langle z, \beta \rangle.(\{f^\triangleleft \langle \alpha, \beta \rangle\} z) \\ [(\triangleright f)^\triangleright] x &= \lambda k.(k \lambda \langle \gamma, y \rangle.(\{[f^\triangleright] \langle x, y \rangle\} \gamma)) & [(\blacktriangleleft f)^\triangleleft] \beta &= \lambda \langle \alpha, z \rangle.(\{f^\triangleleft \langle \alpha, \beta \rangle\} z) \\ [(\triangleleft' f)^\triangleright] \langle x, y \rangle &= \lambda \gamma.(\{[f^\triangleright] y\} \lambda n.(n \langle x, \gamma \rangle)) & [(\blacktriangleright' f)^\triangleleft] \langle \alpha, \beta \rangle &= \lambda z.(\{[f^\triangleleft] \alpha\} \langle z, \beta \rangle) \\ [(\triangleright' f)^\triangleright] \langle x, y \rangle &= \lambda \gamma.(\{[f^\triangleright] x\} \lambda m.(m \langle \gamma, y \rangle)) & [(\blacktriangleleft' f)^\triangleleft] \langle \alpha, \beta \rangle &= \lambda z.(\{[f^\triangleleft] \beta\} \langle \alpha, z \rangle) \end{aligned} \quad (37)$$

Grishin interaction rules. The Grishin rules simply recombine the factors of the input structure. We work this out for $G1$ and its dual $G1'$, leaving the other cases for the reader.

$$\frac{f : A \otimes (B \otimes C) \rightarrow D}{\otimes f : (A \otimes B) \otimes C \rightarrow D} \quad \frac{f^\infty : D \rightarrow (C \oplus B)/A}{(\otimes f)^\infty : D \rightarrow C \oplus (B/A)}$$

⁷ We follow usual functional programming practice writing the function definitions equationally, e.g. $[(1_A)^\triangleright]$ is the function $\lambda x \lambda k.(k x)$.

$$\begin{aligned}
\lceil (\otimes f)^\triangleright \rceil \langle \langle w, v \rangle, z \rangle &= \lambda \delta. ((\lceil f^\triangleright \rceil \langle w, \langle v, z \rangle \rangle) \delta) \\
\lceil (\otimes f)^\triangleright \rceil = \lceil (((\otimes f)^\infty)^\triangleleft \rceil \langle \gamma, \delta \rangle &= \lambda z. (\delta \lambda \langle \beta, x \rangle. ((\lceil (f^\infty)^\triangleleft \rceil \lambda m. (m \langle \langle \gamma, \beta \rangle, x \rangle)), z))
\end{aligned} \tag{38}$$

Example. We contrast the cbv (left) and cbn (right) interpretations of a simple sentence ‘somebody left’ involving a generalized quantifier expression of type $(s \otimes s) \otimes np$. Literals are indexed to facilitate identification of the axiom matchings.

$$\begin{array}{ccc}
\frac{s_4 \vdash s_0 \quad s_1 \vdash s_5}{(s_4 \otimes s_5) \vdash (s_0 \otimes s_1)} \otimes & & \frac{s_5 \vdash s_1 \quad s_0 \vdash s_4}{(s_1 \setminus s_0) \vdash (s_5 \setminus s_4)} \setminus \\
\frac{np_2 \vdash np_3 \quad s_4 \vdash ((s_0 \otimes s_1) \oplus s_5)}{(np_3 \setminus s_4) \vdash (np_2 \setminus ((s_0 \otimes s_1) \oplus s_5))} \triangleright' & & \frac{(s_5 \otimes (s_1 \setminus s_0)) \vdash s_4 \quad np_3 \vdash np_2}{((s_5 \otimes (s_1 \setminus s_0)) \otimes np_2) \vdash (s_4 \otimes np_3)} \triangleleft' \\
\frac{(np_2 \otimes (np_3 \setminus s_4)) \vdash ((s_0 \otimes s_1) \oplus s_5)}{((s_0 \otimes s_1) \otimes (np_2 \otimes (np_3 \setminus s_4))) \vdash s_5} \triangleleft' & & \frac{(s_5 \otimes (s_1 \setminus s_0)) \vdash ((s_4 \otimes np_3) \oplus np_2)}{s_5 \vdash (((s_4 \otimes np_3) \oplus np_2) / (s_1 \setminus s_0))} \triangleright' \\
\frac{((s_0 \otimes s_1) \otimes (np_2 \otimes (np_3 \setminus s_4))) \vdash s_5}{(((s_0 \otimes s_1) \otimes np_2) \otimes (np_3 \setminus s_4)) \vdash s_5} \otimes & & \frac{s_5 \vdash (((s_4 \otimes np_3) \oplus np_2) / (s_1 \setminus s_0))}{s_5 \vdash ((s_4 \otimes np_3) \oplus (np_2 / (s_1 \setminus s_0)))} \otimes^\infty
\end{array}$$

(cbv) $\lambda c. ((\llbracket \text{left} \rrbracket \langle \pi^2 \llbracket \text{somebody} \rrbracket, \lambda z. (\pi^1 \llbracket \text{somebody} \rrbracket \langle z, c \rangle)))$

(cbn) $\lambda c. ((\llbracket \text{somebody} \rrbracket \lambda \langle q, y \rangle. (y \langle c, \lambda c'. (\llbracket \text{left} \rrbracket \langle c', q \rangle))))$

3 Linguistic Applications

In-depth discussion of the linguistic applications of **LG** is beyond the scope of this paper. We suffice with some very simple illustrations. Recall the two problems with Lambek calculus discussed in §1: non-local semantic construal and displacement. These problems find a natural solution in the symmetric Lambek-Grishin systems. As shown in (39), in both cases the solution starts from a lexical type assignment from which the usual Lambek type is derivable.

$$\begin{array}{ll}
\text{someone} & (s \otimes s) \otimes np \vdash s / (np \setminus s) \\
\text{which} & (n \setminus n) / ((s \otimes s) \oplus (s / np)) \vdash (n \setminus n) / (s / np)
\end{array} \tag{39}$$

Non-local scope construal. An example would be a sentence of the type ‘Alice suspects someone is cheating’. The sentence has two natural interpretations: there could be a particular player whom Alice suspects of cheating (for example, because she sees a card sticking out of his sleeve), or she could have the feeling that cheating is going on, without having a particular player in mind (for example, because she finds two aces of spades in her hand). A Lambek type assignment $s / (np \setminus s)$ for ‘someone’ is restricted to local construal in the embedded clause, i.e. the second interpretation. The assignment $(s \otimes s) \otimes np$ also allows construal at the main clause level as required for the first interpretation. In (40) one finds a summary derivation⁸ for the reading where ‘someone’ has wide scope.

⁸ Full proof for the endsequent under \cdot^∞ is $\triangleright' \triangleright' \otimes^* \triangleright \otimes \triangleright \otimes \triangleleft' (\triangleright' \triangleleft' (\triangleright' (1_s \otimes 1_{np}) \otimes (1_s \otimes 1_{np}))) / 1_s$.

By means of the Grishin interactions, the $(s \otimes s)$ subformula moves to the top level leaving behind a np resource *in situ*; $(s \otimes s)$ then shifts to the succedent by means of the dual residuation principle, and establishes scope via the dual application law. Under the CPS interpretation discussed above, the derivation is associated with the term in (41). The reader is invited to consult [25] for a CPS interpretation of the lexical constants which associates this term with the reading $(\exists \lambda x.((\text{suspects}(\text{cheating } x)) \text{ alice}))$ as required.

$$\frac{\frac{\frac{np \otimes (((np \setminus s)/s) \otimes (np \otimes (np \setminus s))) \vdash s \quad s \vdash (s \otimes s) \oplus s}{np \otimes (((np \setminus s)/s) \otimes (np \otimes (np \setminus s))) \vdash ((s \otimes s) \oplus s)} \text{trans}}{((s \otimes s) \otimes (np \otimes (((np \setminus s)/s) \otimes (np \otimes (np \setminus s)))) \vdash s} \text{drp}}{\frac{np \otimes (((np \setminus s)/s) \otimes \underbrace{((s \otimes s) \otimes np)}_{\text{someone}} \otimes (np \setminus s))) \vdash s}{G2} \text{G2} \quad (40)$$

$$\lambda c.(\llbracket \text{someone} \rrbracket \lambda \langle q, y \rangle. (y \langle c, \lambda c'. (\llbracket \text{suspect} \rrbracket \langle \lambda c''. (\llbracket \text{cheating} \rrbracket \langle c'', q \rangle), \langle c', \llbracket \text{alice} \rrbracket \rangle \rangle))) \quad (41)$$

Displacement. The second example deals with *wh* dependencies as in ‘(movie which) John (np) saw $((np \setminus s)/np = tv)$ on TV $((np \setminus s) \setminus (np \setminus s) = adv)$. The shorthand derivation in (42) combines the Grishin principles of (12) and their converses. The (s/np) subformula is added to the antecedent via the dual residuation principle, and *lowered* to the target tv via applications of (Gn^{-1}) . The tv context is then shifted to the succedent by means of the (dual) residuation principles, and the relative clause body with its np hypothesis in place is reconfigured by means of (Gn) and residuation shifting.

$$\frac{\frac{\frac{np \otimes ((tv \otimes np) \otimes adv) \vdash s \quad s \vdash (s \otimes s) \oplus s}{np \otimes ((tv \otimes np) \otimes adv) \vdash (s \otimes s) \oplus s} \text{trans}}{\frac{tv \vdash ((np \setminus (s \otimes s))/adv) \oplus (s/np)}{np \otimes ((tv \otimes (s/np)) \otimes adv) \vdash s \otimes s} \text{Gn, rp}}{\frac{(np \otimes (tv \otimes adv)) \otimes (s/np) \vdash s \otimes s}{(np \otimes (tv \otimes adv)) \otimes (s/np) \vdash s \otimes s} \text{rp, drp}}{\frac{(np \otimes (tv \otimes adv)) \otimes (s/np) \vdash s \otimes s}{np \otimes (tv \otimes adv) \vdash (s \otimes s) \oplus (s/np)} \text{Gn}^{-1} \text{drp}} \quad (42)$$

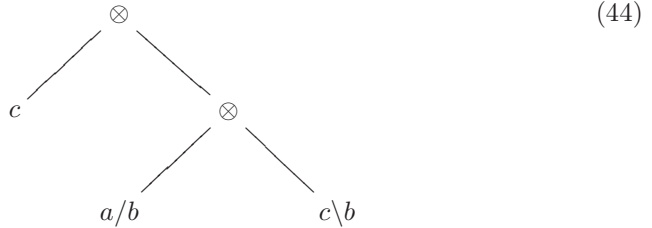
The derivation can be summarized in the derived rule of inference (\dagger), which achieves the same effect as the extraction rule under modal control (\ddagger). An attractive property of **LG** is that the expressivity resides entirely in the Grishin interaction principles: the composition operation \otimes in itself (or the dual \oplus) allows no structural rules at all, which means that the **LG** notion of wellformedness is fully sensitive to linear order and constituent structure of the grammatical material.

$$\frac{\Gamma[\Delta \circ B] \Rightarrow C}{\Gamma[\Delta] \Rightarrow (C \otimes C) \oplus (C/B)} \dagger \quad \frac{\Gamma[\Delta \circ B] \Rightarrow C}{\Gamma[\Delta] \Rightarrow C/\diamond \Box B} \ddagger \quad (43)$$

Similarity. Rotations of the type $((A \setminus C)/B \sim A \setminus (C/B), (C/B)/A \sim (C/A)/B$ make it possible to promote any embedded argument to a left or right peripheral

position where it is visible for slash introduction. Since the original and the rotated types are in the \sim relation, one can lexically assign their meet type according to the algorithm given in the previous section. Below we look at extraction and head adjunction from this perspective. We show how the strategy of assigning a meet type can be used to overcome the limitations of the Lambek calculus (both **NL** and **L**), and how overgeneration can be avoided by appropriate modal decoration.

Phenomena of head adjunction likewise give rise to dependencies for which the rotation invariant proves useful. In (44) below one finds a schematic representation of a crossed dependency in Dutch, as in the phrase ‘(dat Jan) boeken (c) wil (a/b) lezen ($c \backslash b$)’ with the order object–modal auxiliary–transitive infinitive. One would like to derive type a (tensed verb phrase) for this structure. As with extraction, there is a double challenge: one wants to allow the transitive infinitive to communicate with its direct object across the intervening modal auxiliary; at the same time, one has to rule out the ungrammatical order $(a/b) \otimes (c \otimes (c \backslash b))$ which with the indicated types would make a derivable.



To bring our strategy into play, note that

$$c \backslash b \stackrel{(\text{lifting})}{\sim} c \backslash ((a/b) \backslash a) \stackrel{(\text{rotation})}{\sim} (a/b) \backslash (c \backslash a)$$

For the original $c \backslash b$ and the rotated $(a/b) \backslash (c \backslash a)$ we have join C and meet D :

$$C = ((a/b) \backslash a) \oplus (c \backslash (a \otimes a))$$

$$D = ((c \backslash b)/C) \otimes (C \oslash (((a/b) \backslash (c \backslash a)) \oslash C))$$

To make the ungrammatical order modal auxiliary–object–transitive infinitive underivable, we can impose subtyping constraints using modal decoration. It is enough to change the type of the modal auxiliary to $a/\diamond \Box b$, and modify D accordingly, marking the rotated argument:

$$D' = ((c \backslash b)/C) \otimes (C \oslash (((a/\diamond \Box b) \backslash (c \backslash a)) \oslash C))$$

Recall that $\diamond \Box A \vdash A$. The join type C in other words can remain as it was since

$$(a/\diamond \Box b) \backslash (c \backslash a) \vdash ((a/b) \backslash a) \oplus (c \backslash (a \otimes a))$$

4 Conclusions

Natural languages exhibit mismatches between the articulation of compound expressions at the syntactic and at the semantic level; such mismatches seem to obstruct a direct compositional interpretation. In the face of this problem, two kinds of reaction have been prominent. The first kind is exemplified by Curry's [27] position in his contribution to the 1960 New York conference where also [2] was presented: Curry criticizes Lambek for incorporating syntactic considerations in his category concept, and retreats to a semantically biased view of categories. The complementary reaction is found in Chomskyan generative grammar, where precedence is given to syntax, and where the relevance of modeltheoretic semantics is questioned.

The work reviewed in this paper maintains the strong view that the type-forming operations are constants both in the syntactic and in the semantic dimension. Semantic uniformity is reconciled with structural diversity by means of structure-preserving interactions between the composition operation \otimes and its residuals and a dual family \oplus ; the symmetries among these families restore the strong Curry-Howard view on derivational semantics.

References

1. Grishin, V.: On a generalization of the Ajdukiewicz-Lambek system. In: *Studies in Nonclassical Logics and Formal Systems*. Nauka, Moscow (1983) 315–334 [English translation in Abrusci and Casadio (eds.) *Proceedings 5th Roma Workshop*, Bulzoni Editore, Roma (2002)]
2. Lambek, J.: On the calculus of syntactic types. In: Jakobson, R. (ed.) *Structure of Language and Its Mathematical Aspects*, pp. 166–178 (1961)
3. Došen, K.: A brief survey of frames for the Lambek calculus. *Zeitschrift für mathematischen Logik und Grundlagen der Mathematik* 38, 179–187 (1992)
4. Moortgat, M.: Multimodal linguistic inference. *Language and Information* 5(3–4), 349–385 (1996)
5. Bernardi, R.: Reasoning with polarities in categorial type logic. PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University (2002)
6. Vermaat, W.: The logic of variation. A cross-linguistic account of wh-question formation. PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University (2006)
7. Kurtonina, N., Moortgat, M.: Structural control. In: Blackburn, P., de Rijke, M. (eds.) *Specifying Syntactic Structures*, pp. 75–113. CSLI Publications, Stanford, CA (1997)
8. Moortgat, M.: Categorial type logics. In: van Benthem, J., ter Meulen, A. (eds.) *Handbook of Logic and Language*. Elsevier/MIT Press, 93–177 (1997)
9. de Groote, P.: The non-associative Lambek calculus with product in polynomial time. In: Murray, N.V. (ed.) *TABLEAUX 1999. LNCS (LNAI)*, vol. 1617, pp. 128–139. Springer, Heidelberg (1999)
10. Pentus, M.: Lambek calculus is NP-complete. *Theoretical Computer Science* 357, 186–201 (2006)
11. Moot, R.: Proof Nets for Linguistic Analysis. PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University (2002)

12. Dunn, J.M.: Gaggles theory: An abstraction of galois connections and residuation with applications to negation, implication, and various logical operations. In: van Eijck, J. (ed.) *JELIA 1990*. LNCS, vol. 478, pp. 31–51. Springer, Heidelberg (1991)
13. Lambek, J.: From categorial to bilinear logic. In: Schröder-Heister, K.D.P. (ed.) *Substructural Logics*, Oxford University Press, Oxford (1993)
14. Goré, R.: Substructural logics on display. *Logic Journal of IGPL* 6(3), 451–504 (1997)
15. Moortgat, M., Oehrle, R.: Proof nets for the grammatical base logic. In: Abrusci, M., Casadio, C. (eds.) *Dynamic perspectives in logic and linguistics*, Roma, Bulzoni pp. 131–143 (1999)
16. De Groote, P., Lamarche, F.: Classical non-associative Lambek calculus. *Studia Logica* 71, 335–388 (2002)
17. Capelletti, M.: Parsing with structure-preserving categorial grammars. PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University (2007)
18. Lambek, J.: The mathematics of sentence structure. *American Mathematical Monthly* 65, 154–170 (1958)
19. Foret, A.: On the computation of joins for non-associative Lambek categorial grammars. In: Levy, J., Kohlhase, M., Niehren, J., Villaret, M. (eds.) *Proceedings of the 17th International Workshop on Unification, UNIF’03 Valencia*, pp. 25–38 (2003)
20. Pentus, M.: The conjoinability relation in Lambek calculus and linear logic. *ILLC Prepublication Series ML-93-03*, Institute for Logic, Language and Computation, University of Amsterdam (1993)
21. Moortgat, M., Pentus, M.: Type similarity for the Lambek-Grishin calculus. *AI preprint series*, Utrecht University (2007) (Submitted to the 12th Formal Grammar Conference, Dublin)
22. Kurtonina, N.: Frames and Labels. A Modal Analysis of Categorial Inference. PhD thesis, OTS Utrecht, ILLC Amsterdam (1995)
23. Allwein, G., Dunn, J.M.: Kripke models for Linear Logic. *Journal of Symbolic Logic* 58(2), 514–545 (1993)
24. Kurtonina, N., Moortgat, M.: Relational semantics for the Lambek-Grishin calculus. Technical report, *AI Preprint Series*, Utrecht (2007) (Submitted to MOL’10, UCLA, Los Angeles)
25. Bernardi, R., Moortgat, M.: Continuation semantics for symmetric categorial grammar. In: Leivant, D., de Queiroz, R. (eds.) *Proceedings 14th Workshop on Logic, Language, Information and Computation (WoLLIC’07) 2007*, LNCS, vol. 4576, Springer, Heidelberg (this volume)
26. Curien, P., Herbelin, H.: Duality of computation. In: *International Conference on Functional Programming (ICFP’00)*, 2000, pp. 233–243 [2005: corrected version]
27. Curry, H.B.: Some logical aspects of grammatical structure. In: Jacobson, R. (ed.) *Structure of Language and its Mathematical Aspects*. Volume XII of *Proceedings of the Symposia in Applied Mathematics*. American Mathematical Society, pp. 56–68 (1961)

A Admissibility of Cut

A cut-elimination algorithm for the **Frm**(/, \otimes , \backslash) fragment (i.e. **NL**) is presented in Moortgat and Oehrle [15]. Induction is on the degree of a cut inference, measured as the number of type-forming operations in the factors involved ($|A|+|B|+|C|$, where

B is the cut formula, A the left premise lhs, C the right premise rhs). Targets for elimination are ‘uppermost’ cut inferences: instances of cut which are themselves derived without the aid of the cut rule. One shows that such an instance of cut can be replaced by one or more cuts of lower degree; the basis of the induction being the case where one of the cut premises is an instance of the axiom schema. The elimination process is iterated until the derivation is cut-free.

For the transformations one distinguishes principal cuts from permutation cases. Principal cuts, in the combinator presentation of §2.1, are cases where the cut formula in the two premises is introduced by the monotonicity rules. Such cuts are replaced by cuts on the four subformulas involved, with a reduction of the complexity degree. For the non-principal cases, one shows that the application of cut can be pushed upwards, again reducing complexity. The extension of the procedure of [15] to **Frm**(\otimes, \oplus, \odot) is immediate via arrow reversal. In Fig 3 and Fig 4 we give the \oplus and \odot cases with the corresponding equations on the proof terms; the \odot case is symmetric. This covers the minimal symmetric system **LG**₀. For full **LG** (the extension of **LG**₀ with the \mathcal{G}^\uparrow interaction principles of (12)), what remains to be shown is that applications of cut never have to be immediately preceded by applications of the Grishin interaction rules. In Fig 5 is an instance of cut immediately preceded by \odot . We have unfolded the left

$$\begin{array}{c}
 \frac{h : D \rightarrow A \oplus B}{D \otimes B \rightarrow A} \blacktriangleright \quad \frac{f : A \rightarrow A'}{f : A \rightarrow A'} \\
 \frac{\frac{f : A \rightarrow A' \quad g : B \rightarrow B'}{A \oplus B \rightarrow A' \oplus B'} \quad \frac{\frac{D \otimes B \rightarrow A' \quad D \rightarrow A' \oplus B}{A' \otimes D \rightarrow B} \blacktriangleleft \quad g : B \rightarrow B'}{A' \otimes D \rightarrow B'} \blacktriangleleft' \\
 \vdots \quad \frac{D \rightarrow A' \oplus B'}{D \rightarrow A' \oplus B'} \blacktriangleleft' \\
 \vdots \\
 \frac{D \rightarrow C}{D \rightarrow C}
 \end{array}
 \quad \sim \quad
 \begin{array}{c}
 \frac{h : D \rightarrow A \oplus B}{D \otimes B \rightarrow A} \blacktriangleright \quad \frac{f : A \rightarrow A'}{f : A \rightarrow A'} \\
 \frac{D \otimes B \rightarrow A' \quad D \rightarrow A' \oplus B}{A' \otimes D \rightarrow B} \blacktriangleleft \quad g : B \rightarrow B' \\
 \frac{A' \otimes D \rightarrow B'}{D \rightarrow A' \oplus B'} \blacktriangleleft' \\
 \vdots \\
 \frac{D \rightarrow C}{D \rightarrow C}
 \end{array}$$

Fig. 3. $k[f \oplus g] \circ h = k[\blacktriangleleft'(g \circ \blacktriangleleft \blacktriangleright'(f \circ \blacktriangleright h))]$

$$\begin{array}{c}
 \frac{f : C' \rightarrow C \quad g : B \rightarrow B'}{C' \otimes B' \rightarrow C \otimes B} \\
 \vdots \\
 \frac{D \rightarrow C \otimes B}{D \rightarrow C \otimes B} \quad h : C \otimes B \rightarrow A \\
 \frac{D \rightarrow C \otimes B \quad h : C \otimes B \rightarrow A}{D \rightarrow A}
 \end{array}
 \quad \sim \quad
 \begin{array}{c}
 \frac{h : C \otimes B \rightarrow A}{C \rightarrow A \oplus B} \blacktriangleright' \quad \frac{f : C' \rightarrow C}{f : C' \rightarrow C} \\
 \frac{C' \rightarrow A \oplus B}{A \otimes C' \rightarrow B} \blacktriangleleft \quad g : B \rightarrow B' \\
 \frac{A \otimes C' \rightarrow B'}{C' \rightarrow A \oplus B'} \blacktriangleleft' \quad \frac{C' \otimes B' \rightarrow A}{C' \otimes B' \rightarrow A} \blacktriangleright' \\
 \vdots \\
 \frac{D \rightarrow A}{D \rightarrow A}
 \end{array}$$

Fig. 4. $h \circ k[f \otimes g] = k[\blacktriangleright \blacktriangleleft'(g \circ \blacktriangleleft(\blacktriangleright' h \circ f))]$

$$\begin{array}{c}
\frac{g : A \rightarrow A' \quad f : B' \rightarrow B}{\frac{A' \otimes B' \rightarrow A \otimes B}{\vdots}} \\
\frac{k_1[g \otimes f] : E' \rightarrow A \otimes B \quad h : C' \rightarrow C}{\frac{E' \otimes C' \rightarrow (A \otimes B) \otimes C}{\vdots}} \\
\frac{k'_1[k_1[g \otimes f] \otimes h] : E \rightarrow (A \otimes B) \otimes C \quad \otimes k_2 : (A \otimes B) \otimes C \rightarrow D}{\otimes k_2 \circ k'_1[k_1[g \otimes f] \otimes h] : E \rightarrow D}
\end{array}$$

Fig. 5. An instance of $G1$ feeding cut

$$\begin{array}{c}
\frac{k_2 : A \otimes (B \otimes C) \rightarrow D}{\frac{B \otimes C \rightarrow A \oplus D}{(B \otimes C) \otimes D \rightarrow A} \blacktriangleright'} \blacktriangleleft' \\
\frac{g : A \rightarrow A'}{(B \otimes C) \otimes D \rightarrow A'} \blacktriangleright' \\
\frac{B \otimes C \rightarrow A' \oplus D}{B \rightarrow (A' \oplus D)/C} \blacktriangleright \\
\frac{f : B' \rightarrow B}{B' \rightarrow (A' \oplus D)/C} \blacktriangleright' \\
\frac{h : C' \rightarrow C}{\frac{B' \otimes C \rightarrow A' \oplus D}{C \rightarrow B' \setminus (A' \oplus D)} \blacktriangleright'} \blacktriangleleft' \\
\frac{C' \rightarrow B' \setminus (A' \oplus D)}{B' \otimes C' \rightarrow A' \oplus D} \blacktriangleleft' \\
\frac{A' \otimes (B' \otimes C') \rightarrow D}{(A' \otimes B') \otimes C' \rightarrow D} \otimes \\
\frac{(A' \otimes B') \otimes C' \rightarrow D}{A' \otimes B' \rightarrow D/C'} \blacktriangleright \\
\vdots \\
\frac{E' \rightarrow D/C'}{E' \otimes C' \rightarrow D} \blacktriangleright' \\
\vdots \\
E \rightarrow D
\end{array}$$

$$\otimes k_2 \circ k'_1[k_1[g \otimes f] \otimes h] = k'_1[\blacktriangleright' k_1[\blacktriangleright \otimes \blacktriangleleft' (\blacktriangleleft \blacktriangleright' (\blacktriangleright \blacktriangleright' (g \circ \blacktriangleright \blacktriangleleft' k_2) \circ f) \circ h)]]$$

Fig. 6. Permutability of Grishin and cut

cut premise so as to unveil the applications of the \otimes and \oplus monotonicity rules within their contexts. This derivation can be rewritten as in Fig 6 where the cut on $(A \otimes B) \otimes C$ is replaced by cuts on the factors A , B and C , followed by the Grishin inference \otimes .

Computational Interpretations of Classical Linear Logic

Paulo Oliva

Department of Computer Science
Queen Mary, University of London
London E1 4NS, UK
pbo@dcs.qmul.ac.uk
<http://www.dcs.qmul.ac.uk/~pbo>

Abstract. We survey several computational interpretations of classical linear logic based on two-player one-move games. The moves of the games are higher-order functionals in the language of finite types. All interpretations discussed treat the exponential-free fragment of linear logic in a common way. They only differ in how much advantage one of the players has in the exponentials games. We discuss how the several choices for the interpretation of the modalities correspond to various well-known functional interpretations of intuitionistic logic, including Gödel's Dialectica interpretation and Kreisel's modified realizability.

1 Introduction

This article surveys several interpretations [3,16,17,18] of classical linear logic based on one-move two-player (Eloise and Abelard) games. As we will see, these are related to functional interpretations of intuitionistic logic such as Gödel's Dialectica interpretations [11] and Kreisel's modified realizability [14].

The intuition behind the interpretation is that each formula A defines an adjudication relation between arguments pro (Eloise's move) and against (Abelard's move) the truth of A . If the formula is in fact true, then Eloise should have no problem in winning the game. The interpretation of each of the logical connectives, quantifiers and exponentials corresponds to constructions that build new games out of given games. Given the symmetry of the interpretation, the game corresponding to the linear negation of A is simply the game A with the roles of the two players swapped. The simplest interpretation of the exponential games view these as games where only one player needs to make a move. For instance, in the game $?A$, only Abelard makes a move, and Eloise will win in case she has a winning move for the game A with the given Abelard's move. A symmetric situation occurs in the case of the game $!A$, only that Abelard now has the advantage. The idea is that the exponentials $?$ and $!$ serve as trump cards for Eloise and Abelard, respectively.

The paper is organised as follows. The basic interpretation of the exponential-free fragment of classical linear logic is presented in Section 2, and soundness of the interpretation is proved. Completeness of the interpretation is presented in Section 3. A simple form of branching quantifier is used for the proof of completeness. In Section 4, we discuss the various possibilities for the interpretation of the exponentials.

For an introduction to modified realizability see chapter III of [20] or the book chapter [21]. Background on Gödel's Dialectica interpretation can be obtained in [1]. For an introduction to linear logic see Girard's original papers [9,10].

1.1 Classical Linear Logic LL^ω

We work with an extension of classical linear logic to the language of all finite types. The set of *finite types* \mathcal{T} is inductively defined as follows:

- $o \in \mathcal{T}$;
- if $\rho, \sigma \in \mathcal{T}$ then $\rho \rightarrow \sigma \in \mathcal{T}$.

For simplicity, we deal with only one basic finite type o .

We assume that the terms of LL^ω contain all typed λ -terms, i.e. variables x^ρ for each finite type ρ ; λ -abstractions $(\lambda x^\rho. t^\sigma)^{\rho \rightarrow \sigma}$; and term applications $(t^{\rho \rightarrow \sigma} s^\rho)^\sigma$. Note that we work with the standard typed λ -calculus, and not with a linear variant thereof. The atomic formulas of LL^ω are $A_{\text{at}}, B_{\text{at}}, \dots$ and $A_{\text{at}}^\perp, B_{\text{at}}^\perp, \dots$. For simplicity, the standard propositional constants $0, 1, \perp, \top$ of linear logic have been omitted, since the realizability interpretation of atomic formulas is trivial (see Definition 1).

Table 1. Structural rules

$A_{\text{at}}, A_{\text{at}}^\perp$	(id)	$\frac{\Gamma, A \quad \Delta, A^\perp}{\Gamma, \Delta}$	(cut)	$\frac{\Gamma}{\pi\{\Gamma\}}$	(per)
--------------------------------------	------	--	-------	--------------------------------	-------

Formulas are built out of atomic formulas $A_{\text{at}}, B_{\text{at}}, \dots$ and $A_{\text{at}}^\perp, B_{\text{at}}^\perp, \dots$ via the connectives $A \wp B$ (par), $A \otimes B$ (tensor), $A \diamond_z B$ (if-then-else), and quantifiers $\forall x A$ and $\exists x A$ (exponentials are treated in Section 4). The *linear negation* A^\perp of an arbitrary formula A is an abbreviation as follows:

$$\begin{aligned}
 (A_{\text{at}})^\perp &\equiv A_{\text{at}}^\perp & (A_{\text{at}}^\perp)^\perp &\equiv A_{\text{at}} \\
 (\exists z A)^\perp &\equiv \forall z A^\perp & (\forall z A)^\perp &\equiv \exists z A^\perp \\
 (A \wp B)^\perp &\equiv A^\perp \otimes B^\perp & (A \otimes B)^\perp &\equiv A^\perp \wp B^\perp \\
 (A \diamond_z B)^\perp &\equiv A^\perp \diamond_z B^\perp.
 \end{aligned}$$

So, $(A^\perp)^\perp$ is syntactically equal to A .

The structural rules of linear logic (shown in Table 1) do not contain the usual rules of weakening and contraction. These are added separately, in a controlled manner via the use of modalities (cf. Section 4). The rules for the multiplicative connectives and quantifiers are shown in Table 2, with the usual side condition in the rule (\forall) that the variable z must not appear free in Γ .

We will deviate from the standard formulation of linear logic, in the sense that we will use the if-then-else logical constructor $A \diamond_z B$ instead of standard additive

Table 2. Rules for multiplicative connectives and quantifiers

$\frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \otimes B} (\otimes)$	$\frac{\Gamma, A, B}{\Gamma, A \wp B} (\wp)$	$\frac{\Gamma, A}{\Gamma, \forall z A} (\forall)$	$\frac{\Gamma, A[t/z]}{\Gamma, \exists z A} (\exists)$
---	--	---	--

conjunction and disjunction¹. The logical rules for \diamond_z are shown in Table 3, where $(z)(\gamma_0, \gamma_1)$ denotes a conditional λ -term which reduces to either γ_0 or γ_1 depending on whether the boolean variable z reduces to true or false, respectively. The standard additives can be defined as

$$A \wedge B := \forall z (A \diamond_z B)$$

$$A \vee B := \exists z (A \diamond_z B)$$

with the help of quantification over booleans.

Table 3. Rules for if-then-else connective

$\frac{\Gamma[\gamma_0], A \quad \Gamma[\gamma_1], B}{\Gamma[(z)(\gamma_0, \gamma_1)], A \diamond_z B} (\diamond_z)$	$\frac{\Gamma, A}{\Gamma, A \diamond_t B} (\diamond_t)$	$\frac{\Gamma, B}{\Gamma, A \diamond_f B} (\diamond_f)$
--	---	---

Notation. We use bold face variables $\mathbf{f}, \mathbf{g}, \dots, \mathbf{x}, \mathbf{y}, \dots$ for tuples of variables, and bold face terms $\mathbf{a}, \mathbf{b}, \dots, \gamma, \delta, \dots$ for tuples of terms. Given sequence of terms \mathbf{a} and \mathbf{b} , by $\mathbf{a}(\mathbf{b})$, we mean the sequence of terms $a_0(\mathbf{b}), \dots, a_n(\mathbf{b})$. Similarly for $\mathbf{a}[\mathbf{b}/\mathbf{x}]$.

2 Basic Interpretation

To each formula A of the exponential-free fragment of linear logic we associate a quantifier-free formula $|A|_{\mathbf{y}}^{\mathbf{x}}$, where \mathbf{x}, \mathbf{y} are fresh-variables not appearing in A . The variables \mathbf{x} in the superscript are called the *witnessing variables*, while the subscript variables \mathbf{y} are called the *challenge variables*. Intuitively, the interpretation of a formula A is a two-player (Eloise and Abelard) one-move game, where $|A|_{\mathbf{y}}^{\mathbf{x}}$ is the adjudication relation. We want that Eloise has a winning move whenever A is provable in LL^ω . Moreover, the linear logic proof of A will provide Eloise's winning move \mathbf{a} , i.e. $\forall \mathbf{y} |A|_{\mathbf{y}}^{\mathbf{a}}$.

Definition 1 (Basic interpretation). Assume we have already defined $|A|_{\mathbf{y}}^{\mathbf{x}}$ and $|B|_{\mathbf{w}}^{\mathbf{v}}$, we define

¹ See Girard's comments in [9] (p13) and [10] (p73) on the relation between the additive connectives and the if-then-else construct.

$$\begin{aligned}
|A \wp B|_{f,w}^{f,g} &:= |A|_v^{fw} \wp |B|_w^{gv} \\
|A \otimes B|_{f,g}^{x,v} &:= |A|_f^{xv} \otimes |B|_{gx}^{vx} \\
|A \diamond_z B|_{y,w}^{x,v} &:= |A|_y^{xv} \diamond_z |B|_w^{vx} \\
|\exists z A(z)|_f^{x,z} &:= |A(z)|_{fz}^x \\
|\forall z A(z)|_{y,z}^f &:= |A(z)|_y^{fz}.
\end{aligned}$$

The interpretation of atomic formulas are the atomic formulas themselves, i.e.

$$\begin{aligned}
|A_{\text{at}}| &:= A_{\text{at}} \\
|A_{\text{at}}^\perp| &:= A_{\text{at}}^\perp.
\end{aligned}$$

Notice that for atomic formulas the tuples of witnesses and challenges are both empty.

It is easy to see that $|A^\perp|_x^y \equiv (|A|_y^x)^\perp$. We now prove the soundness of the basic interpretation, i.e. we show how Eloise's winning move in the game $|A|_y^x$ can be extracted from a proof of A in classical linear logic (exponentials treated in Section 4).

Theorem 1 (Soundness). *Let A_0, \dots, A_n be formulas of LL^ω , with z as the only free-variables. If*

$$A_0(z), \dots, A_n(z)$$

is provable in LL^ω then from this proof terms $\mathbf{a}_0, \dots, \mathbf{a}_n$ can be extracted such that

$$|A_0(z)|_{x_0}^{\mathbf{a}_0}, \dots, |A_n(z)|_{x_n}^{\mathbf{a}_n}$$

is also provable in LL^ω , where $\text{FV}(\mathbf{a}_i) \in \{z, x_0, \dots, x_n\} \setminus \{x_i\}$.

Proof. The proof is by induction on the derivation of A_0, \dots, A_n . The only relevant rule where free-variables matter is the universal quantifier rule. Therefore, for all the other rules we will assume the tuple of parameters z is empty. The cases of the axiom, permutation rule and if-then-else are trivial. The other rules are treated as follows:

Multiplicatives.

$$\frac{\frac{| \Gamma |_\gamma^{[x]}, | A |_x^a}{| \Gamma |_\gamma^{[fb]}, | A |_{fb}^a} [\frac{fb}{x}] \quad \frac{| \Delta |_\delta^{[y]}, | B |_y^b}{| \Delta |_\delta^{[ga]}, | B |_{ga}^b} [\frac{ga}{y}]}{| \Gamma |_\gamma^{[fb]}, | \Delta |_\delta^{[ga]}, | A |_{fb}^a \otimes | B |_{ga}^b} (\otimes)} \quad \frac{| \Gamma |_\gamma, | A |_x^{a[y]}, | B |_y^{b[x]} (\wp)}{| \Gamma |_\gamma, | A \wp B |_{x,y}^{\lambda y. a[y], \lambda x. b[x]} (\wp)} \quad (\text{D1})$$

Cut.

$$\frac{\frac{| \Gamma |_\gamma^{[x]}, | A |_x^a}{| \Gamma |_\gamma^{[a^-]}, | A |_{a^-}^a} [\frac{a^-}{x}] \quad \frac{| \Delta |_\delta^{[x^-]}, | A^\perp |_{x^-}^{a^-}}{| \Delta |_\delta^{[a]}, | A^\perp |_a^{a^-}} [\frac{a}{x^-}]}{| \Gamma |_\gamma^{[a^-]}, | \Delta |_\delta^{[a]} (\text{cut})}$$

Note that the assumption that the tuple of variables x (respectively x^-) does not appear free in a (respectively a^-) is used crucially in the soundness of the cut rule in order to remove any circularity in the two simultaneous substitutions.

Quantifiers.

$$\frac{| \Gamma |_{\mathbf{v}}^{\gamma[z]}, | A |_{\mathbf{x}}^{a[z]} }{| \Gamma |_{\mathbf{v}}^{\gamma[z]}, | \forall z A |_{\mathbf{x},z}^{\lambda z. a[z]} } \quad (D1) \qquad \frac{| \Gamma |_{\mathbf{v}}^{\gamma[x]}, | A(t) |_{\mathbf{x}}^a }{| \Gamma |_{\mathbf{v}}^{\gamma[g^t]}, | A(t) |_{\mathbf{g}^t}^a } \frac{[\frac{g^t}{x}]}{| \Gamma |_{\mathbf{v}}^{\gamma[g^t]}, | \exists z A(z) |_{\mathbf{g}}^{a,t} } \quad (D1')$$

This concludes the proof. \square

3 Completeness

In this section we investigate the completeness of the interpretation given above. More precisely, we ask the question: for which extension of classical linear logic LL^* it is the case that if there are terms a_0, \dots, a_n such that $|A_0(z)|_{x_0}^{a_0}, \dots, |A_n(z)|_{x_n}^{a_n}$ is provable in LL^* then the sequence A_0, \dots, A_n is also provable in LL^* ? The idea that a formula A is interpreted as a *symmetric* game $|A|_{\mathbf{y}}^x$ between two players suggests that A is equivalent to $\exists_{\mathbf{y}}^x |A|_{\mathbf{y}}^x$, using a simple form of branching quantifier to ensure that no player has an advantage over the other. This simple branching quantifier (we will refer to these as *simultaneous quantifiers*²) can be axiomatised with the rule:

$$\frac{A_0(a_0, y_0), \dots, A_n(a_n, y_n)}{\exists_{y_0}^{x_0} A_0(x_0, y_0), \dots, \exists_{y_n}^{x_n} A_n(x_n, y_n)} \quad (\exists')$$

with the side-condition: y_i may only appear free in the terms a_j , for $j \neq i$. In particular, we will have that each y_i will not be free in the conclusion of the rule.

The standard quantifier rules can be obtained from this single rule. The rule (\forall) can be obtained in the case when only the tuple y_n is non-empty. The rule (\exists) can be obtained in the case when only the tuple x_n is non-empty. Hence, for the rest of this section we will consider that standard quantifiers $\forall x A$ and $\exists x A$ are in fact abbreviations for $\exists_x A$ and $\exists^x A$, respectively.

In terms of games, the new quantifier embodies the idea of the two players performing their moves simultaneously. The most interesting characteristic of this simultaneous quantifier is with respect to linear negation, which is defined as

$$(\exists_{\mathbf{y}}^x A)^\perp \equiv \exists_{\mathbf{x}}^y A^\perp$$

and corresponds precisely to the switch of roles between the players. Let us refer to the extension of LL^ω with the new simultaneous quantifier by LL_q^ω .

² According to Hyland [13] (footnote 18) “the identification of a sufficiently simple tensor as a Henkin quantifier is a common feature of a number of interpretations of linear logic”. The simultaneous quantifier can be viewed as a simplification of Henkin’s (branching) quantifier [4,12], in which no alternation of quantifiers is allowed on the two branches. See Bradfield [5] as well, where this simple form of branching quantifier is also used.

Theorem 2. *Extend the interpretation (Definition 1) to the system LL_q^ω by defining*

$$|\exists_w^v A(v, w)|_{g,v}^{f,v} \equiv |A(v, w)|_{g,v}^{f,w}.$$

Theorem 1 holds for the extended system LL_q^ω .

Proof. The rule for the simultaneous quantifier is handled as follows:

$$\frac{\frac{|A_0(a_0, w_0)|_{y_0}^{b_0}, \dots, |A_n(a_n, w_n)|_{y_n}^{b_n}}{|A_0(a_0, w_0)|_{g_0 a_0}^{b_0}, \dots, |A_n(a_n, w_n)|_{g_n a_n}^{b_n}} \frac{g_i a_i}{y_i}}{|\exists_{w_0}^{v_0} A_0(v_0, w_0)|_{g_0, w_0}^{\lambda w_0, b_0, a_0}, \dots, |\exists_{w_n}^{v_n} A_n(v_n, w_n)|_{g_n, w_n}^{\lambda w_n, b'_n, a_n}} \quad (\text{D1})$$

where b'_j is the sequence of terms b_j after the substitutions $g_i a_i / y_i$, for $i \neq j$. \square

In fact, since the simultaneous quantifiers are eliminated, we obtain an interpretation of LL_q^ω into LL^ω . Let us proceed now to define an extension of LL_q^ω which is complete with respect to the interpretation of Section 2. First we need some simple facts about LL_q^ω .

Lemma 1. *The following are derivable in LL_q^ω*

$$\begin{aligned} \exists_{y,z}^f A(fz, y, z) &\multimap \forall z \exists_y^x A(x, y, z) \\ \exists_y^x A(y) \otimes \exists_w^v B(w) &\multimap \exists_{f,g}^{x,v} (A(fv) \otimes B(gx)). \end{aligned}$$

Proof. These can be derived as

$$\frac{\frac{A^\perp(fz, y, z), A(fz, y, z)}{\exists_f^{y,z} A^\perp(fz, y, z), \exists_y^x A(x, y, z)} (\exists)}{\exists_f^{y,z} A^\perp(fz, y, z), \forall z \exists_y^x A(x, y, z)} (\forall)$$

and

$$\frac{\frac{\frac{A^\perp(fv), A(fv)}{A^\perp(fv), B^\perp(gx), B(gx)} (\otimes)}{A^\perp(fv), B^\perp(gx), A(fv) \otimes B(gx)} (\exists)}{\frac{\exists_x^y A^\perp(y), \exists_v^w B^\perp(w), \exists_{f,g}^{x,v} (A(fv) \otimes B(gx))}{\exists_x^y A^\perp(y) \wp \exists_v^w B^\perp(w), \exists_{f,g}^{x,v} (A(fv) \otimes B(gx))} (\wp)}$$

respectively. \square

The converse of the implications in Lemma 1, however, require extra logical principles. Consider the following principles for the simultaneous quantifier

$$\begin{aligned} \text{AC}_s &: \forall z \exists_y^x A(x, y, z) \multimap \exists_{y,z}^f A(fz, y, z) \\ \text{AC}_p &: \exists_{f,g}^{x,v} (A(fv) \otimes B(gx)) \multimap \exists_y^x A(y) \otimes \exists_w^v B(w) \end{aligned}$$

for quantifier-free formula A and B . We refer to these as the *sequential choice* AC_s and *parallel choice* AC_p . For those familiar with the modified realizability of intuitionistic logic, the principle AC_s corresponds to the standard (intentional) axiom of choice, while AC_p is a generalisation of the independence of premise principle (case when tuples x, v and w are empty).

Lemma 2. *The principles AC_s and AC_p are sound for our interpretation, i.e. for any instance P of these principles, there are terms t such that $|P|_y^t$ is derivable in LL^ω .*

Proof. Consider an instance of AC_s

$$\forall z \exists y^x A(x, y, z) \multimap \exists y^f A(fz, y, z).$$

Since A is quantifier-free, it is easy to see that premise and conclusion have the same interpretation, namely

$$\begin{aligned} |\forall z \exists y^x A(x, y, z)|_{y,z}^f &\equiv A(fz, y, z) \\ |\exists y^f A(fz, y, z)|_{y,z}^f &\equiv A(fz, y, z). \end{aligned}$$

The same is true for the premise and conclusion of AC_p . □

Let us denote by LL_{q+}^ω the extension of LL_q^ω with these two extra schemata AC_s and AC_p . The next lemma shows that, in fact, these extra principles are all one needs to show the equivalence between A and its interpretation $\exists y^x |A|_y^x$.

Lemma 3. *The equivalence between A and $\exists y^x |A|_y^x$ can be derived in the system LL_{q+}^ω .*

Proof. By induction on the logical structure of A . Consider for instance the case of $A \otimes B$.

$$A \otimes B \stackrel{(IH)}{\longleftrightarrow} \exists y^x |A|_y^x \otimes \exists w^v |B|_w^v \stackrel{(L1, AC_p)}{\longleftrightarrow} \exists_{f.g}^{x,v} (|A|_{f^x}^x \otimes |B|_{g^v}^v).$$

The other cases are treated similarly. □

Theorem 3. *Let A be a formula in the language of LL^ω . Then A is derivable in LL_{q+}^ω if and only if $|A|_y^t$ is derivable in LL^ω , for some sequence of terms t .*

Proof. The forward direction can be obtained with an extension of Theorem 1 given by Lemma 2. The converse follows from Lemma 3. □

4 Possible Interpretations of Exponentials

The exponential-free fragment of LL^ω , despite its nice properties, bears little relation to the standard logical systems of classical and intuitionistic logic. In order to recover the full strength of classical logic, we need to add back contraction and weakening. These are recovered in linear logic in a controlled manner, with the help of modalities (exponentials) $?A$ and $!A$ (cf. Table 4). The exponentials are dual to each other, i.e.

$$(?A)^\perp \equiv !A^\perp \quad (!A)^\perp \equiv ?A^\perp.$$

Girard's points out in several places (cf. [10] (p84)) that these modalities, contrary to the other connectives, are not canonical. More precisely, if we add new modalities $?^!A$ and $!^!A$ with the same rules as shown in Table 4, we will not be able to derive the equivalences $?A \leftrightarrow ?^!A$ and $!A \leftrightarrow !^!A$. This is reflected in the flexibility with which we can interpret these modalities discussed below.

Table 4. Rules for the exponentials

$\frac{? \Gamma, A}{? \Gamma, !A} (!)$	$\frac{\Gamma, A}{\Gamma, ?A} (?)$	$\frac{\Gamma, ?A, ?A}{\Gamma, ?A} (\text{con})$	$\frac{\Gamma}{\Gamma, ?A} (\text{wkn})$
--	------------------------------------	--	--

4.1 Interpretation 1: Kreisel's Modified Realizability

The first alternative for the interpretation of the exponentials we consider is one in which the game $?A$ gives maximal advantage to Eloise, and game $!A$ gives maximal advantage to Abelard. The maximal advantage corresponds to the player in question not needing to make any move, with their best possible move being played for them. More precisely, the interpretation is defined as:

$$\begin{aligned} |!A|^x &:= !\forall y |A|_y^x \\ |?A|_y &:= ?\exists x |A|_y^x. \end{aligned}$$

It is easy to see that Theorem 1 still holds when Definition 1 is extended in this way. For instance, the soundness of the rules $(!)$ and (con) are obtained as:

$$\begin{array}{c} \frac{|? \Gamma|_v, |A|_y^a}{|? \Gamma|_v, \forall y |A|_y^a} (\forall) \\ \frac{|? \Gamma|_v, \forall y |A|_y^a}{|? \Gamma|_v, !\forall y |A|_y^a} (!) \\ \hline |? \Gamma|_v, |!A|^a \end{array} \quad \begin{array}{c} \frac{| \Gamma|_v^{[y_0, y_1]}, |?A|_{y_0}, |?A|_{y_1}}{|\Gamma|_v^{[y, y]}, |?A|_y, |?A|_y} [\frac{y}{y_0}, \frac{y}{y_1}] \\ \hline | \Gamma|_v^{[y, y]}, |?A|_y} (\text{con}) \end{array}$$

We have shown [16] that when combined with the embedding of intuitionistic logic into linear logic, this choice for the interpretation of the exponentials corresponds to Kreisel's modified realizability interpretation [14] of intuitionistic logic.

Note that given this interpretation for the exponentials the relation $|A|_y^x$ is no longer quantifier-free. It is the case, however, that formulas in the image of the interpretation (we call these *fixed formulas*) are also in the kernel of the interpretation. More, precisely, if A is in the kernel of the interpretation then $|A| \equiv A$. The completeness result of Section 3 needs to be calibrated, as the schemata AC_s and AC_p need to be taken for all fixed-formulas (and not just quantifier-free formulas). Moreover, we need an extra principle

$$\text{TA} : !\exists_y^x A \multimap \exists x !\forall y A$$

called *trump advantage*, for fixed-formulas A , in order to obtain the equivalences involving exponentials, i.e. equivalence between $!\exists_y^x A$ and $\exists x !\forall y A$.

4.2 Interpretation 2: Diller-Nahm Interpretation

Another possibility for the interpretation is to give the player in question a restricted advantage by simply allowing the player to see the opponent's move, and then select

a finite set of moves. If any of these is a good move the player wins. This leads to the following interpretation of the exponentials:

$$\begin{aligned} |!A|_f^x &::= !\forall \mathbf{y} \in \mathbf{f} \mathbf{x} \mid A|_{\mathbf{y}}^x \\ |?A|_y^f &::= ?\exists \mathbf{x} \in \mathbf{f} \mathbf{y} \mid A|_{\mathbf{y}}^x. \end{aligned}$$

Again, an extension of Definition 1 in this direction would also make the Soundness Theorem 1 valid for the full classical linear logic. For instance, the soundness of the contraction rule is obtained as:

$$\frac{\frac{\frac{|\Gamma|_{\mathbf{v}}^{\gamma[\mathbf{y}_0, \mathbf{y}_1]}, |?A|_{\mathbf{y}_0}^{a_0}, |?A|_{\mathbf{y}_1}^{a_1}}{|\Gamma|_{\mathbf{v}}^{\gamma[\mathbf{y}_0, \mathbf{y}_1]}, ?\exists \mathbf{x}_0 \in a_0 \mathbf{y}_0 \mid A|_{\mathbf{y}_0}^{x_0}, ?\exists \mathbf{x}_1 \in a_1 \mathbf{y}_1 \mid A|_{\mathbf{y}_1}^{x_1}} \text{ (def)}}{|\Gamma|_{\mathbf{v}}^{\gamma[\mathbf{y}, \mathbf{y}]}, ?\exists \mathbf{x} \in (a_0 \mathbf{y}) \cup (a_1 \mathbf{y}) \mid A|_{\mathbf{y}}^x} \left[\frac{\mathbf{y}}{\mathbf{y}_0}, \frac{\mathbf{y}}{\mathbf{y}_1} \right]}{|\Gamma|_{\mathbf{v}}^{\gamma[\mathbf{y}, \mathbf{y}]}, |?A|_{\mathbf{y}}^{\lambda \mathbf{y}((a_0 \mathbf{y}) \cup (a_1 \mathbf{y}))}} \text{ (con)}}$$

while the rules (!) is dealt with as follows:

$$\frac{\frac{|\Gamma|_{\mathbf{v}}^{\gamma[\mathbf{y}]}, |A|_{\mathbf{y}}^{a[\mathbf{v}]}}{|\Gamma|_{\mathbf{v}}^{\bigcup_{\mathbf{y} \in \mathbf{f}(\mathbf{a}[\mathbf{v}])} (\gamma[\mathbf{y}]\mathbf{v})}, \forall \mathbf{y} \in \mathbf{f}(\mathbf{a}[\mathbf{v}]) \mid A|_{\mathbf{y}}^{a[\mathbf{v}]}} \text{ (!)}}{|\Gamma|_{\mathbf{v}}^{\lambda \mathbf{v} \cdot \bigcup_{\mathbf{y} \in \mathbf{f}(\mathbf{a}[\mathbf{v}])} (\gamma[\mathbf{y}]\mathbf{v})}, |!A|_{\mathbf{f}}^{a[\mathbf{v}]}} \text{ (!)}}$$

It is clear in this case that enough term construction needs to be added to the verifying system in order to deal with finite sets of arbitrary type. This choice for the treatment of the exponentials corresponds to a variant of Gödel's Dialectica interpretation due to Diller and Nahm [6].

4.3 Interpretation 3: Stein's Interpretation

A hybrid interpretation between options 1 and 2 can also be given for each parameter $n \in \mathbb{N}$. The natural number n dictates from which type level we should use option 2 (Diller-Nahm), and up to which level we should choose option 1 (modified realizability). The slogan is “only higher-type objects are witnessed”. Given a tuple of variable \mathbf{x} , we will denote by $\underline{\mathbf{x}}$ the sub-tuple containing the variables in \mathbf{x} which have type level $\geq n$, whereas $\overline{\mathbf{x}}$ denotes the sub-tuple of the variables in \mathbf{x} which have type level $< n$. For each fixed type level $n \in \mathbb{N}$ we can define an interpretation of the exponentials as:

$$\begin{aligned} |!A|_f^x &::= !\forall \underline{\mathbf{y}} \in \text{rng}(\mathbf{f} \mathbf{x}) \forall \overline{\mathbf{y}} \mid A|_{\mathbf{y}}^x \\ |?A|_y^f &::= ?\exists \underline{\mathbf{x}} \in \text{rng}(\mathbf{f} \mathbf{y}) \exists \overline{\mathbf{x}} \mid A|_{\mathbf{y}}^x. \end{aligned}$$

where $\forall \mathbf{y} \in \text{rng}(\mathbf{b}^{(n-1) \rightarrow \rho} A[\mathbf{y}])$ and $\exists \mathbf{y} \in \text{rng}(\mathbf{b}^{(n-1) \rightarrow \rho} A[\mathbf{y}])$ are used as an abbreviation for $\forall i^{n-1} A[\mathbf{b}i]$ and $\exists i^{n-1} A[\mathbf{b}i]$, respectively ($n-1$ is the pure type of type level $n-1$).

This choice corresponds to Stein's interpretation [19], and again leads to a sound interpretation of full classical linear logic.

4.4 Interpretation 4: Bounded Dialectica Interpretation

In [7,8], a “bounded” variant of Gödel’s Dialectica interpretation was developed in order to deal with strong analytical principles in classical feasible analysis. The interpretation makes use of Howard-Bezem’s strong majorizability relation \leq^* between functionals (cf. [2]). Using the majorizability relation, we can define well-behaved bounded sets which can be used as moves in the treatment of the exponential games. More precisely, the interpretation of the exponentials can also be given as:

$$\begin{aligned} |!A|_f^x &:= !\forall y \leq^* f x |A|_y^x \\ |?A|_y^f &:= ?\exists x \leq^* f y |A|_y^x. \end{aligned}$$

As argued in [15], in this case we must first perform a relativisation of the quantifiers to Bezem’s model \mathcal{M} of strongly majorizable functionals. After that, all candidate witnesses and challenges are monotone, and the soundness of the contraction rule can be derived as

$$\frac{\frac{| \Gamma |_{\gamma_v^{[y_0, y_1]}}, |?A|_{y_0}^{a_0}, |?A|_{y_1}^{a_1}}{| \Gamma |_{\gamma_v^{[y_0, y_1]}}, ?\exists x_0 \leq^* a_0 y_0 |A|_{y_0}^{x_0}, ?\exists x_1 \leq^* a_1 y_1 |A|_{y_1}^{x_1}} \text{ (def)}}{| \Gamma |_{\gamma_v^{[y, y]}}, ?\exists x \leq^* \max(a_0 y, a_1 y) |A|_y^x, \text{ (con)}} \left[\frac{y}{y_0}, \frac{y}{y_1} \right]$$

where $\max(x^\rho, y^\rho)$ is the pointwise maximum.

Besides being sound for the principles AC_s, AC_p of Section 3, and the principle TA of Section 4.1, the Dialectica interpretation of LL^ω will also interpret the linear counterpart of the bounded Markov principle

$$MP_B : \quad \forall z !\forall x \leq^* z A \multimap !\forall x A$$

for bounded formulas A . The completeness for the bounded interpretation can then be extended to deal with the exponentials as

$$\begin{array}{ccccc} & \xrightarrow{\text{TA}} & \xrightarrow{LL^\omega} & \xrightarrow{LL_q^\omega} & \\ !\exists_y^x A & \exists x !\forall y A & \exists x \forall z !\forall y \leq^* z A & \exists_f^x !\forall y \leq^* f x A & \\ & \xleftarrow{LL_q^\omega} & \xleftarrow{MP_B} & \xleftarrow{AC_s} & \end{array}$$

MP_B in particular implies the majorizability axiom by taking $A(x, y) \equiv (x = y)$. In fact, the principles AC_s, AC_p, TA, MP_B can be viewed as refinements of the (linear logic versions of) the extra principles used in the bounded functional interpretation [7]. For instance, the bounded Markov principle translates into linear logic as

$$(!\forall y A \multimap B) \multimap \exists b (!\forall y \leq^* b A \multimap B)$$

for bounded formulas A and B . This can be derived as: By TA we get from $!\forall y A \multimap B$ to $\forall b !\forall y \leq^* b A \multimap B$. Then by AC_p we get $\exists b (!\forall y \leq^* b A \multimap B)$.

4.5 Interpretation 5: Gödel's Dialectica Interpretation

The most restricted interpretation is the one in which the player's advantage in the exponential game is minimal. The only head-start will be to be able to see the opponents move. Based on the opponent's move the player will then have to make a single move. This leads to an extension of the interpretation given in Definition 1 with the interpretation of the exponentials as

$$\begin{aligned} !A|_f^x &:= !A|_{fx}^x \\ ?A|_y^f &:= ?A|_y^{fy}. \end{aligned}$$

Note that in this case the target of the interpretation is again a quantifier-free calculus (as in the basic interpretation of Section 2). For the soundness, however, we must assume that quantifier-free formulas are decidable (a usual requirement for Dialectica interpretations) in order to satisfy the contraction rule

$$\frac{\frac{| \Gamma | \gamma_v^{[y_0, y_1]}, | ?A |_{y_0}^{a_0}, | ?A |_{y_1}^{a_1} }{ | \Gamma | \gamma_v^{[y_0, y_1]}, ?A|_{y_0}^{a_0 y_0}, ?A|_{y_1}^{a_1 y_1} } \text{ (def)}}{| \Gamma | \gamma_v^{[y, y]}, ?A|_y^{a y}, } \left[\frac{y}{y_0}, \frac{y}{y_1} \right] \text{ (con)}}$$

where

$$a y := \begin{cases} a_0 y & \text{if } ?A|_y^{a_0 y} \\ a_1 y & \text{otherwise.} \end{cases}$$

The soundness of the weakening rule, and the rules (!) and (?) is trivial. This interpretation corresponds to Gödel's Dialectica interpretation [11] intuitionistic logic, used in connection to a partial realisation of Hilbert's consistency program (the consistency of classical first-order arithmetic relative to the consistence of the quantifier-free calculus T).

Besides being sound for the principles AC_s, AC_p (Section 3) and the principle TA (Section 4.1) the Dialectica interpretation of LL^ω will also be sound for the following principle

$$MP_D : \quad \forall x !A \multimap !\forall x A$$

for quantifier-free formulas A . This is the linear logic counterpart of the (semi) intuitionistic Markov principle. In fact, these are all the extra principles needed to show the equivalence between A and its Dialectica interpretation $\exists_f^x !A|_f^x$. For instance, in the case of the exponentials we have

$$\begin{array}{ccccccc} & \xrightarrow{\text{TA}} & \xrightarrow{LL^\omega} & \xrightarrow{LL_q^\omega} & & & \\ !\exists_y^x A & \xrightarrow{\quad} & \exists x !\forall y A & \xrightarrow{\quad} & \exists x \forall y !A & \xrightarrow{\quad} & \exists_f^x !A[f x / y] \\ & \xleftarrow{LL_q^\omega} & \xleftarrow{MP_D} & \xleftarrow{AC_s} & & & \end{array}$$

Acknowledgements. The interpretations presented here come from work of de Paiva [17], Blass [3], Shirahata [18] and recent work of the author [16]. The author gratefully acknowledges support of the Royal Society under grant 516002.K501/RH/kk.

References

1. Avigad, J., Feferman, S.: Gödel's functional ("Dialectica") interpretation. In: Avigad, J., Feferman, S. (eds.) *Handbook of proof theory. Studies in Logic and the Foundations of Mathematics*, vol. 137, pp. 337–405. North Holland, Amsterdam (1998)
2. Bezem, M.: Strongly majorizable functionals of finite type: a model for bar recursion containing discontinuous functionals. *The Journal of Symbolic Logic* 50, 652–660 (1985)
3. Blass, A.: Questions and answers – a category arising in linear logic, complexity theory, and set theory. In: Girard, J.-Y., Lafont, Y., Regnier, L. (eds.) *Advances in Linear Logic*. London Math. Soc. Lecture Notes, vol. 222, pp. 61–81 (1995)
4. Blass, A., Gurevich, Y.: Henkin quantifiers and complete problems. *Annals of Pure and Applied Logic* 32, 1–16 (1986)
5. Bradfield, J.: Independence: Logic and concurrency. In: *Proceedings of Computer Science Logic* (2000)
6. Diller, J., Nahm, W.: Eine Variant zur Dialectica-interpretation der Heyting Arithmetik endlicher Typen. *Arch. Math. Logik Grundlagenforsch* 16, 49–66 (1974)
7. Ferreira, F., Oliva, P.: Bounded functional interpretation. *Annals of Pure and Applied Logic* 135, 73–112 (2005)
8. Ferreira, F., Oliva, P.: Bounded functional interpretation in feasible analysis. *Annals of Pure and Applied Logic* 145, 115–129 (2007)
9. Girard, J.-Y.: Linear logic. *Theoretical Computer Science* 50(1), 1–102 (1987)
10. Girard, J.-Y.: Towards a geometry of interaction. *Contemporary Mathematics*, 92 (1989)
11. Gödel, K.: Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica* 12, 280–287 (1958)
12. Henkin, L.: Some remarks on infinitely long formulas. In: *Infinitistic methods*, pp. 167–183. Pergamon Press, New York, and Polish Scientific Publishers, Warsaw (1961)
13. Hyland, J.M.E.: Proof theory in the abstract. *Annals of Pure and Applied Logic* 114, 43–78 (2002)
14. Kreisel, G.: Interpretation of analysis by means of constructive functionals of finite types. In: Heyting, A. (ed.) *Constructivity in Mathematics*, pp. 101–128. North Holland, Amsterdam (1959)
15. Oliva, P.: Unifying functional interpretations. *Notre Dame Journal of Formal Logic* 47(2), 263–290 (2006)
16. Oliva, P.: Modified realizability interpretation of classical linear logic. In: *Proc. of the Eighteenth Annual IEEE Symposium on Logic in Computer Science LICS'07*, IEEE Computer Society Press, Los Alamitos (2007)
17. de Paiva, V.C.V.: A Dialectica-like model of linear logic. In: Dybjer, P., Pitts, A.M., Pitt, D.H., Poigné, A., Rydeheard, D.E. (eds.) *Category Theory and Computer Science. LNCS*, vol. 389, pp. 341–356. Springer, Heidelberg (1989)
18. Shirahata, M.: The Dialectica interpretation of first-order classical linear logic. *Theory and Applications of Categories* 17(4), 49–79 (2006)
19. Stein, M.: Interpretationen der Heyting-Arithmetik endlicher Typen. *Arch. Math. Logik Grundlag* 19, 175–189 (1979)
20. Troelstra, A.S.: *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. Lecture Notes in Mathematics, vol. 344. Springer, Heidelberg (1973)
21. Troelstra, A.S.: Realizability. *Handbook of proof theory* 137, 408–473 (1998)

Autonomous Programmable Biomolecular Devices Using Self-assembled DNA Nanostructures*

John H. Reif¹ and Thomas H. LaBean²

¹ Department of Computer Science, Duke University, Durham, NC 27708 USA

² Department of Chemistry, Duke University, Durham, NC 27708 USA

1 Introduction

1.1 Why Computer Science Is Relevant to the Nano-scale

The particular molecular-scale devices that are the topic of this article are known as DNA nanostructures. As will be explained, DNA nanostructures have some unique advantages among nanostructures: they are relatively easy to design, fairly predictable in their geometric structures, and have been experimentally implemented in a growing number of labs around the world. They are constructed primarily of synthetic DNA. A key principle in the study of DNA nanostructures is the use of self-assembly processes to actuate the molecular assembly. Since self-assembly operates naturally at the molecular scale, it does not suffer from the limitation in scale reduction that so restricts lithography or other more conventional top-down manufacturing techniques.

This article particularly illustrates the way in which computer science techniques and methods have impact on this emerging field. Some of the key questions one might ask about biomolecular devices are:

- What is the theoretical basis for these devices?
- How will such devices be designed?
- How can we simulate them prior to manufacture?
- How can we optimize their performance?
- How will such devices be manufactured?
- How much do the devices cost?
- How scalable is the device design?
- How will I/O be done?
- How will they be programmed?
- What efficient algorithms can be programmed?
- What will be their applications?
- How can we correct for errors or repair them?

Note that these questions are exactly the sort of questions that computer scientists routinely ask about conventional computing devices. The discipline of computer science

* Supported by NSF grants CCF-0523555, CCF-0432038, CCF-0432047. An extended version of this paper is at <http://www.cs.duke.edu/~reif/paper/AutonomousDNA/AutonomousDNA.pdf>

has developed a wide variety of techniques to address such basic questions, and we will later point out some which have an important impact to molecular-scale devices.

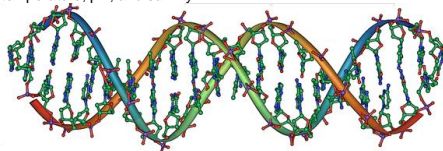
1.2 Introducing DNA Nanotechnology and Its Use to Assemble Molecular-Scale Devices

In general, nanoscience research is highly interdisciplinary. In particular, DNA self-assembly uses techniques from multiple disciplines such as biochemistry, physics, chemistry, and material science, as well as computer science and mathematics. While this makes the topic quite intellectually exciting, it also makes it challenging for a typical computer science reader. Having no training in biochemistry, he or she must obtain a coherent understanding of the topic from a short article. For this reason, this article was written with the expectation that the reader is a computer scientist with little background knowledge of chemistry or biochemistry. See Sidebar 1 for a brief introduction to DNA. In Sidebar 2 we list some reasons why DNA is uniquely suited for assembly of molecular-scale devices.

Sidebar 1: A Brief Introduction to DNA

Single stranded DNA (denoted ssDNA) is a linear polymer consisting of a sequence of DNA bases oriented along a backbone with chemical directionality. By convention, the base sequence is listed starting from the 5-prime end of the polymer and ending at the 3-prime end (these names refer to particular carbon atoms in the deoxyribose sugar units of the sugar-phosphate backbone, the details of which are not critical to the present discussion). The consecutive bases (monomer units) of an ssDNA molecule are joined via covalent bonds. There are 4 types of DNA bases adenine, thymine, guanine and cytosine typically denoted by the symbols A, T, G, and C, respectively. These bases form the alphabet of DNA; the specific sequence comprises DNA's information content. The bases are grouped into *complementary pairs* (G, C) and (A, T).

The most basic DNA operation is *hybridization* where two ssDNA oriented in opposite directions can bind to form a double stranded DNA *helix* (dsDNA) by pairing between complementary bases. DNA hybridization occurs in a buffer solution with appropriate temperature, pH, and salinity.



Structure of a DNA double helix (Created by Michael Ströck and released under the GNU Free Documentation License(GFDL).) Since the binding energy of the pair (G, C) is approximately half-again the binding energy of the pair (A, T), the association strength of hybridization depends on the sequence of complementary bases, and can be approximated by known software packages. The *melting temperature* of a DNA helix is the temperature at which half of all the molecules are fully hybridized as double helix, while the other half are single stranded. The kinetics of the DNA hybridization process is quite well understood; it often occurs in a (random) zipper-like manner, similar to a biased one-dimensional random walk.

Whereas ssDNA is a relatively floppy molecule, dsDNA is quite stiff (over lengths of less than 150 or so bases) and has the well characterized double helix structure. The exact geometry (angles and positions) of each segment of a double helix depends slightly on the component bases of its strands and can be determined from known tables. There are about 10.5 bases per full rotation on this helical axis. A *DNA nanostructure* is a multi-molecular complex consisting of a number of ssDNA that have partially hybridized along their sub-segments.

In Sidebar 3 we list some known enzymes used for manipulation of DNA nanostructures.

The area of DNA self-assembled nanostructures and robotics is by no means simply a theoretical topic - many dramatic experimental demonstrations have already been made and a number of these will be discussed. The complexity of these demonstrations have been increasing at an impressive rate (even in comparison to the rate of improvement of silicon-based technologies).

Sidebar 2: Why use DNA to Assemble Molecular-Scale Devices?

There are many advantages of DNA as a material for building things at the molecular scale.

(a) From the perspective of design, the advantages are:

- The structure of most complex DNA nanostructures can be reduced to determining the structure of short segments of dsDNA. The basic geometric and thermodynamic properties of dsDNA are well understood and can be predicted by available software systems from key relevant parameters like sequence composition, temperature and buffer conditions.
- Design of DNA nanostructures can be assisted by software. To design a DNA nanostructure or device, one needs to design a library of ssDNA strands with specific segments that hybridize to (and only to) specific complementary segments on other ssDNA. There are a number of software systems (developed at NYU, Caltech, and Duke University) for design of the DNA sequences composing DNA tiles and for optimizing their stability, which employ heuristic optimization procedures for this combinatorial sequence design task.

(b) From the perspective of experiments, the advantages are:

- The synthesis of ssDNA is now routine and inexpensive; a test tube of ssDNA consisting of any specified short sequence of bases (<150) can be obtained from commercial sources for modest cost (about half a US dollar per base at this time); it will contain a very large number (typically at least 10^{12}) identical ssDNA molecules. The synthesized ssDNA can have errors (premature termination of the synthesis is the most frequent error), but can be easily purified by well-known techniques (e.g., electrophoresis as mentioned below).
- The assembly of DNA nanostructures is a very simple experimental process: in many cases, one simply combines the various component ssDNA into a single test tube with an appropriate buffer solution at an initial temperature above the melting temperature, and then slowly cools the test tube below the melting temperature.
- The assembled DNA nanostructures can be characterized by a variety of techniques. One such technique is electrophoresis. It provides information about the relative molecular mass of DNA molecules, as well as some information regarding their assembled structures, depending on what type (denaturing or native, respectively) of electrophoresis is used. Other techniques like Atomic Force Microscopy (AFM) and Transmission Electron Microscopy (TEM) provide images of the actual assembled DNA nanostructures on 2D surfaces.

Sidebar 3: Manipulation of DNA

In addition to the hybridization reaction, there are a wide variety of known enzymes and other proteins used for manipulation of DNA nanostructures that have predictable effects. (Interestingly, these proteins were discovered in natural bacterial cells and tailored for laboratory use.) These include:

- *Restriction enzymes*, some of which can cut (or nick, which is to cut only one strand) strands of a DNA helix at locations determined by short specific DNA base sequences.
- *Ligase enzymes* that can heal nicks in a DNA helix.
- *Polymerase*, which given an initial "primer" DNA strand hybridized onto a segment of a template DNA strand, can extend the primer strand in the 5' to 3' direction by appending free nucleotides complementary to the template's nucleotides.

Besides their extensive use in other biotechnology, the above reactions, together with hybridization, are often used to execute and control DNA computations and DNA robotic operations. The restriction enzyme reactions are programmable in the sense that they are site specific, only executed as determined by the appropriate DNA base sequence. The latter two reactions, using ligase and polymerase, require the expenditure of energy via consumption of ATP molecules, and thus can be controlled by ATP concentration.

Molecular-scale devices using DNA nanostructures have been engineered to have various capabilities, ranging from (i) execution of molecular-scale computation, (ii) use as scaffolds or templates for the further assembly of other materials (such as scaffolds for various hybrid molecular electronic architectures or perhaps

high-efficiency solar-cells), (iii) robotic movement and molecular transport, and (iv) exquisitely sensitive molecular detection and amplification of single molecular events (v) transduction of molecular sensing to provide drug delivery.

2 Adelman's Initial Demonstration of a DNA-Based Computation

2.1 Adleman's Experiment

The field of DNA computing began in 1994 with a laboratory experiment described in [A98]. The goal of the experiment was to find a Hamiltonian path in a graph, which is a path that visits each node exactly once. To solve this problem, a set of ssDNA were designed based on the set of edges of the graph. When combined in a test tube and cooled, they self-assembled into dsDNA. Each of these DNA nanostructures was a linear DNA helix that corresponded to a path in the graph. If the graph had a Hamiltonian path, then one of these DNA nanostructures encoded the Hamiltonian path. By conventional biochemical extraction methods, Adelman was able to isolate only DNA nanostructures encoding Hamiltonian paths, and by determining their sequence, the explicit Hamiltonian path. It should be mentioned that this landmark experiment was designed and experimentally demonstrated by Adleman alone, a computer scientist with limited training in biochemistry.

2.2 The Non-scalability of Adelman's Experiment

While this experiment founded the field of DNA computing, it was not scalable in practice, since the number of different DNA strands needed increased exponentially with the number of nodes of the graph. Although there can be an enormous number of DNA strands in a test tube (10^{15} or more, depending on solution concentration), the size of the largest graph that could be solved by his method was limited to at most a few dozen nodes. This is not surprising, since finding the Hamiltonian path is an NP complete problem, whose solution is likely to be intractable using conventional computers. Even though DNA computers operate at the molecular-scale, they are still equivalent to conventional computers (e.g., deterministic Turing machines) in computational power. This experiment taught a healthy lesson to the DNA computing community (which is now well-recognized): to carefully examine scalability issues and to judge any proposed experimental methodology by its scalability.

2.3 Autonomous Biomolecular Computation

Shortly following Adleman's experiment, there was a burst of further experiments in DNA computing, many of which were quite ingenious. However, almost none of these DNA computing methods were autonomous, and instead required many tedious

laboratory steps to execute. In retrospect, one of the most notable aspects of Adleman's experiment was that the self-assembly phase of the experiment was completely autonomous - it required no exterior mediation. This autonomous property makes an experimental laboratory demonstration much more feasible as the scale increases. The remaining article mostly discusses autonomous devices for biomolecular computation based on self-assembly.

3 Self-assembled DNA Tiles and Lattices

Sidebar 4: DNA Nanostructures

Recall that a DNA nanostructure is a multi-molecular complex consisting of a number of ssDNA that have partially hybridized along their sub-segments. The field of DNA nanostructures was pioneered by Seeman [S04].

Particularly useful types of motifs often found in DNA nanostructures include:

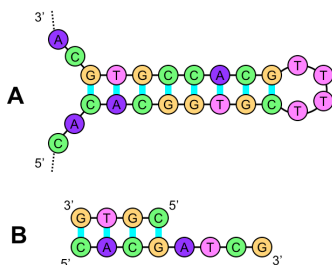


Figure of a Stem-Loop and a Sticky End. A *stem-loop* (A), where ssDNA loops back to hybridize on itself (that is, one segment of the ssDNA (near the 5' end) hybridizes with another segment further along (nearer the 3' end) on the same ssDNA strand). The shown stem consists of the dsDNA region with sequence CACGGTGC on the bottom strand. The shown loop consists of the ssDNA region with sequence TTTT. Stem-loops are often used to form patterning on DNA nanostructures. A *sticky end* (B), where unhybridized ssDNA protrudes from the end of a double helix. The sticky end shown (ATCG) protrudes from dsDNA (CACG on the bottom strand). Sticky ends are often used to combine two DNA nanostructures together via hybridization of their complementary ssDNA. The figure shows the antiparallel nature of dsDNA with the 5' end of each strand pointing toward the 3' end of its partner strand.

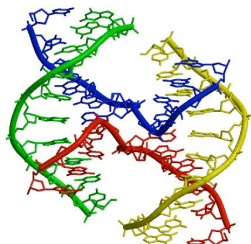


Figure of a Holliday Junction (Created by Miguel Ortiz-Lombardía, CNIO, Madrid, Spain.)

A *Holliday junction*, where two parallel DNA helices form a junction with one strand of each DNA helix (blue and red) crossing over to the other DNA helix. Holliday junctions are often used to tie together various parts of a DNA nanostructure.

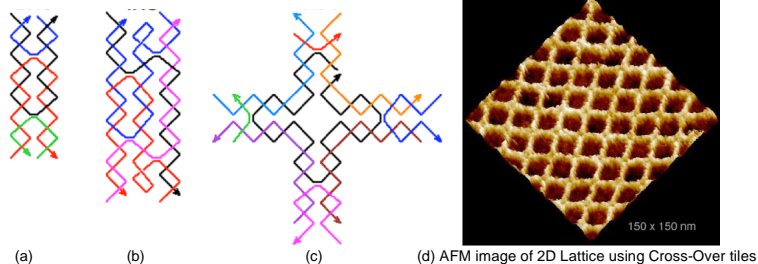
3.1 Computation by Self-assembly

The most basic way that computer science ideas have impacted DNA nanostructure design is via the pioneering work by theoretical computer scientists on a formal model of 2D tiling due to Wang in 1961, which culminated in a proof by Berger in 1966 that universal computation could be done via tiling assemblies. Winfree was the first to propose applying the concepts of computational tiling assemblies to DNA molecular constructs. His core idea was to use tiles composed of DNA to perform computations during their self-assembly process. To understand this idea, we will need an overview of DNA nanostructures, as presented in Sidebar 4.

3.2 DNA Tiles and Lattices

A *DNA tile* is a DNA nanostructure that has a number of sticky ends on its sides, which are termed *pads*. A DNA lattice is a DNA nanostructure composed of a group of DNA tiles that are assembled together via hybridization of their pads. Generally the strands composing the DNA tiles are designed to have a melting temperature above those of the pads, ensuring that when the component DNA molecules are combined together in solution, first the DNA tiles assemble, and only then, as the solution is further cooled, do the tiles bind together via hybridization of their pads.

Sidebar 5: DNA Tiles

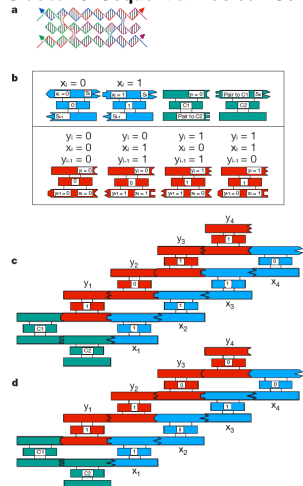


(a) Seeman and Winfree in 1998 developed a family of DNA tiles known collectively as DX tiles (see left tile (a)) that consisted of two parallel DNA helices linked by immobile Holliday junctions. They demonstrated that these tiles formed large 2D lattices, as viewed by AFM. (b) Subsequently, other DNA tiles were developed to provide for more complex strand topology and interconnections, including a family of DNA tiles known as TX tiles (see b) [L00] composed of three DNA helices. Both the DX tiles and the TX tiles are rectangular in shape, where two opposing edges of the tile have pads consisting of ssDNA sticky ends of the component strands. In addition, TX tiles have topological properties that allow for strands to propagate in useful ways through tile lattices (this property is often used for aid in patterning DNA lattices as described below). (c) Other DNA tiles known as Cross-Over tiles (see c) [Y03b] are shaped roughly square, and have pads on all four sides, allowing for binding of the tile directly with neighbors in all four directions in the lattice plane.

To program a tiling assembly, the pads of tiles are designed so that tiles assemble together as intended. Proper designs ensure that only the adjacent pads of neighboring tiles are complementary, so only those pads hybridize together. Sidebar 5 describes some principal DNA tiles.

4 Autonomous Finite State Computation Using Linear DNA Nanostructures

Sidebar 6: Sequential Boolean Computation via a Linear DNA Tiling Assembly



This figure is adapted with permission from [M00]. It shows a unit TX tile (a) and the sets of input and output tiles (b) with geometric shapes conveying sticky-end complementary matching. The tiles of (b) execute binary computations depending on their pads, as indicated by the table in (b). The (blue) input layer and (green) corner condition tiles were designed to assemble first (see example computational assemblies (c) & (d)). The (red) output layer then assembles specifically starting from the bottom left using the inputs from the blue layer. (See [M00] for more details of this molecular computation.) The tiles were designed such that an output reporter strand ran through all the n tiles of the assembly by bridges across the adjoining pads in input, corner, and output tiles. This reporter strand was pasted together from the short ssDNA sequences within the tiles using ligation enzyme mentioned previously. When the solution was warmed, this output strand was isolated and identified. The output data was read by experimentally determining the sequence of cut sites (see below). In principle, the output could be used for subsequent computations.

4.1 The First Experimental Demonstration of Autonomous Computations Using Self-assembly of DNA Nanostructures

The first experimental demonstrations of computation using DNA tile assembly was [M00]. It demonstrated a 2-layer, linear assembly of TX tiles that executed a bit-wise cumulative XOR computation. In this computation, n bits are input and n bits are output, where the i^{th} output is the XOR of the first i input bits. This is the computation occurring when one determines the output bits of a full-carry binary adder circuit. The experiment [M00] is described further in Sidebar 6.

This experiment [M00] provided answers to some of the most basic questions of concern to a computer scientist:

- *How can one provide data input to a molecular computation using DNA tiles?*

In this experiment the input sequence of n bits was defined as an “input” ssDNA strand with the input bits (1’s & 0’s) encoded by distinct short subsequences. Two different tile types (depending on whether the input bit was 0 or 1, these had specific stick-ends and also specific subsequences at which restriction enzymes can cut the

DNA backbone) were available to assemble around each of the short subsequences comprising the input strand, forming the blue input layer illustrated in Sidebar 6.

The next question of concern to a computer scientist is:

- *How can one execute a step of computation using DNA tiles?*

To execute steps of computation, the TX tiles were designed to have pads at one end that encoded the cumulative XOR value. Also, since the reporter strand segments ran through each such tile, the appropriate input bit was also provided within its structure. These two values implied the opposing pad on the other side of the tile be the XOR of these two bits.

The final question of concern to a computer scientist is:

- *How can one determine and/or display the output values of a DNA tiling computation?*

The output in this case was read by determining which of two possible cut sites (endonuclease cleavage sites) were present at each position in the tile assembly. This was executed by first isolating the reporter strand, then digesting separate aliquots with each endonuclease separately and the two together, finally these samples were examined by gel electrophoresis and the output values were displayed as banding patterns on the gel.

Another method for output (presented below) is the use of AFM observable patterning. The patterning was made by designing the tiles computing a bit 1 to have a stem loop protruding from the top of the tile. The sequence of this molecular patterning was clearly viewable under appropriate AFM imaging conditions.

Although only very simple computations, the experiments of [M00] and [Y03b] did demonstrate for the first time methods for autonomous execution of a sequence of finite-state operations via algorithmic self-assembly, as well as for providing inputs and for outputting the results.

4.2 Autonomous Finite-State Computations Via Disassembly of DNA Nanostructures

An alternative method for autonomous execution of a sequence of finite-state transitions was subsequently developed by [S06]. Their technique essentially operated in the reverse of the assembly methods described above, and instead was based on disassembly. They began with a linear DNA nanostructure whose sequence encoded the inputs, and then they executed series of steps that digested the DNA nanostructure from one end. On each step, a sticky end at one end of the nanostructure encoded the current state, and the finite transition was determined by hybridization of the current sticky end with a small “rule” nanostructure encoding the finite-state transition rule. Then a restriction enzyme, which recognized the sequence encoding the current input as well as the current state, cut the appended end of the linear DNA nanostructure, to expose a new sticky end encoding the next state.

5 Conclusions

Other topics that were originally intended but have been dropped because of the space limitations (see the extended version of this paper at <http://www.cs.duke.edu/~reif/paper/AutonomousDNA/AutonomousDNA.pdf>) are:

- Assembling patterned and addressable 2D DNA lattices, such as attaching materials to DNA [Y03c], and methods for programmable assembly of patterned 2D DNA lattices [Y03a, R06a, R04, P06].
- Autonomous molecular transport devices self-assembled from DNA [Y04, M00].
- Future challenges for self-assembled DNA nanostructures, such as error correction and self-repair at the molecular-scale [R06b], and 3D DNA lattices.

In attempting to understand these modern developments, it is worth recalling that mechanical methods for computation date back to the very onset of computer science, for example to the cog-based mechanical computing machine of Babbage. Lovelace stated in 1843 that Babbage's "Analytical Engine weaves algebraic patterns just as the Jacquard-loom weaves flowers and leaves". In some of the recently demonstrated methods for biomolecular computation, computational patterns were essentially woven into molecular fabric (DNA lattices) via carefully controlled and designed self-assembly processes [Y03a, R04, P06]. We have observed that many of these self-assembly processes are computational-based and programmable, and it seems likely that computer science techniques will be essential to the further development of this emerging field of biomolecular computation.

References

- [A98] Adleman, L.: Computing with DNA. *Scientific American* 279(2), 34–41 (1998)
- [D06] Deng, Z., Chen, Y., Tian, Y., Mao, C.: A fresh look at DNA nanotechnology. In: Chen, J., Jonoska, N., Rozenberg, G. (eds.) *Nanotechnology: Science and Computation, Natural Computing*, pp. 23–34. Springer, Heidelberg (2006)
- [M00] Mao, C., LaBean, T.H., Reif, J.H., Seeman, N.: Logical Computation Using Algorithmic Self-Assembly of DNA Triple-Crossover Molecules. *Nature* 407, 493–495 (2000)
- [R04] Rothmund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic Self-Assembly of DNA Sierpinski Triangles, *PLoS Biology* 2 (12) (December 2004)
- [R06a] Rothmund, P.W.K.: Folding DNA to create nanoscale shapes and patterns. *Nature* 440, 297–302 (2006)
- [R06b] Reif, J.H., Sahu, S., Yin, P.: Compact Error-Resilient Computational DNA Tiling Assemblies. In: John, H., Jonoska, N., Rozenberg, G. (eds.) *Nanotechnology: Science and Computation, Natural Computing*, pp. 79–104. Springer, Heidelberg (2006)
- [S04] Seeman, N.C.: Nanotechnology and the Double Helix. *Scientific American* 290(6), 64–75 (2004)
- [S06] Shapiro, E., Benenson, Y.: Bringing DNA Computers to Life. *Scientific American*, 45–51 (May 2006)

- [Y03a] Yan, H., LaBean, T.H., Feng, L., Reif, J.H.: Directed Nucleation Assembly of Barcode Patterned DNA Lattices. *PNAS* 100(14), 8103–8108 (2003)
- [Y03b] Yan, H., Feng, L., LaBean, T.H., Reif, J.: DNA Nanotubes, Parallel Molecular Computations of Pairwise Exclusive-Or (XOR) Using DNA String Tile. Self-Assembly in *Journal of American Chemistry Society(JACS)* 125(47), 14246–14247 (2003)
- [Y03c] Yan, H., Park, S.H., Finkelstein, G., Reif, J.H., LaBean, T.H.: DNA-Templated Self-Assembly of Protein Arrays and Highly Conductive Nanowires. *Science* 301, 1882–1884 (2003)
- [Y04] Yin, P., Yan, H., Daniel, X.G., Turberfield, A.J., Reif, J.H.: A Unidirectional DNA Walker Moving Autonomously Along a Linear Track. *Angewandte Chemie [International Edition]* 43(37), 4906–4911 (2004)

Interval Valued QL-Implications

R.H.S. Reiser¹, G.P. Dimuro¹, B.C. Bedregal², and R.H.N. Santiago²

¹ Programa de Pós-Graduação em Informática, Universidade Católica de Pelotas

Rua Felix da Cunha 412, 96010-000 Pelotas, Brazil

{reiser,liz}@ucpel.tche.br

² Depto de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte

Campus Universitário s/n, 59072-970 Natal, Brazil

{bedregal,regivan}@dimap.ufrn.br

Abstract. The aim of this work is to analyze the interval canonical representation for fuzzy QL-implications and automorphisms. Intervals have been used to model the uncertainty of a specialist's information related to truth values in the fuzzy propositional calculus: the basic systems are based on interval fuzzy connectives. Thus, using subsets of the real unit interval as the standard sets of truth degrees and applying continuous t-norms, t-conorms and negation as standard truth interval functions, the standard truth interval function of an QL-implication can be obtained. Interesting results on the analysis of interval canonical representation for fuzzy QL-implications and automorphisms are presented. In addition, commutative diagrams are used in order to understand how an interval automorphism acts on interval QL-implications, generating other interval fuzzy QL-implications.

1 Introduction

The tolerance for imprecision and the ability to make decisions under uncertainties of control systems provide the seminal motivation for development of fuzzy set theory. Under its mathematical approach, the fuzzy logic, concerned with the narrow sense of fuzzy approach, has been developed as a system of formal deductive systems with a comparative notion of truth to formalize deduction under vagueness. Thus, fuzzy logic gives foundations for approximate reasoning using imprecise propositions based on fuzzy set theory. And, all these matters involve the knowledge of the basic characteristics of approximative reasoning processes response from the viewpoint of uncertainty or incomplete available information and knowledge of the parameters that affect human reasoning and need to be subjected to scientific investigation.

The extension of classical logic connectives to the real unit interval is fundamental for the studies on fuzzy logic and therefore are essential to the development of fuzzy systems. This extension must preserve the behaviors of the connectives at the interval endpoints (crisp values) and important properties, such as commutativity and associativity, which results in the notions of triangular norms and triangular conorms. Fuzzy implications play an important role in fuzzy logic, both in the broad sense (heavily

applied to fuzzy control, analysis of vagueness in natural language and techniques of soft-computing) and in the narrow sense (developed as a branch of many-valued logic which is able to investigate deep logical questions). However, there is no consensus among researchers which extra properties fuzzy implications should satisfy. In the literature, several fuzzy implication properties have already been considered and their interrelationship with the other kinds of connectives are generally presented. There exist three main classes of fuzzy implications associated to fuzzy connectives, namely R-implications (generated from t-norms), S-implications (generated from t-conorms and fuzzy negations) and QL-implications, which are generated from t-norms, t-conorms joint with fuzzy negations.

In another formal approach, the correctness and optimality of interval mathematics have been applied in technological and scientific computations to provide accuracy of calculations together with the automatic and rigorous control of the errors that occur in numerical computations [1]. In this sense, interval computation is adequate to deal with the imprecision of the input values or imprecision caused by the rounding errors which occur during the computation [2,3].

Interval mathematics is another form of information theory which is related to but independent from fuzzy logic. However, when intervals can be considered as a particular type of fuzzy set or when the interval membership degree is considered to be an imprecision in the belief degree of a specialist, it seems natural and interesting to deal with the interval fuzzy approach.¹

Among several papers connecting these areas [6,9,10,11,12,13,14], we consider Bédregal and Takahashi's [15,16] work, which has provided interval extensions for the fuzzy connectives considering both correctness (accuracy) and optimality aspects, as properly shown in [17].

This paper is organized as follows. In Sect. 2, we consider the conditions to obtain the interval representation of a real function and recall the related definition of best interval. The interval extension of fuzzy t-conorms (t-norms) and fuzzy negations are presented in sections 3 and 4, respectively. Further analysis of properties met by fuzzy QL-implications are studied in Sect. 5. Section 6 shows that minimal properties of fuzzy implications may be extended from interval fuzzy degrees, in a natural way. In Sect. 6.1, commutative classes of interval fuzzy QL-implications are discussed. In Sect 7, we show how the initial uncertainties of the membership degrees are propagated by the best interval representations of QL-implications. The action of an interval automorphism on interval QL-implication is analyzed in Sect. 8. The canonical construction of an automorphism of an interval automorphism is introduced in Sect. 8.1, including its best interval representation (Sect. 8.2) and the relation between interval implications and automorphism (Sect. 8.3). Section 9 is the Conclusion.

¹ Interval membership degrees are useful to map the uncertainty and the difficulty of an expert in determining which is the fairest membership degree for an element with respect to a linguistic term. In this case, the radius of the interval indicates a maximal bound for the error [4,5,6], providing an estimation of that uncertainty. Interval degrees also can summarize the opinion of several experts with respect to the exact membership degree for an element with respect to a linguistic term. In this case, the left and right interval endpoints are, respectively, the least and the greatest degrees provided by a group of experts [6,7,8,9].

2 Interval Representations

Consider the real unit interval $U = [0, 1] \subseteq \mathbb{R}$. Let $\mathbb{U} = \{[a, b] \mid 0 \leq a \leq b \leq 1\}$ be the set of subintervals of U . The interval set has two projections $l, r : \mathbb{U} \rightarrow U$ defined by $l([a, b]) = a$ and $r([a, b]) = b$, respectively. As a convention, for each $X \in \mathbb{U}$ the projections $l(X)$ and $r(X)$ will also be denoted by \underline{X} and \overline{X} , respectively.

Several natural partial orders may be defined on \mathbb{U} [18]. The most used orders in the context of interval mathematics and considered in this work, are the following:

1. *Product*: $X \leq Y$ if and only if $\underline{X} \leq \underline{Y}$ and $\overline{X} \leq \overline{Y}$.
2. *Inclusion order*: $X \subseteq Y$ if and only if $\underline{X} \geq \underline{Y}$ and $\overline{X} \leq \overline{Y}$.

Since these orders are partial, not all pairs of intervals present maximum and minimum, but there are always an infimum and a supremum. For the case of the product order, it holds that:

$$\begin{aligned} \inf\{X, Y\} &= [\min\{\underline{X}, \underline{Y}\}, \min\{\overline{X}, \overline{Y}\}] \\ \sup\{X, Y\} &= [\max\{\underline{X}, \underline{Y}\}, \max\{\overline{X}, \overline{Y}\}]. \end{aligned}$$

Let X and Y be interval representations of a real number α , that is, $\alpha \in X$ and $\alpha \in Y$. X is a better representation of α than Y if X is narrower than Y , that is, if $X \subseteq Y$.

Definition 1. A function $F : \mathbb{U}^n \rightarrow \mathbb{U}$ is an **interval representation** of a function $f : U^n \rightarrow U$ if, for each $\mathbf{X} \in \mathbb{U}^n$ and $\mathbf{x} \in \mathbf{X}$, $f(\mathbf{x}) \in F(\mathbf{X})$ [17]².

An interval function $F : \mathbb{U}^n \rightarrow \mathbb{U}$ is a **better interval representation** of $f : U^n \rightarrow U$ than $G : \mathbb{U}^n \rightarrow \mathbb{U}$, denoted by $G \subseteq F$, if, for each $\mathbf{X} \in \mathbb{U}^n$, $F(\mathbf{X}) \subseteq G(\mathbf{X})$.

2.1 Best Interval Representation

Definition 2. According to [17], the **best interval representation** of a real function $f : U^n \rightarrow U$, is the interval function $\hat{f} : \mathbb{U}^n \rightarrow \mathbb{U}$ defined by

$$\hat{f}(\mathbf{X}) = [\inf\{f(\mathbf{x}) : \mathbf{x} \in \mathbf{X}\}, \sup\{f(\mathbf{x}) : \mathbf{x} \in \mathbf{X}\}]. \quad (1)$$

The interval function \hat{f} is well defined and for any other interval representation F of f , $F \subseteq \hat{f}$. The interval function \hat{f} returns a narrower interval than any other interval representation of f . Thus, \hat{f} has the *optimality property* of interval algorithms mentioned by Hickey et al. [1], when it is seen as an algorithm to compute a real function f .

Remark 1. The best interval representation of a function $f : U^n \rightarrow U$ coincides with the range of f , that is, for each $\mathbf{X} \in \mathbb{U}^n$, $\hat{f}(\mathbf{X}) = \{f(\mathbf{x}) : \mathbf{x} \in \mathbf{X}\} = f(\mathbf{X})$, if and only if f is continuous in the usual sense [17].

The main result in [17] can be adapted to our context, for U^n instead of \mathbb{R} , as shown below:

Theorem 1. For a function $f : U^n \rightarrow U$, the following statements are equivalent: (i) f is continuous; (ii) \hat{f} is Scott continuous; (iii) \hat{f} is Moore continuous.

² Other authors, e.g., [19,20,21], also consider this definition but with other name and purpose.

3 Interval t-norm and Interval t-conorm

Based on the main contribution of the interval generalization proposed in [15], an interval triangular conorm (norm), *t-conorm (t-norm)* for short, may be considered as an interval representation of a t-conorm (t-norm). This generalization fits with the fuzzy principle, which means that the interval membership degree may be thought as an approximation of the exact degree [6,7,8,9,10].

Notice that a t-conorm (t-norm) is a function $S(T) : U^2 \rightarrow U$ which is commutative, associative, monotonic and has 0 (1) as neutral element.

Example 1. Typical examples of t-norms and t-conorms, respectively, are the following:

1. Minimum and maximum: $T_M(x, y) = \min\{x, y\}$ and $S_M(x, y) = \max\{x, y\}$;
2. Product and probabilistic sum: $T_P(x, y) = xy$ and $S_P(x, y) = x + y - xy$;
3. Drastic product and drastic sum:

$$T_D(x, y) = \begin{cases} 0 & \text{if } x < 1 \text{ and } y < 1 \\ \min\{x, y\} & \text{otherwise;} \end{cases}$$

$$S_D(x, y) = \begin{cases} 1 & \text{if } 0 < x \text{ and } 0 < y \\ \max\{x, y\} & \text{otherwise.} \end{cases}$$

In the following, an extension of the t-conorm (t-norm) notion for \mathbb{U} is considered [15,16].

Definition 3. A function $\mathbb{S} : \mathbb{U}^2 \rightarrow \mathbb{U}$ is an **interval t-conorm (t-norm)** if it is commutative, associative, monotonic with respect to the product and inclusion order and $[0, 0]$ ($[1, 1]$) is the neutral element.

Proposition 1. If $S(T)$ is a t-conorm (t-norm) then $\widehat{S} : \mathbb{U}^2 \rightarrow \mathbb{U}$ ($\widehat{T} : \mathbb{U}^2 \rightarrow \mathbb{U}$) is an interval t-conorm (t-norm). Characterizations of \widehat{S} and \widehat{T} are given, respectively, by:

$$\widehat{S}(X, Y) = [S(\underline{X}, \underline{Y}), S(\overline{X}, \overline{Y})], \quad (2)$$

$$\widehat{T}(X, Y) = [T(\underline{X}, \underline{Y}), T(\overline{X}, \overline{Y})]. \quad (3)$$

Example 2. The best interval representations of the typical t-norms and t-conorms given in Example 1, are, respectively, the following:

1. $\widehat{T}_M(X, Y) = \inf\{X, Y\}$ and $\widehat{S}_M(x, y) = \sup\{X, Y\}$;
2. $\widehat{T}_P(X, Y) = [\underline{XY}, \overline{XY}]$ and $\widehat{S}_P(X, Y) = [\underline{X} + \underline{Y} - \underline{XY}, \overline{X} + \overline{Y} - \overline{XY}]$;
- 3.

$$\widehat{T}_D(X, Y) = \begin{cases} [0, 0] & \text{if } \overline{X} < 1 \text{ and } \overline{Y} < 1 \\ \inf\{X, Y\} & \text{if } \sup\{X, Y\} = [1, 1] \\ [0, \min\{\overline{X}, \overline{Y}\}] & \text{otherwise;} \end{cases}$$

$$\widehat{S}_D(X, Y) = \begin{cases} [1, 1] & \text{if } 0 < \underline{X} \text{ and } 0 < \underline{Y} \\ \sup\{X, Y\} & \text{if } \inf\{X, Y\} = [0, 0] \\ [\max\{\underline{X}, \underline{Y}\}, 1] & \text{otherwise.} \end{cases}$$

An interpretation for the case of \widehat{T}_D is that this interval t-norm always returns the least possible interval, with respect to the product order, and, therefore, it is the least interval t-norm.

4 Interval Fuzzy Negation

A function $N : U \rightarrow U$ is a *fuzzy negation* if

N1: $N(0) = 1$ and $N(1) = 0$;

N2: If $x \geq y$ then $N(x) \leq N(y)$, $\forall x, y \in I$.

In addition, fuzzy negations satisfying the involutive property are called *strong fuzzy negations* [22,23]:

N3: $N(N(x)) = x$, $\forall x \in U$.

Example 3. A typical example of a strong fuzzy negation is the complement, that is, the function $N_C(x) = 1 - x$.

When the t-norm is considered, it is also possible to establish a partial order on fuzzy negations in a natural way. Considering two fuzzy negations N_1 and N_2 , define:

$$N_1 \leq N_2 \text{ if, for each } x \in U, N_1(x) \leq N_2(x).$$

Remark 2. If $N_1 \leq N_2$ and $x \geq y$ then $N_1(x) \leq N_2(y)$.

Definition 4. An interval function $\mathbb{N} : \mathbb{U} \rightarrow \mathbb{U}$ is an *interval fuzzy negation* if, for any X, Y in \mathbb{U} , the following properties hold:

N1 : $\mathbb{N}([0, 0]) = [1, 1]$ and $\mathbb{N}([1, 1]) = [0, 0]$;

N2 : If $X \geq Y$ then $\mathbb{N}(X) \leq \mathbb{N}(Y)$;

N3 : If $X \subseteq Y$ then $\mathbb{N}(X) \subseteq \mathbb{N}(Y)$.

If \mathbb{N} also meets the involutive property, it is said to be a *strong interval fuzzy negation*:

N4 : $\mathbb{N}(\mathbb{N}(X)) = X$, $\forall X \in \mathbb{U}$.

Let $N : U \rightarrow U$ be a fuzzy negation. A characterization of \widehat{N} is given by:

$$\widehat{N}(X) = [N(\overline{X}), N(\underline{X})]. \quad (4)$$

Example 4. The best interval representation of the strong negation N_C , given in Example 3, is $\widehat{N}_C(X) = [1 - \overline{X}, 1 - \underline{X}]$.

Theorem 2. Let $N : U \rightarrow U$ be a (strong) fuzzy negation. Then \widehat{N} is a (strong) interval fuzzy negation.

5 Fuzzy Implication

Several definitions for fuzzy implication together with related properties have been given (see, e.g., [23,24,25,26,27,28,29,30,31,32,33]). The unique consensus in these definitions is that the fuzzy implication should have the same behavior as the classical implication for the crisp case. Thus, a binary function $I : U^2 \rightarrow U$ is a *fuzzy implication* if I meets the minimal boundary conditions:

$$I(1, 1) = I(0, 1) = I(0, 0) = 1 \text{ and } I(1, 0) = 0.$$

Several reasonable extra properties may be required for fuzzy implications. The only two properties considered in this paper are the following:

I1: If $y \leq z$ then $I(x, y) \leq I(x, z)$ (second place isotonicity);

I2: $I(1, x) = x$, (left neutrality principle);

5.1 QL-Implication

Let S be a t-conorm, N be a strong fuzzy negation and T be a t-norm. A fuzzy implication, called QL-implication, is defined by

$$I_{S,N,T}(x, y) = S(N(x), T(x, y)), \forall (x, y) \in [0, 1]. \quad (5)$$

Example 5. The QL-implications defined from the t-norms and the t-conorms given in Example 1 and the strong negation presented in Example 3 are the following:

1.

$$I_{T_M, S_M, N_C}(x, y) = \max\{1 - x, \min\{x, y\}\} = \begin{cases} 1 - x & \text{if } x \leq 0.5 \\ \min\{x, y\} & \text{if } \min\{x, y\} \geq 0.5 \\ \max\{1 - x, y\} & \text{otherwise.} \end{cases}$$

2. $I_{T_P, S_P, N_C}(x, y) = 1 + x^2y - x.$

3.

$$I_{T_D, S_D, N_C}(x, y) = \begin{cases} 1 & \text{if } y = 1; \\ y & \text{if } x = 1 \text{ and } y < 1; \\ 1 - x & \text{otherwise.} \end{cases}$$

Based on Sect. 3.3. in [34], a result relating the QL-implications presented above with the properties **I1** and **I2** is seen in the following proposition.

Proposition 2. *Let $I : U^2 \rightarrow U$ be a fuzzy implication. If I is an QL-implication then the properties **I1** and **I2** hold.*

6 Interval Fuzzy Implication

According to the idea that values in interval mathematics are identified with degenerate intervals, the minimal properties of fuzzy implications can be naturally extended from interval fuzzy degrees, when the respective degenerate intervals are considered. Thus, a function $\mathbb{I} : \mathbb{U}^2 \rightarrow \mathbb{U}$ is an *interval fuzzy implication* if the following conditions hold:

$$\mathbb{I}([1, 1], [1, 1]) = \mathbb{I}([0, 0], [0, 0]) = \mathbb{I}([0, 0], [1, 1]) = [1, 1] \text{ and } \mathbb{I}([1, 1], [0, 0]) = [0, 0].$$

The two extra properties of fuzzy implication seen in the previous section can be naturally extended.

- $\mathbb{I}1$: If $Y \leq Z$ then $\mathbb{I}(X, Y) \leq \mathbb{I}(X, Z)$;
 $\mathbb{I}2$: $\mathbb{I}([1, 1], X) = X$.

Starting from any fuzzy implication I , it is always possible to obtain canonically an interval fuzzy implication \mathbb{I} . Then, the interval fuzzy implication \mathbb{I} also meets the optimality property and preserves the same properties satisfied by the fuzzy implication I . In the following two propositions, the best interval representation \hat{I} of fuzzy implication I is shown as an inclusion-monotonic function in both arguments. The related proofs are constructed straightforwardly from the definition of the interval fuzzy implication \hat{I} . See [16].

Proposition 3. *If I is a fuzzy implication then \hat{I} is an interval fuzzy implication.*

We can recover the original fuzzy implication from its best interval representation.

Proposition 4. *If I is a fuzzy implication then, for each $x, y \in U$, it holds that $I(x, y) = l(\hat{I}([x, x], [y, y])) = r(\hat{I}([x, x], [y, y]))$.*

Proposition 5. *Let I be a fuzzy implication. For each $X_1, X_2, Y_1, Y_2 \in \mathbb{U}$, if $X_1 \subseteq X_2$ and $Y_1 \subseteq Y_2$ then $\hat{I}(X_1, Y_1) \subseteq \hat{I}(X_2, Y_2)$.*

Proof. If $X_1 \subseteq X_2$ and $Y_1 \subseteq Y_2$, then, since $I(X, Y) = \{I(x, y) : x \in X \text{ and } y \in Y\}$, it holds that $I(X_1, Y_1) \subseteq I(X_2, Y_2)$. So, we have that $\inf I(X_1, Y_1) \geq \inf I(X_2, Y_2)$ and $\sup I(X_1, Y_1) \leq \sup I(X_2, Y_2)$. Thus, since $\hat{I}(X, Y) = [\inf I(X, Y), \sup I(X, Y)]$, it holds that $\hat{I}(X_1, Y_1) \subseteq \hat{I}(X_2, Y_2)$.

Theorem 3. *Let I be a fuzzy implication. I satisfies the property $\mathbb{I}1$ ($\mathbb{I}2$) iff \hat{I} satisfies the property $\mathbb{I}1$ ($\mathbb{I}2$).*

Proof. (\Rightarrow) If I satisfies $\mathbb{I}1$, then, for each $X, Y, Z \in \mathbb{U}$, $\hat{I}(X, Y) = [\inf\{I(x, \underline{Y}) : x \in X\}, \sup\{I(x, \overline{Y}) : x \in X\}]$ and $\hat{I}(X, Z) = [\inf\{I(x, \underline{Z}) : x \in X\}, \sup\{I(x, \overline{Z}) : x \in X\}]$. Therefore, if $Y \leq Z$, then $\hat{I}(X, Y) \leq \hat{I}(X, Z)$ and, thus, \hat{I} satisfies $\mathbb{I}1$.

If I satisfies $\mathbb{I}2$, then, for each $x \in X$, $I(1, x) = x$ and, thus, $\{I(1, x) : x \in X\} = X$. Thus, since $\hat{I}([1, 1], X)$ is the narrowest interval containing $\{I(1, x) : x \in X\}$, then $\hat{I}([1, 1], X) = X$ and so \hat{I} satisfies $\mathbb{I}2$.

(\Leftarrow) Consider $y, z \in U$ such that $y \leq z$. Then, it holds that $[y, y] \leq [z, z]$. Thus, if \hat{I} satisfies $\mathbb{I}1$, then, for each $x \in \mathbb{U}$, $\hat{I}([x, x], [y, y]) \leq \hat{I}([x, x], [z, z])$ and, therefore, $l(\hat{I}([x, x], [y, y])) \leq l(\hat{I}([x, x], [z, z]))$. Hence, by Prop. 4, $I(x, y) \leq I(x, z)$ and, thus, I satisfies $\mathbb{I}1$.

If \hat{I} satisfies $\mathbb{I}2$, then for each $x \in U$, $\hat{I}([1, 1], [x, x]) = [x, x]$. So, from Prop. 4, it holds that $I(1, x) = l(\hat{I}([1, 1], [x, x])) = l([x, x]) = x$. Therefore, I satisfies $\mathbb{I}2$.

6.1 Interval QL-Implication

An interval fuzzy implication \mathbb{I} is an **interval QL-implication** if there are an interval t-conorm \mathbb{S} , a strong interval fuzzy negation \mathbb{N} and an interval t-norm \mathbb{T} such that

$$\mathbb{I}_{\mathbb{S}, \mathbb{N}, \mathbb{T}}(X, Y) = \mathbb{S}(\mathbb{N}(X), \mathbb{T}(X, Y)). \quad (6)$$

Theorem 4. Let S be a t -conorm, N be a fuzzy negation and T be a t -norm. If S , T and N are continuous then

$$\mathbb{I}_{\widehat{S}, \widehat{N}, \widehat{T}} = \widehat{I_{S,N,T}}. \quad (7)$$

Proof. Considering $X, Y \in \mathbb{U}$, one has that

$$\begin{aligned} \mathbb{I}_{\widehat{S}, \widehat{N}, \widehat{T}}(X, Y) &= \widehat{S}(\widehat{N}(X), \widehat{T}(X, Y)) && \text{by equation (6)} \\ &= \widehat{S}([N(\underline{X}), N(\overline{X})], [T(\underline{X}, \underline{Y}), T(\overline{X}, \overline{Y})]) && \text{by equations (4) and (3)} \\ &= [S(N(\underline{X}), T(\underline{X}, \underline{Y})), S(N(\overline{X}), T(\overline{X}, \overline{Y}))] && \text{by equation (2)} \end{aligned}$$

Since, for each $x \in X$ and $y \in Y$,

$$S(N(\overline{X}), T(\underline{X}, \underline{Y})) \leq S(N(x), T(x, y)) \leq S(N(\underline{X}), T(\overline{X}, \overline{Y})),$$

then, $\widehat{I_{S,N,T}}(X, Y) \subseteq \mathbb{I}_{\widehat{S}, \widehat{N}, \widehat{T}}(X, Y)$.

On the other hand, if $z \in \widehat{S}(\widehat{N}(X), \widehat{T}(X, Y))$, then, by the continuity of N and Remark 1, there exist $z_1 \in \widehat{N}(X)$ and $z_2 \in \widehat{T}(X, Y)$ such that $S(z_1, z_2) = z$. Thus, by the continuity of N and T , and Remark 1, there exist $z_{1a} \in X$, $z_{2a} \in X$ and $z_{2b} \in Y$ such that $N(z_{1a}) = z_1$ and $T(z_{2a}, z_{2b}) = z_2$ and, thus, $S(N(z_{1a}), T(z_{2a}, z_{2b})) = z$. If $z_{1a} \leq z_{2a}$, then one has that $S(N(z_{1a}), T(z_{1a}, z_{2b})) \leq z$, or if $z_{2a} \leq z_{1a}$, then it is valid that $S(N(z_{2a}), T(z_{2a}, z_{2b})) \geq z$. So, $z \in [I_{S,N,T}(z_{1a}, z_{2b}), I_{S,N,T}(z_{2a}, z_{2b})] \subseteq \{I_{S,N,T}(x, y) : x \in X \text{ and } y \in Y\} \subseteq \widehat{I_{S,N,T}}(X, Y)$. Therefore, $\mathbb{I}_{\widehat{S}, \widehat{N}, \widehat{T}}(X, Y) = \widehat{S}(\widehat{N}(X), \widehat{T}(X, Y)) \subseteq \widehat{I_{S,N,T}}(X, Y)$.

Corollary 1. If I is a QL-Implication then \widehat{I} is an interval QL-implication.

Proof. It is straightforward, following from Theorem 4.

Example 6. The best interval representation of the QL-implications given in Example 5 are the following. Denote $\widehat{I_{S_M, N_C, T_M}}$, $\widehat{I_{S_P, N_C, T_P}}$ and $\widehat{I_{S_D, N_C, T_D}}$ by \mathbb{I}_M , \mathbb{I}_P and \mathbb{I}_D , respectively.

1.

$$\mathbb{I}_M(X, Y) = \begin{cases} 1 - X & \text{if } \overline{X} \leq 0.5 \text{ or } (X \ll 0.5 \text{ and } \overline{Y} < 0.5); \\ \inf\{X, Y\} & \text{if } 0.5 < \underline{X}; \\ [\min\{1 - \underline{X}, \overline{X}\}, \max\{1 - \underline{X}, \overline{X}\}] & \text{if } X \ll 0.5 \text{ and } \max\{1 - \underline{X}, \overline{X}\} < \underline{Y}; \\ [\min\{\underline{Y}, 1 - \overline{X}\}, \overline{X}] & \text{if } X \ll 0.5 \text{ and } X \subseteq Y; \\ [\min\{\underline{Y}, 1 - \underline{X}\}, \overline{Y}] & \text{if } X \ll 0.5 \text{ and } Y \subseteq X. \end{cases}$$

where $X \ll 0.5$ means that $0.5 \in X$, but $\underline{X} \neq 0.5$ and $\overline{X} \neq 0.5$.

$$2. \mathbb{I}_P(X, Y) = [1 + \overline{X}^2 \underline{Y} - \overline{X}, 1 + \underline{X}^2 \overline{Y} - \underline{X}].$$

$$3. \mathbb{I}_D(X, Y) = [I_1(X, Y), I_2(X, Y)] \text{ where}$$

$$I_1(X, Y) = \begin{cases} 0 & \text{if } (1 \in X \text{ and } \underline{Y} < 1 \text{ and } \underline{X} < 1) \\ \min\{\underline{Y}, 1 - \overline{X}\} & \text{if } 1 \notin X \text{ and } \underline{Y} < 1 \\ \underline{Y} & \text{if } X = [1, 1] \text{ and } \underline{Y} < 1 \\ 1 & \text{if } Y = [1, 1], \end{cases}$$

$$I_2(X, Y) = \begin{cases} 1 & \text{if } 0 \in X \text{ or } 1 \in Y \\ \max\{\overline{Y}, 1 - \underline{X}\} & \text{if } 1 \in X, \underline{Y} < 1 \\ 1 - \underline{X} & 1 \notin X. \end{cases}$$

Proposition 6. Let \mathbb{I} be an interval fuzzy implication. If \mathbb{I} is an interval QL-implication then the properties $\mathbb{I}1$ and $\mathbb{I}2$ hold.

Proof. Consider \mathbb{I} as an interval QL-implication and $X, Y, Z \in \mathbb{U}$.

$\mathbb{I}1$: Based on the monotonicity of the interval t-norm \mathbb{T} and the interval t-conorm \mathbb{S} , if $Y \leq Z$ then $\mathbb{S}(\mathbb{N}(X), \mathbb{T}(X, Y)) \leq \mathbb{S}(\mathbb{N}(X), \mathbb{T}(X, Z))$. Therefore, $\mathbb{I}_{\mathbb{S}, \mathbb{N}, \mathbb{T}}(X, Y) \leq \mathbb{I}_{\mathbb{S}, \mathbb{N}, \mathbb{T}}(X, Z)$.

$\mathbb{I}2$: $\mathbb{I}_{\mathbb{S}, \mathbb{N}, \mathbb{T}}([1, 1], X) = \mathbb{S}(\mathbb{N}([1, 1]), \mathbb{T}([1, 1], X)) = \mathbb{S}([0, 0], X) = X$.

Denote by $\mathcal{C}(S)$, $\mathcal{C}(T)$, $\mathcal{C}(N)$ and $\mathcal{C}(I)$ the classes of continuous t-conorms, continuous t-norms, strong fuzzy negations and continuous QL-implications, respectively. The related interval extensions are indicated by $\mathcal{C}(\mathbb{S})$, $\mathcal{C}(\mathbb{T})$, $\mathcal{C}(\mathbb{N})$ and $\mathcal{C}(\mathbb{I})$, respectively. The results in Sect. 6.1 together with Theorem 4 state the commutativity of the diagram in Fig. 1.

$$\begin{array}{ccc} \mathcal{C}(S) \times \mathcal{C}(N) \times \mathcal{C}(T) & \xrightarrow{eq(5)} & \mathcal{C}(I) \\ \downarrow (eq(2), eq(4), eq(3)) & & \downarrow eq(7) \\ \mathcal{C}(\mathbb{S}) \times \mathcal{C}(\mathbb{N}) \times \mathcal{C}(\mathbb{T}) & \xrightarrow{eq(6)} & \mathcal{C}(\mathbb{I}) \end{array}$$

Fig. 1. Commutative classes of interval implications

7 Propagation of an Interval Uncertainty in Interval QL-Implications

In this section, we show how the initial uncertainties of the membership degrees are propagated by the best interval representations of QL-implications, analyzing the interval QL-implication \mathbb{I}_M presented in Example 6. Observe that the measure of the uncertainty of an interval X is given by its *width* (ou *diameter*), defined as $width(X) = \overline{X} - \underline{X}$.

If $\overline{X} \leq 0.5$, one has that $\mathbb{I}_M(X, Y) = 1 - X$, and, therefore, the measure of the uncertainty of $\mathbb{I}_M(X, Y)$ is given by the width of X . Observe that the width of Y does not have any influence in the result.

When $0.5 < \underline{X}$, the measure of the uncertainty of $\mathbb{I}_M(X, Y)$ is given by the width of the interval $Z = \inf\{X, Y\}$, which is always smaller or equal than $width(X)$, or $width(Y)$, or both.

If $X \ll 0.5$ and $0.5 < \underline{Y}$, then one has that $\mathbb{I}_M(X, Y) = 1 - X$, and, therefore, the measure of the uncertainty of $\mathbb{I}_M(X, Y)$ is given by the width of X .

If $X \ll 0.5$ and $X \subseteq Y$ (or $Y \subseteq X$), then the measure of the uncertainty of $\mathbb{I}_M(X, Y)$ is between the widths of X and Y .

If X and Y are degenerate intervals, then the best interval representation of a QL-implication also returns a degenerate interval.

In the case of \mathbb{I}_P , the uncertainty always increases, or, in the best case, it remains the same (for example, for degenerate intervals).

Summarizing, the propagation of the uncertainty in the case of a multi-level reasoning process using interval QL-implication depends on the particular case, that is, for some interval QL-implications it always either increases or remains the same, but, for other cases, it may always either diminish or remain the same. However, for the majority, the uncertainty will increase for some cases and diminish for other cases.

8 Interval Automorphism

Definition 5. A mapping $\rho : U \longrightarrow U$ is an **automorphism** if it is bijective and monotonic: $x \leq y \Rightarrow \rho(x) \leq \rho(y)$ [35,36].

An equivalent definition is given in [23], where $\rho : U \longrightarrow U$ is an automorphism if it is a continuous and strictly increasing function such that $\rho(0) = 0$ and $\rho(1) = 1$. $Aut(U)$ denotes the set of all automorphisms on U . Automorphisms are closed under composition: if ρ and ρ' are automorphisms then $\rho \circ \rho'(x) = \rho(\rho'(x))$ is also an automorphism. The inverse of an automorphism is also an automorphism. Let ρ be an automorphism and I be a fuzzy implication. The **action of ρ on I** , denoted by I^ρ , defined as follows

$$I^\rho(x, y) = \rho^{-1}(I(\rho(x), \rho(y))) \quad (8)$$

is a fuzzy implication. Moreover, if I is an QL-implication then I^ρ is also an QL-implication.

8.1 Canonical Construction of an Interval Automorphism

A mapping $\varrho : \mathbb{U} \longrightarrow \mathbb{U}$ is an **interval automorphism** if it is bijective and monotonic w.r.t. the product order [37,38], that is, $X \leq Y$ implies that $\varrho(X) \leq \varrho(Y)$. The set of all interval automorphisms on \mathbb{U} is denoted by $Aut(\mathbb{U})$. In what follows, the equation (9) provides a canonical construction of interval automorphisms from automorphisms and, therefore, a bijection between the sets $Aut(U)$ and $Aut(\mathbb{U})$. See Theorem 3 of [37].

Theorem 5. Let $\varrho : \mathbb{U} \longrightarrow \mathbb{U}$ be an interval automorphism. Then there exists an automorphism $\rho : U \longrightarrow U$ such that

$$\varrho(X) = [\rho(\underline{X}), \rho(\overline{X})]. \quad (9)$$

8.2 The Best Interval Representation of an Automorphism

Interval automorphisms can be generated as best interval representations of automorphisms. The following results were proved in [15].

Theorem 6 (Automorphism representation theorem). Let $\rho : U \rightarrow U$ be an automorphism. $\widehat{\rho}$ is an interval automorphism and its characterization is given by

$$\widehat{\rho}(X) = [\rho(\underline{X}), \rho(\overline{X})]. \quad (10)$$

Corollary 2. Consider $\varrho : \mathbb{U} \rightarrow \mathbb{U}$. ϱ is an interval automorphism iff there is an automorphism $\rho : U \rightarrow U$, such that $\varrho = \widehat{\rho}$.

Proof. It is straightforward, following from Theorems 5 and 6.

Thus, each interval automorphism is the best interval representation of some automorphism. Notice that t-conorms were required by definition to satisfy \subseteq -monotonicity. Nevertheless, this property was not required by the definition of interval automorphism. As described in [15], interval automorphisms are \subseteq -monotonic.

Corollary 3. If ϱ is an interval automorphism then ϱ is inclusion monotonic, that is, if $X \subseteq Y$ then $\varrho(X) \subseteq \varrho(Y)$.

Analogously, considering the definition of automorphism used by [23], we can provide alternative characterizations for interval automorphisms based on the Moore and Scott continuities. Consider $\varrho : \mathbb{U} \rightarrow \mathbb{U}$. ϱ is **strictly increasing** if, for each $X, Y \in \mathbb{U}$, if $X < Y$ (i.e. $X \leq Y$ and $X \neq Y$) then $\varrho(X) < \varrho(Y)$.

Proposition 7. Consider $\varrho : \mathbb{U} \rightarrow \mathbb{U}$. ϱ is an interval automorphism iff ϱ is Moore-continuous, strictly increasing, $\varrho([0, 0]) = [0, 0]$ and $\varrho([1, 1]) = [1, 1]$.

Corollary 4. Let $\varrho : \mathbb{U} \rightarrow \mathbb{U}$ be a Moore-continuous and strictly increasing function such that $\varrho([0, 0]) = [0, 0]$ and $\varrho([1, 1]) = [1, 1]$. Then there exists an automorphism ρ such that $\varrho = \widehat{\rho}$.

The case of Scott-continuity follows the same setting.

Lemma 1. Let ϱ_1 and ϱ_2 be interval automorphisms. Then it holds that $(\varrho_1 \circ \varrho_2)^{-1} = \varrho_2^{-1} \circ \varrho_1^{-1}$.

Lemma 2. Let ρ be an automorphism. Then it holds that

$$\widehat{\rho^{-1}} = \widehat{\rho}^{-1}. \quad (11)$$

In the following theorem, an interval automorphism ϱ acts on an interval fuzzy implication \mathbb{I} , generating a new interval fuzzy implication \mathbb{I}^{ϱ} .

Theorem 7. Let $\varrho : \mathbb{U} \rightarrow \mathbb{U}$ be an interval automorphism and $\mathbb{I} : \mathbb{U}^2 \rightarrow \mathbb{U}$ be an interval implication. Then the mapping $\mathbb{I}^{\varrho} : \mathbb{U}^2 \rightarrow \mathbb{U}$, defined by

$$\mathbb{I}^{\varrho}(X, Y) = \varrho^{-1}(\mathbb{I}(\varrho(X), \varrho(Y))), \quad (12)$$

is an interval implication.

Theorem 8. Let I be a an implication and ρ be an automorphism. Then it holds that $\widehat{I^{\rho}} = \widehat{I}^{\widehat{\rho}}$.

Proof. Considering $X, Y \in \mathbb{U}$, one has that

$$\begin{aligned}
 \widehat{I}^\rho(X, Y) &= [\inf\{I^\rho(x, y) : x \in X \text{ and } y \in Y\}, \\
 &\quad \sup\{I^\rho(x, y) : x \in X \text{ and } y \in Y\}] && \text{by equation (1)} \\
 &= [\inf\{\rho^{-1}(I(\rho(x), \rho(y))) : x \in X \text{ and } y \in Y\}, \\
 &\quad \sup\{\rho^{-1}(I(\rho(x), \rho(y))) : x \in X \text{ and } y \in Y\}] && \text{by equation (8)} \\
 &= [\rho^{-1}(\inf\{I(\rho(x), \rho(y)) : x \in X \text{ and } y \in Y\}), \\
 &\quad \widehat{\rho^{-1}(\sup\{I(\rho(x), \rho(y)) : x \in X \text{ and } y \in Y\})}] && \text{by continuity of } \rho^{-1} \\
 &= \widehat{\rho^{-1}([\inf\{I(\rho(x), \rho(y)) : x \in X \text{ and } y \in Y\}, \\
 &\quad \sup\{I(\rho(x), \rho(y)) : x \in X \text{ and } y \in Y\}])} && \text{by equation (8)} \\
 &= \widehat{\rho^{-1}([\inf\{I(x, y) : x \in \rho(X) \text{ and } y \in \rho(Y)\}, \\
 &\quad \sup\{I(x, y) : x \in \rho(X) \text{ and } y \in \rho(Y)\}])} && \text{by continuity of } \rho \\
 &= \widehat{\rho^{-1}([\inf\{I(x, y) : x \in \rho(X) \text{ and } y \in \rho(Y)\}, \\
 &\quad \sup\{I(x, y) : x \in \widehat{\rho}(X) \text{ and } y \in \widehat{\rho}(Y)\}])} && \text{by Remark 1} \\
 &= \widehat{\rho^{-1}(\widehat{I}(\widehat{\rho}(X), \widehat{\rho}(Y)))} && \text{by equation (1)} \\
 &= \widehat{\rho^{-1}(\widehat{I}(\widehat{\rho}(X), \widehat{\rho}(Y)))} && \text{by equation (11)} \\
 &= \widehat{I}^{\widehat{\rho}}(X, Y) && \text{by equation (12)}
 \end{aligned}$$

Proposition 8. Let \mathbb{I} be an interval implication and ϱ_1 and ϱ_2 be interval automorphisms. Then it holds that $(\mathbb{I}^{\varrho_1})^{\varrho_2} = \mathbb{I}^{\varrho_1 \circ \varrho_2}$.

Proposition 9. Let \mathbb{N} be an interval fuzzy negation and ϱ be an interval automorphism. Then it is valid that

$$\mathbb{N}^{\varrho}(X) = \varrho^{-1}(\mathbb{N}(\varrho(X))). \quad (13)$$

In the following, we apply the results presented in the last subsection, concerned with the generation of new interval implications based on the action of interval automorphisms. In particular, we consider the interval extensions of QL-implications.

8.3 Interval Automorphism Acting on Interval QL-Implication

Theorem 9. Let $\varrho : \mathbb{U} \longrightarrow \mathbb{U}$ be an interval automorphism and $\mathbb{I} : \mathbb{U}^2 \longrightarrow \mathbb{U}$ be an interval QL-implication. Then $\mathbb{I}^{\varrho} : \mathbb{U}^2 \longrightarrow \mathbb{U}$ is an interval QL-implication defined by

$$\mathbb{I}_{\mathbb{S}, \mathbb{N}, \mathbb{T}}^{\varrho}(X, Y) = \mathbb{S}^{\varrho}(\mathbb{N}^{\varrho}(X), \mathbb{T}^{\varrho}(X, Y)). \quad (14)$$

Proof. Considering $X, Y \in \mathbb{U}$, we have that

$$\begin{aligned}
 \mathbb{I}_{\mathbb{S}, \mathbb{N}, \mathbb{T}}^{\varrho}(X, Y) &= \varrho^{-1}(\mathbb{I}_{\mathbb{S}, \mathbb{N}, \mathbb{T}}(\varrho(X), \varrho(Y))) && \text{by equation (12)} \\
 &= \varrho^{-1}(\mathbb{S}(\mathbb{N}(\varrho(X)), \mathbb{T}(\varrho(X), \varrho(Y)))) && \text{by equation (6)} \\
 &= \varrho^{-1}(\mathbb{S}(\varrho \circ \varrho^{-1}(\mathbb{N}(\varrho(X))), \varrho \circ \varrho^{-1}(\mathbb{T}(\varrho(X), \varrho(Y))))) \\
 &= \varrho^{-1}(\mathbb{S}(\varrho(\mathbb{N}^{\varrho}(X)), \varrho(\mathbb{T}^{\varrho}(X, Y)))) && \text{by equations (13) and (7) in [15]} \\
 &= \mathbb{S}^{\varrho}(\mathbb{N}^{\varrho}(X), \mathbb{T}^{\varrho}(X, Y))
 \end{aligned}$$

According to Theorem 8, the commutative diagram pictured in Fig. 2 holds.

Based on Theorems 8 and 9, (interval) QL-implications and (interval) automorphisms can be seen as objects and morphisms of the category $\mathfrak{C}(\mathcal{C}(I), \text{Aut}(I))$ ($\mathfrak{C}(\mathcal{C}(\mathbb{I}),$

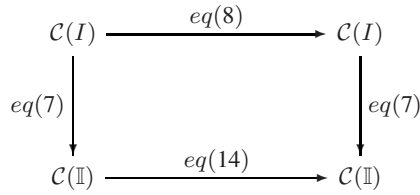


Fig. 2. Commutative classes of interval QL-implications and automorphisms

$Aut(\mathbb{I}))$), respectively. In a categorical approach, the action of an interval automorphism on an interval QL-implication can be conceived as a covariant functor whose application over the QL-implications and automorphisms in $\mathfrak{C}(\mathcal{C}(I), Aut(I))$ returns the related best interval representations in $\mathfrak{C}(\mathcal{C}(\mathbb{I}), Aut(\mathbb{I}))$. Therefore, interval automorphisms can be used to deal with optimality of interval fuzzy algorithms.

9 Conclusion and Final Remarks

This work emphasized that both interval mathematics and fuzzy set theory are firmly integrated with principles of information theory used to underlying logic system for expert systems. Thus, this paper complements the results of previous work [15], extending the generalization of the main properties of interval fuzzy QL-implication and interval automorphisms. The interval extension considers the best interval representation of a real function to deal with the imprecision of a specialist in providing an exact value to measure membership uncertainty.

In this work, intervals are used to model the uncertainty of a specialist's information related to truth values in the fuzzy propositional calculus. The basic systems are based on interval t-conorms. Using subsets of the real unit interval as the standard sets of truth degrees and applying continuous t-conorms and negation as standard truth interval functions, the standard truth interval function of an QL-implication can be obtained.

In addition, we mainly discussed under which conditions generalized fuzzy QL-implications applied to interval values preserve properties of canonical forms generated by interval t-conorms and interval t-norms. It is shown that properties of fuzzy logic may be naturally extended for interval fuzzy degrees considering the respective degenerate intervals. The significance of interval fuzzy QL-implication is emphasized, showing that interval QL-implications can be constructed from interval automorphisms, which are preserved by the interval canonical representation. These results are important not only to analyze deductive systems in mathematical depth but also as foundations of methods based on interval fuzzy logic. They provide the accuracy criteria and the optimality property of computer computations integrated to a formal mathematical theory for the representation of uncertainty, concerned with fuzzy set theory.

Acknowledgments

This work was partially supported by the Brazilian funding agency CNPq (Proc. 470871/2004-0). We are very grateful to the referees for their valuable suggestions that help us to improve the paper.

References

1. Hickey, T., Ju, Q., Emdem, M.: Interval arithmetic: from principles to implementation. *Journal of the ACM* 48(5), 1038–1068 (2001)
2. Moore, R.: *Methods and Applications of Interval Analysis*. SIAM, Philadelphia (1979)
3. Alefeld, G., Herzberger, J.: *Introduction to Interval Computations*. Academic Press, New York (1983)
4. Moore, R.E.: *Interval Arithmetic and Automatic Error Analysis in Digital Computing*. PhD thesis, Stanford University, Stanford (1962)
5. Kearfort, R.B., Kreinovich, V. (eds.): *Applications of Interval Computations*. Kluwer Academic Publishers, Boston (1996)
6. Nguyen, H., Kreinovich, V., Zuo, Q.: Interval-valued degrees of belief: applications of interval computations to expert systems and intelligent control. *International Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems* 5(3), 317–358 (1997)
7. Walley, P.: *Statistical Reasoning with Imprecise Probabilities*. Chapman & Hall, London (1991)
8. Walley, P.: Measures of uncertainty. *Artificial Intelligence* 83, 1–58 (1996)
9. Dubois, D., Prade, H.: Interval-valued fuzzy sets, possibility theory and imprecise probability. In: *Proc. Intl. Conf. Fuzzy Logic and Technology*, Barcelona, pp. 314–319 (2005)
10. Lodwick, W.: Preface. *Reliable Computing* 10(4), 247–248 (2004)
11. Turksen, I.: Fuzzy normal forms. *Fuzzy Sets and Systems* 69, 319–346 (1995)
12. Moore, R., Lodwick, W.: Interval analysis and fuzzy set theory. *Fuzzy Sets and Systems* 135(1), 5–9 (2003)
13. Gasse, B.V., Cornelis, G., Deschrijver, G., Kerre, E.: On the properties of a generalized class of t-norms in interval-valued fuzzy logics. *New Mathematics and Natural Computation* 2, 29–42 (2006)
14. Zadeh, L.A.: The concept of a linguistic variable and its application to approximate reasoning - I. *Information Sciences* 6, 199–249 (1975)
15. Bedregal, B., Takahashi, A.: The best interval representation of t-norms and automorphisms. *Fuzzy Sets and Systems* 157(24), 3220–3230 (2006)
16. Bedregal, B., Takahashi, A.: Interval valued versions of t-conorms, fuzzy negations and fuzzy implications. In: *IEEE Proc. Intl. Conf. on Fuzzy Systems*, Vancouver, Vancouver, pp. 9553–9559. IEEE Computer Society Press, Los Alamitos (2006)
17. Santiago, R., Bedregal, B., Ációly, B.: Formal aspects of correctness and optimality in interval computations. *Formal Aspects of Computing* 18(2), 231–243 (2006)
18. Callejas-Bedregal, R., Bedregal, B.: Intervals as a domain constructor. *TEMA* 2(1), 43–52 (2001), <http://www.sbmec.org.br/tema>
19. Kearfott, R.B.: *Rigorous Global Search: Continuous problems*. Kluwer Academic Publishers, Dordrecht (1996)
20. Caprani, O., Madsen, K., Stauning, O.: Existence test for asynchronous interval iteration. *Reliable Computing* 3(3), 269–275 (1997)
21. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: *Applied Interval Analysis: With examples in parameter and state estimation, robust control and robotic*. Springer, Heidelberg (2001)
22. Klement, E., Mesiar, R., Pap, E.: *Triangular Norms*. Kluwer Academic Publishers, Dordrecht (2000)
23. Bustince, H., Burilo, P., Soria, F.: Automorphism, negations and implication operators. *Fuzzy Sets and Systems* 134, 209–229 (2003)
24. Baczyński, M.: Residual implications revisited. *Notes on the Smets-Magrez*. *Fuzzy Sets and Systems* 145(2), 267–277 (2004)

25. Balasubramaniam, J.: Yager's new class of implications J_f and some classical tautologies. *Information Sciences* (2006)
26. Fodor, J.C.: On fuzzy implication operators. *Fuzzy Sets and Systems* 42, 293–300 (1991)
27. Fodor, J., Roubens, M.: *Fuzzy Preference Modelling and Multicriteria Decision Support*. Kluwer Academic Publishers, Dordrecht (1994)
28. Horcik, R., Navara, M.: Validation sets in fuzzy logics. *Kybernetika* 38(2), 319–326 (2002)
29. Leski, J.: ε -insensitive learning techniques for approximate reasoning system. *International Jour. of Computational Cognition* 1(1), 21–77 (2003)
30. Ruan, D., Kerre, E.: Fuzzy implication operators and generalized fuzzy methods of cases. *Fuzzy Sets and Systems* 54, 23–37 (1993)
31. Yager, R.: On the implication operator in fuzzy logic. *Information Sciences* 31, 141–164 (1983)
32. Yager, R.: On global requirements for implication operators in fuzzy modus ponens. *Fuzzy Sets and Systems* 106, 3–10 (1999)
33. Yager, R.: On some new classes of implication operators and their role in approximate reasoning. *Information Sciences* 167, 193–216 (2004)
34. Shi, Y., Ruan, D., Kerre, E.E.: On the characterizations of fuzzy implications satisfying $I(x, y) = I(x, I(x, y))$. *Information Sciences* 177(14), 2954–2970 (2007)
35. Klement, E., Navara, M.: A survey on different triangular norm-based fuzzy logics. *Fuzzy Sets and Systems* 101, 241–251 (1999)
36. Navara, M.: Characterization of measures based on strict triangular norms. *Mathematical Analysis and Applications* 236, 370–383 (1999)
37. Gehrke, M., Walker, C., Walker, E.: Some comments on interval valued fuzzy sets. *International Journal of Intelligent Systems* 11, 751–759 (1996)
38. Gehrke, M., Walker, C., Walker, E.: Algebraic aspects of fuzzy sets and fuzzy logics. In: *Proc. of the Work. on Current Trends and Development in Fuzzy Logic, Thessaloniki*, pp. 101–170 (1999)

Behavioural Differential Equations and Coinduction for Binary Trees

Alexandra Silva^{1,*} and Jan Rutten^{1,2}

¹ Centrum voor Wiskunde en Informatica (CWI)

² Vrije Universiteit Amsterdam (VUA)

{ams,janr}@cwi.nl

Abstract. We study the set T_A of infinite binary trees with nodes labelled in a semiring A from a coalgebraic perspective. We present coinductive definition and proof principles based on the fact that T_A carries a final coalgebra structure. By viewing trees as formal power series, we develop a calculus where definitions are presented as behavioural differential equations. We present a general format for these equations that guarantees the existence and uniqueness of solutions. Although technically not very difficult, the resulting framework has surprisingly nice applications, which is illustrated by various concrete examples.

1 Introduction

Infinite data structures are often used to model problems and computing solutions for them. Therefore, reasoning tools for such structures have become more and more relevant. Coalgebraic techniques turned out to be suited for proving and deriving properties of infinite systems.

In [6], a coinductive calculus of formal power series was developed. In close analogy to classical analysis, the definitions were presented as behavioural differential equations and properties were proved in a calculational (and very natural) way. This approach has shown to be quite effective for reasoning about streams [6,7] and it seems worthwhile to explore its effectiveness for other data structures as well.

In this paper, we shall take a coalgebraic perspective on a classical data structure – infinite binary trees, and develop a similar calculus, using the fact that binary trees are a particular case of formal power series.

The contributions of the present paper are: a coinductive calculus, based on the notion of derivative, to define and to reason about trees and functions on trees; a set of illustrative examples and properties that show the usefulness and expressiveness of such calculus; the formulation of a general format that guarantees the existence and uniqueness of solutions of behavioural differential equations.

Infinite trees arise in several forms in other areas. Formal tree series (functions from trees to an arbitrary semiring) have been studied in [3], closely related to

* Partially supported by the Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BD/27482/2006.

distributive Σ -algebras. The work presented in this paper is completely different since we are concerned with infinite binary trees and not with formal series over trees. In [5], infinite trees appear as generalisations of infinite words and an extensive study of tree automata and topological aspects of trees is made. We have not yet addressed the relation of our work with automata theory. Here we emphasize coinductive definition and proof principles for defining and reasoning about (functions on) trees.

At the end of the paper, in Section 6, the novelty of our approach is discussed further. Also several directions for further applications are mentioned there.

2 Trees and Coinduction

We introduce the set T_A of infinite node-labelled binary trees, show that T_A satisfies a coinduction proof principle and illustrate its usefulness.

The set T_A of infinite node-labelled binary trees, where to each node is assigned a value in A , is the final coalgebra for the functor $FX = X \times A \times X$ and can be formally defined by:

$$T_A = \{t \mid t : \{L, R\}^* \rightarrow A\}$$

The set T_A carries a final coalgebra structure consisting of the following function:

$$\begin{aligned} \langle l, i, r \rangle &: T_A \rightarrow T_A \times A \times T_A \\ t &\mapsto \langle \lambda w. t(Lw), t(\varepsilon), \lambda w. t(Rw) \rangle \end{aligned}$$

where l and r return the left and right subtrees, respectively, and i gives the label of the root node of the tree. Here, ε denotes the empty word and, for $b \in \{L, R\}$, bw denotes the word resulting from prefixing the word w with the letter b .

These definitions of the set T_A and the respective coalgebra map may not seem obvious. The follow reasoning justifies its correctness:

- It is well known from the literature [4] that the final system for the functor $G(X) = A \times X^B$ is (A^{B^*}, π) , where

$$\begin{aligned} \pi &: A^{B^*} \rightarrow A \times (A^{B^*})^B \\ \pi(\phi) &= \langle \phi(\varepsilon), \lambda b \ v. \ \phi(bv) \rangle \end{aligned}$$

- The functor F is isomorphic to $H(X) = A \times X^2$.
- Therefore, the set A^{2^*} is the final coalgebra for the functor F . Considering $2 = \{L, R\}$ we can derive the definition of $\langle l, i, r \rangle$ from the one presented above for π .

The fact that T_A is a final coalgebra means that for any arbitrary coalgebra $\langle lt, o, rt \rangle : U \rightarrow U \times A \times U$, there exists a unique $f : U \rightarrow T_A$, such that the following diagram commutes:

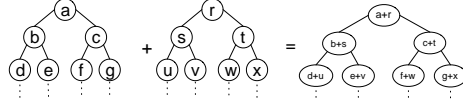
$$\begin{array}{ccc} U & \xrightarrow{\exists! f} & T_A \\ \langle lt, o, rt \rangle \downarrow & & \downarrow \langle l, i, r \rangle \\ U \times A \times U & \xrightarrow{f \times id_A \times f} & T_A \times A \times T_A \end{array}$$

The existence part of this statement gives us a *coinductive definition principle*. Every triplet of functions $lt : U \rightarrow U$, $o : U \rightarrow A$ and $rt : U \rightarrow U$ defines a function $h : U \rightarrow T_A$, such that:

$$i(h(x)) = o(x) \quad l(h(x)) = h(lt(x)) \quad r(h(x)) = h(rt(x))$$

We will see a more general formulation of this principle in section 3, where the right hand side of the above equations will be more general.

Taking $A = \mathbb{R}$ we present the definition of the elementwise sum as an example.



By the definition principle presented above, taking $o(\langle\sigma, \tau\rangle) = i(\sigma) + i(\tau)$, $lt(\langle\sigma, \tau\rangle) = \langle l(\sigma), l(\tau) \rangle$ and $rt(\langle\sigma, \tau\rangle) = \langle r(\sigma), r(\tau) \rangle$ there is a unique function $+: T_{\mathbb{R}} \times T_{\mathbb{R}} \rightarrow T_{\mathbb{R}}$ satisfying:

$$i(\sigma + \tau) = i(\sigma) + i(\tau) \quad l(\sigma + \tau) = l(\sigma) + l(\tau) \quad r(\sigma + \tau) = r(\sigma) + r(\tau)$$

Note that in the first equation above we are using $+$ to represent both the sum of trees and the sum of real numbers.

Now that we have explained the formal definition for the set T_A and how one can uniquely define functions into T_A , another important question is still to be answered: how do we prove equality on T_A ? In order to prove that two trees σ and τ are equal it is necessary and sufficient to prove

$$\forall w \in \{L, R\}^* \quad \sigma(w) = \tau(w) \quad (1)$$

The use of induction on w (prove that $\sigma(\varepsilon) = \tau(\varepsilon)$ and that whenever $\sigma(w) = \tau(w)$ holds then $\sigma(aw) = \tau(aw)$ also holds, for $a \in \{L, R\}$) clearly is a correct method to establish the validity of (1). However, we will often encounter examples where there is not a general formula for $\sigma(w)$ and $\tau(w)$. Instead, we take a coalgebraic perspective on T_A and use the *coinduction proof principle* in order to establish equalities. This proof principle is based on the notion of bisimulation. A bisimulation on T_A is a relation $S \subseteq T_A \times T_A$ such that, for all σ and τ in T_A ,

$$(\sigma, \tau) \in S \Rightarrow \sigma(\varepsilon) = \tau(\varepsilon) \wedge (l(\sigma), l(\tau)) \in S \wedge (r(\sigma), r(\tau)) \in S$$

We will write $\sigma \sim \tau$ whenever there exists a bisimulation that contains (σ, τ) . The relation \sim , called the bisimilarity relation, is the union of all bisimulations (one can easily check that the union of bisimulation is itself a bisimulation).

The following theorem expresses the proof principle mentioned above.

Theorem 1 (Coinduction). *For all trees σ and τ in T_A , if $\sigma \sim \tau$ then $\sigma = \tau$.*

Proof. Consider two trees σ and τ in T_A and let $S \subseteq T_A \times T_A$ be a bisimulation relation which contains the pair (σ, τ) . The equality $\sigma(w) = \tau(w)$ now follows

by induction on the length of w . We have that $\sigma(\varepsilon) = \tau(\varepsilon)$, because S is a bisimulation. If $w = Lw'$, then

$$\begin{aligned}\sigma(Lw') &= l(\sigma)(w') && \text{(Definition of } l) \\ &= l(\tau)(w') && (S \text{ is a bisimulation and induction hypothesis}) \\ &= \tau(Lw') && \text{(Definition of } l)\end{aligned}$$

Similarly, if $w = Rw'$, then $\sigma(Rw') = r(\sigma)(w') = r(\tau)(w') = \tau(Rw')$. Therefore, for all $w \in \{L, R\}^*$, $\sigma(w) = \tau(w)$. This proves $\sigma = \tau$.

Thus, in order to prove that two trees are equal, it is sufficient to show that they are bisimilar. We shall see several examples of proofs by *coinduction* below.

As a first simple example, let us prove that the pointwise sum for trees of real numbers defined before is commutative. Let $S = \{\langle \sigma + \tau, \tau + \sigma \rangle \mid \sigma, \tau \in T_{\mathbb{R}}\}$. Since $i(\sigma + \tau) = i(\sigma) + i(\tau) = i(\tau + \sigma)$ and

$$\begin{aligned}l(\sigma + \tau) &= l(\sigma) + l(\tau) & S & l(\tau) + l(\sigma) = l(\tau + \sigma) \\ r(\sigma + \tau) &= r(\sigma) + r(\tau) & S & r(\tau) + r(\sigma) = r(\tau + \sigma)\end{aligned}$$

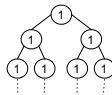
for any σ and τ , S is a bisimulation relation on $T_{\mathbb{R}}$. The commutativity property follows by coinduction.

3 Behavioural Differential Equations

In this section, we shall view trees as formal power series. Following [6], coinductive definitions of operators into T_A and constant trees then take the shape of so-called *behavioural differential equations*. We shall prove a theorem guaranteeing the existence of a unique solution for a large family of systems of behavioural differential equations.

Formal power series are functions $\sigma : \mathcal{X}^* \rightarrow k$ from the set of words over an alphabet \mathcal{X} to a semiring k . If A is a semiring, T_A , as defined in section 2, is the set of all formal power series over the alphabet $\{L, R\}$ with coefficients in A . In accordance with the general notion of *derivative* of formal power series [6] we shall write σ_L for $l(\sigma)$ and σ_R for $r(\sigma)$. We will often refer to σ_L , σ_R and $\sigma(\varepsilon)$ as the left derivative, right derivative and initial value of σ .

Following [6], we will develop a coinductive calculus of infinite binary trees. From now on coinductive definitions will have the shape of *behavioural differential equations*. Let us illustrate this approach by a simple example – the coinductive definition of a tree, called *one*, decorated with 1's in every node.

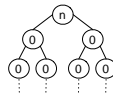


A formal definition of this tree consists the following behavioural differential equations:

differential equations	initial value
$one_L = one$	$one(\varepsilon) = 1$
$one_R = one$	

The fact that there exists a unique tree that satisfies the above equations will follow from the theorem below, which presents a general format for behavioural differential equations guaranteeing the existence and uniqueness of solutions.

Behavioural differential equations will be used not just to define single constant trees but also functions on trees. We shall see examples below. Before we present the main result of this section, we need one more definition. We want to be able to view any element $n \in A$ as a tree (which we will denote by $[n]$):



The tree $[n]$ is formally defined as

$$\begin{aligned} [n](\varepsilon) &= n \\ [n](w) &= 0 \quad w \neq \varepsilon \end{aligned}$$

Next we present a syntax describing the format of behavioural differential equations that we will consider. Let Σ be a set of function symbols, each with an arity $r(f) \geq 0$ for $f \in \Sigma$. (As usual we call f a constant if $r(f) = 0$.) Let $X = \{x_1, x_2, \dots\}$ be a set of (syntactic) variables, and let $X_L = \{x_L \mid x \in X\}$, $X_R = \{x_R \mid x \in X\}$, $[X(\varepsilon)] = \{[x(\varepsilon)] \mid x \in X\}$ and $X(\varepsilon) = \{x(\varepsilon) \mid x \in X\}$ be sets of notational variants of them. The variables $x \in X$ will play the role of place holders for trees $\tau \in T_A$. Variables x_L , x_R , and $[x(\varepsilon)]$ will then act as place holders for the corresponding trees τ_L , τ_R and $[\tau(\varepsilon)]$ in T_A , while $x(\varepsilon)$ (without the square brackets) will correspond to τ 's initial value $\tau(\varepsilon) \in A$. We call a behavioural differential equation for a function symbol $f \in \Sigma$ with arity $r = r(f)$ *well-formed* if it is of the form

differential equations	initial value
$f(x_1, \dots, x_r)_L = t_1$	$(f(x_1, \dots, x_r))(\varepsilon) = c(x_1(\varepsilon), \dots, x_r(\varepsilon))$
$f(x_1, \dots, x_r)_R = t_2$	

where $c : A^r \rightarrow A$ is a given function, and where t_1 and t_2 are terms built out of function symbols in Σ and variables in $\{x_1, \dots, x_r\}$ and their corresponding notational variants in X_L , X_R and $[X(\varepsilon)]$. A well-formed system of equations for Σ will then consist of one well-formed equation for each $f \in \Sigma$. A *solution* of such a system of equations is a set of tree functions

$$\tilde{\Sigma} = \{\tilde{f} : (T_A)^r \rightarrow T_A \mid f \in \Sigma\}$$

satisfying, for all $f \in \Sigma$ with arity r and for all $\tau_1, \dots, \tau_r \in T_A$,

$$\left(\tilde{f}(\tau_1, \dots, \tau_r) \right) (\varepsilon) = c(\tau_1(\varepsilon), \dots, \tau_r(\varepsilon))$$

and

$$\left(\tilde{f}(\tau_1, \dots, \tau_r) \right)_L = \tilde{t}_1 \quad \text{and} \quad \left(\tilde{f}(\tau_1, \dots, \tau_r) \right)_R = \tilde{t}_2$$

where the tree $\tilde{t}_1 \in T_A$ (and similarly for \tilde{t}_2) is obtained from the term t_1 by replacing (all occurrences of) x_i by τ_i , $(x_i)_L$ by $(\tau_i)_L$, $(x_i)_R$ by $(\tau_i)_R$, and $[x_i(\varepsilon)]$ by $[\tau_i(\varepsilon)]$, for all $i = 1, \dots, r$, and all function symbols $g \in \Sigma$ by their corresponding function \tilde{g} .

Theorem 2. *Let Σ be a set of function symbols. Every well-formed system of behavioural differential equations for Σ has precisely one solution of tree functions $\tilde{\Sigma}$.*

Proof. Please see appendix A.

Let us illustrate the generality of this theorem by mentioning a few examples of systems of differential equations that satisfy the format above. As a first example, take $\Sigma = \{\text{one}\}$ consisting of a single constant symbol (with arity 0) and $X = \emptyset$. We observe that the differential equations for *one* mentioned at the beginning of this section satisfy the format of the theorem. For a second example, let $\Sigma = \{+, \times\}$ with arities $r(+) = r(\times) = 2$ and let $X = \{\sigma, \tau\}$. Consider the following equations:

differential equations	initial value
$(\sigma + \tau)_L = \sigma_L + \tau_L$ $(\sigma + \tau)_R = \sigma_R + \tau_R$	$(\sigma + \tau)(\varepsilon) = \sigma(\varepsilon) + \tau(\varepsilon)$

differential equations	initial value
$(\sigma \times \tau)_L = (\sigma_L \times \tau) + ([\sigma(\varepsilon)] \times \tau_L)$ $(\sigma \times \tau)_R = (\sigma_R \times \tau) + ([\sigma(\varepsilon)] \times \tau_R)$	$(\sigma \times \tau)(\varepsilon) = \sigma(\varepsilon) \times \tau(\varepsilon)$

These equations define the operations of *sum* and *convolution product* of trees, to be further discussed in Section 4. Note that the right-hand side of the equation for $(\sigma \times \tau)_L$ (and similarly for $(\sigma \times \tau)_R$) is a good illustration of the general format: it is built from the functions $+$ and \times , applied to (a subset of) the variables on the left (τ), their derivatives (σ_L and τ_L), and their initial values viewed as trees ($[\sigma(\varepsilon)]$).

Clearly there are many interesting instances of well-formed differential equations. Note, however, that the format *does* impose certain restrictions. The main point is that in the right-hand sides of the equations, only single L and R derivatives are allowed. The following is an example of a system of equations that is *not* well-formed and that does *not* have a unique solution.

Let $\Sigma = \{f\}$, with arity $r(f) = 1$, and let $X = \{\sigma\}$. The equations for f are

differential equations	initial value
$f(\sigma)_L = f(f(\sigma_{LL}))$ $f(\sigma)_R = [0]$	$f(\sigma)(\varepsilon) = \sigma(\varepsilon)$

Both $g(\sigma) = [\sigma(\varepsilon)] + (L \times [\sigma_{LL}(\varepsilon)])$ and $h(\sigma) = [\sigma(\varepsilon)] + (L \times [\sigma_{LL}(\varepsilon)] + L^2 \times (1 - L)^{-1})$ are solutions.

All the examples of systems of behavioural differential equations that will appear in the rest of this document fit into the format of Theorem 2. Therefore, we will omit proofs of the existence and uniqueness of solutions for those systems.

In the next section we will define operators on trees, based on some general operators on formal power series [6].

4 Tree Calculus

In this section, we present operators on trees, namely sum, convolution product and inverse, and state some elementary properties, which we will prove using coinduction.

The sum of two trees is defined as the unique operator satisfying:

differential equations	initial value
$(\sigma + \tau)_L = \sigma_L + \tau_L$ $(\sigma + \tau)_R = \sigma_R + \tau_R$	$(\sigma + \tau)(\varepsilon) = \sigma(\varepsilon) + \tau(\varepsilon)$

Note that this is a generalisation of the sum on trees of real numbers defined in section 2 and that again we are overloading the use of $+$ to represent both sum on trees and sum on the elements of the semiring.

Sum satisfies some desired properties, easily proved by coinduction, such as commutativity or associativity:

Theorem 3. *For all σ, τ and ρ in T_A , $\sigma + 0 = \sigma$, $\sigma + \tau = \tau + \sigma$ and $\sigma + (\tau + \rho) = (\sigma + \tau) + \rho$.*

Here, we are using 0 as a shorthand for $[0]$. We shall use this convention (for all $n \in A$) throughout this document. We define the convolution product of two trees as the unique operation satisfying:

differential equations	initial value
$(\sigma \times \tau)_L = (\sigma_L \times \tau) + (\sigma(\varepsilon) \times \tau_L)$ $(\sigma \times \tau)_R = (\sigma_R \times \tau) + (\sigma(\varepsilon) \times \tau_R)$	$(\sigma \times \tau)(\varepsilon) = \sigma(\varepsilon) \times \tau(\varepsilon)$

Note that in the above definition we use \times for representing both multiplication on trees and on the elements of the semiring. Following the convention mentioned above $\sigma(\varepsilon) \times \tau_L$ and $\sigma(\varepsilon) \times \tau_R$ are shorthands for $[\sigma(\varepsilon)] \times \tau_L$ and $[\sigma(\varepsilon)] \times \tau_R$. We shall also use the standard convention of writing $\sigma\tau$ for $\sigma \times \tau$.

The general formula to compute the value of σ according to a path given by the word $w \in \{L, R\}^*$ is:

$$(\sigma \times \tau)(w) = \sum_{w=u \cdot v} \sigma(u) \times \tau(v)$$

where \cdot denotes word concatenation.

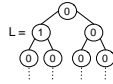
To give the reader some intuition about this operation we will give a concrete example. Take A to be the Boolean semiring $\mathbb{B} = \{0, 1\}$, with operations $+$ and \times . Then, T_A corresponds to the languages over a two letter alphabet, and in this case the tree product operator coincides with language concatenation.

The following theorem states some familiar properties of this operator.

Theorem 4. *For all σ, τ, ρ in T_A and a, b in A , $\sigma \times 1 = 1 \times \sigma = \sigma$, $\sigma \times 0 = 0 \times \sigma = 0$, $\sigma \times (\tau + \rho) = (\sigma \times \tau) + (\sigma \times \rho)$, $\sigma \times (\tau \times \rho) = (\sigma \times \tau) \times \rho$ and $[a] \times [b] = [a \times b]$.*

Proof. An exercise in coinduction. In [7], these properties are proved for streams.

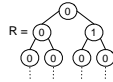
Note that the convolution product is not commutative. Before we present the inverse operation, let us introduce two (very useful) constants, which we shall call left constant L and right constant R . They will have an important role in the tree calculus that we are about to develop and will turn out to have interesting properties when interacting with the product operation. The left constant L is a tree with 0's in every node except in the root of the left branch where it has a 1:



It is formally defined as

$$\begin{aligned} L(w) &= 1 \text{ if } w = L \\ L(w) &= 0 \text{ otherwise} \end{aligned}$$

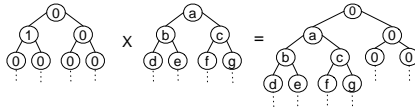
Similarly, the right constant R is only different from 0 in the root of its right branch:



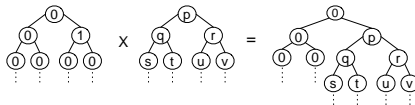
and is defined as

$$\begin{aligned} R(w) &= 1 \text{ if } w = R \\ R(w) &= 0 \text{ otherwise} \end{aligned}$$

These constants have interesting properties when multiplied by an arbitrary tree. $L \times \sigma$ produces a tree whose root and right subtrees are null and the left branch is σ :



Dually, $R \times \sigma$ produces a tree whose root and left subtrees are null and the right branch is σ :

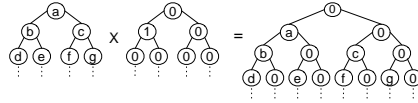


As before, if we see L and σ as languages and the product as concatenation, we can gain some intuition on the meaning of this operation. $L \times \sigma$ will prefix every word of σ with the letter L , meaning that no word starting by R will be an element of $L \times \sigma$, and thus, $L \times \sigma$ has a null right branch. Similar for $R \times \sigma$.

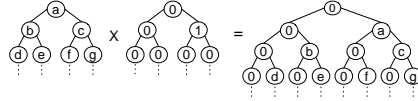
As we pointed out before, the product operation is not commutative. For example, $\sigma \times L \neq L \times \sigma$ and $\sigma \times R \neq R \times \sigma$. In fact, multiplying a tree σ on the right with L or R is interesting in itself. For instance, $\sigma \times L$ satisfies

$$(\sigma \times L)(w) = \begin{cases} \sigma(u) & w = uL \\ 0 & \text{otherwise} \end{cases}$$

which corresponds to the following transformation:



Similarly, $\sigma \times R$ produces the following tree:



Again, if you interpret these operations in the language setting, what is being constructed is the language that has all words of the form uL and uR , respectively, such that $\sigma(u) \neq 0$.

We define the inverse of a tree as the unique operator satisfying:

differential equations	initial value
$(\sigma^{-1})_L = \sigma(\varepsilon)^{-1} \times (-\sigma_L \times (\sigma^{-1}))$ $(\sigma^{-1})_R = \sigma(\varepsilon)^{-1} \times (-\sigma_R \times (\sigma^{-1}))$	$\sigma^{-1}(\varepsilon) = \sigma(\varepsilon)^{-1}$

We are using $-\sigma_L$ and $-\sigma_R$ as shorthands for $[-1] \times \sigma_L$ and $[-1] \times \sigma_R$, respectively. In this definition, we need to require A to be a ring, in order to have additive inverses. Moreover, the tree σ is supposed to have also a multiplicative inverse for its initial value.

The inverse of a tree has the usual properties:

Theorem 5. For all σ and τ in T_A :

$$\sigma^{-1} \text{ is the unique tree s.t. } \sigma \times \sigma^{-1} = 1 \quad (2)$$

$$(\sigma \times \tau)^{-1} = \tau^{-1} \times \sigma^{-1} \quad (3)$$

Proof. For the existence part of (2), note that

1. $(\sigma \times \sigma^{-1})(\varepsilon) = \sigma(\varepsilon) \times \sigma(\varepsilon)^{-1} = 1$
2. $(\sigma \times \sigma^{-1})_L = (\sigma_L \times \sigma^{-1}) + (\sigma(\varepsilon) \times (\sigma(\varepsilon)^{-1} \times (-\sigma_L \times \sigma^{-1}))) = 0$
3. $(\sigma \times \sigma^{-1})_R = (\sigma_R \times \sigma^{-1}) + (\sigma(\varepsilon) \times (\sigma(\varepsilon)^{-1} \times (-\sigma_R \times \sigma^{-1}))) = 0$

So, by uniqueness (using the behavioural differential equations that define 1) we have proved that $\sigma \times \sigma^{-1} = 1$. Now, for the uniqueness part of (2), suppose that there is a tree τ such that $\sigma \times \tau = 1$. We shall prove that $\tau = \sigma^{-1}$. Note that from the equality $\sigma \times \tau = 1$ we derive that

1. $\tau(\varepsilon) = \sigma(\varepsilon)^{-1}$
2. $\tau_L = \sigma(\varepsilon) \times (-\sigma_L \times \tau)$
3. $\tau_R = \sigma(\varepsilon) \times (-\sigma_R \times \tau)$

Thus, by uniqueness of solutions for systems of behavioural differential equations, $\tau = \sigma^{-1}$.

For (3), note that $(\sigma \times \tau) \times \tau^{-1} \times \sigma^{-1} = \sigma \times (\tau \times \tau^{-1}) \times \sigma^{-1} = 1$. Therefore, using the uniqueness property of (2), $(\sigma \times \tau)^{-1} = \tau^{-1} \times \sigma^{-1}$.

5 Applications of Tree Calculus

We will illustrate the usefulness of our calculus by looking at a series of interesting examples. In order to compute closed formulae for trees we will be using the following identity:

$$\forall \sigma \in T_A \quad \sigma = \sigma(\varepsilon) + (L \times \sigma_L) + (R \times \sigma_R) \quad (4)$$

which can be easily proved by coinduction. We will now show how to use this identity to construct the closed formula for a tree.

Recall our first system of behavioural differential equations:

differential equations	initial value
$one_L = one$	$one(\varepsilon) = 1$
$one_R = one$	

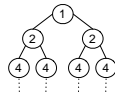
There we saw that the unique solution for this system was the tree with 1's in every node. Alternatively, we can compute the solution using (4) as follows.

$$\begin{aligned}
 one &= one(\varepsilon) + (L \times one_L) + (R \times one_R) \\
 \Leftrightarrow one &= 1 + (L \times one) + (R \times one) \\
 \Leftrightarrow (1 - L - R)one &= 1 \\
 \Leftrightarrow one &= (1 - L - R)^{-1}
 \end{aligned}$$

Therefore, one can be represented by the (very compact) closed formula¹ $(1 - L - R)^{-1}$.

Let us see a few more examples. From now on we will work with $A = \mathbb{R}$, where we have the extra property: $[n] \times \sigma = \sigma \times [n]$, for all $n \in \mathbb{R}$ and $\sigma \in T_{\mathbb{R}}$.

The tree where every node at level k is labelled with the value 2^k , called *pow*,



is defined by the following system:

¹ Note the similarity of this closed formula with the one obtained for the stream $(1, 1, \dots)$ in [7]: $(1 - X)^{-1}$.

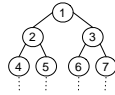
differential equations	initial value
$pow_L = 2 \times pow$ $pow_R = 2 \times pow$	$pow(\varepsilon) = 1$

We proceed as before, applying (4):

$$\begin{aligned}
 pow &= pow(\varepsilon) + (L \times pow_L) + (R \times pow_R) \\
 \Leftrightarrow pow &= 1 + (2L \times pow) + (2R \times pow) \\
 \Leftrightarrow (1 - 2L - 2R)pow &= 1 \\
 \Leftrightarrow pow &= (1 - 2L - 2R)^{-1}
 \end{aligned}$$

which gives us a nice closed formula for pow^2 .

The tree with the natural numbers



is represented by the following system of differential equations:

differential equations	initial value
$nat_L = nat + pow$ $nat_R = nat + (2 \times pow)$	$nat(\varepsilon) = 1$

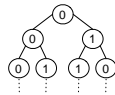
Applying identity (4):

$$\begin{aligned}
 nat &= nat(\varepsilon) + (L \times nat_L) + (R \times nat_R) \\
 \Leftrightarrow nat &= 1 + (L \times (nat + pow)) + (R \times (nat + 2pow)) \\
 \Leftrightarrow (1 - L - R)nat &= 1 + L(1 - 2L - 2R)^{-1} + 2R(1 - 2L - 2R)^{-1} \\
 \Leftrightarrow (1 - L - R)nat &= (1 - L) \times (1 - 2L - 2R)^{-1} \\
 \Leftrightarrow nat &= (1 - L - R)^{-1} \times (1 - L) \times (1 - 2L - 2R)^{-1}
 \end{aligned}$$

The Thue-Morse sequence [1] can be obtained by taking the parities of the counts of 1's in the binary representation of non-negative integers. Alternatively, it can be defined by the repeated application of the substitution map $\{0 \rightarrow 01; 1 \rightarrow 10\}$:

$$0 \rightarrow 01 \rightarrow 0110 \rightarrow 01101001 \rightarrow \dots$$

We can encode this substitution map in a binary tree, called *thue*, which at each level k will have the first 2^k digits of the Thue-Morse sequence:



In this example, we take for A the Boolean ring $2 = \{0, 1\}$ (where $1 + 1 = 0$). The following system of differential equations defines *thue*:

² Again, there is a strong similarity with streams: the closed formula for the stream $(1, 2, 4, 8, \dots)$ is $(1 - 2X)^{-1}$.

differential equations	initial value
$thue_L = thue$ $thue_R = thue + one$	$thue(\varepsilon) = 0$

Note that $thue + one$ equals the (elementwise) complement of $thue$. Applying (4) to $thue$, we calculate:

$$\begin{aligned}
 thue &= (L \times thue) + (R \times (thue + one)) \\
 \Leftrightarrow (1 - L - R) \times thue &= R \times one \\
 \Leftrightarrow thue &= (1 - L - R)^{-1} \times R \times one
 \end{aligned}$$

which then leads to the following pretty formula for $thue$:

$$thue = one \times R \times one$$

Let us present another example – a substitution operation, which given two trees σ and τ , replaces the left subtree of σ by τ .

$$subst \left(\begin{array}{c} \sigma(\varepsilon) \\ \swarrow \quad \searrow \\ \sigma_L \quad \sigma_R \end{array}, \tau \right) = \begin{array}{c} \sigma(\varepsilon) \\ \swarrow \quad \searrow \\ \tau \quad \sigma_R \end{array}$$

It is easy to see that the equations that define this operation are:

differential equations	initial value
$subst(\sigma, \tau)_L = \tau$ $subst(\sigma, \tau)_R = \sigma_R$	$subst(\sigma, \tau)(\varepsilon) = \sigma(\varepsilon)$

Then, we apply (4) and we reason:

$$\begin{aligned}
 subst(\sigma, \tau) &= \sigma(\varepsilon) + (L \times \tau) + (R \times \sigma_R) \\
 \Leftrightarrow subst(\sigma, \tau) &= \sigma - (L \times \sigma_L) + (L \times \tau) \\
 \Leftrightarrow subst(\sigma, \tau) &= \sigma - L(\sigma_L - \tau)
 \end{aligned}$$

Note that in the second step, we applied identity (4) to σ . Moreover, remark that the final closed formula for $subst(\sigma, \tau)$ gives us the algorithm to compute the substitution:

$$\begin{array}{c}
 subst \left(\begin{array}{c} \sigma(\varepsilon) \\ \swarrow \quad \searrow \\ \sigma_L \quad \sigma_R \end{array}, \tau \right) = \begin{array}{c} \sigma(\varepsilon) \\ \swarrow \quad \searrow \\ \tau \quad \sigma_R \end{array} \\
 \downarrow \begin{array}{c} \begin{array}{c} 0 \\ \swarrow \quad \searrow \\ \sigma_L \quad 0 \end{array} \end{array} \quad \nearrow \\
 \begin{array}{c} \begin{array}{c} \sigma(\varepsilon) \\ \swarrow \quad \searrow \\ 0 \quad \sigma_R \end{array} + \begin{array}{c} 0 \\ \swarrow \quad \searrow \\ \tau \quad 0 \end{array} \end{array}
 \end{array}$$

We can now wonder how to define a more general substitution operation that has an arbitrary path $P \in \{L, R\}^+$ as an extra argument and replaces the subtree of σ given by this path by τ . It seems obvious to define it as

$$\text{subst}(\sigma, \tau, P) = \sigma - P(\sigma_P - \tau)$$

where, in the right hand side, $P = a_1 a_2 \dots a_n$ is interpreted as $a_1 \times a_2 \times \dots \times a_n$ and the derivative σ_P is defined as

$$\sigma_P = \begin{cases} \sigma_\delta & P = \delta \\ (\sigma_\delta)_{P'} & P = \delta.P' \end{cases}$$

with δ being either L or R .

Let us check that our intuition is correct. First, we present the definition for this operation:

differential equations	initial value
$\text{subst}(\sigma, \tau, P)_\delta = \begin{cases} \tau & P = \delta \\ \text{subst}(\sigma_\delta, \tau, P') & P = \delta.P' \\ \sigma_\delta & P = \delta'.P' \end{cases}$	$\text{subst}(\sigma, \tau, P)(\varepsilon) = \sigma(\varepsilon)$

where $\delta' \neq \delta$. Now, observe that

$$R = \{ \langle \text{subst}(\sigma, \tau, P), \sigma - P(\sigma_P - \tau) \rangle \mid \sigma, \tau \in T_{\mathbb{R}}, P \in \{L, R\}^+ \} \cup \{ \langle \sigma, \sigma \rangle \mid \sigma \in T_{\mathbb{R}} \}$$

is a bisimulation relation because:

1. $(\sigma - P(\sigma_P - \tau))(\varepsilon) = \sigma(\varepsilon) = \text{subst}(\sigma, \tau, P)(\varepsilon)$
2. For $\delta \in \{L, R\}$,

$$\begin{aligned} (\sigma - P(\sigma_P - \tau))_\delta &= \sigma_\delta - P_\delta(\sigma_P - \tau) \\ &= \begin{cases} \tau & P = \delta \\ \sigma_\delta - P'((\sigma_\delta)_{P'} - \tau) & P = \delta.P' \\ \sigma_\delta & P = \delta'.P' \end{cases} \\ &R \begin{cases} \tau & P = \delta \\ \text{subst}(\sigma_\delta, \tau, P') & P = \delta.P' \\ \sigma_\delta & P = \delta'.P' \end{cases} \\ &= \text{subst}(\sigma, \tau, P)_\delta \end{aligned}$$

Therefore, by Theorem 1, $\text{subst}(\sigma, \tau, P) = \sigma - P(\sigma_P - \tau)$.

Using this formula we can now prove properties about this operation. For instance, one would expect that $\text{subst}(\sigma, \sigma_P, P) = \sigma$ and also $\text{subst}(\text{subst}(\sigma, \tau, P), \sigma_P, P) = \sigma$.

The first equality follows easily: $\text{subst}(\sigma, \sigma_P, P) = \sigma - P(\sigma_P - \sigma_P) = \sigma$. For the second one we have:

$$\begin{aligned} &\text{subst}(\text{subst}(\sigma, \tau, P), \sigma_P, P) \\ &= \text{subst}(\sigma - P(\sigma_P - \tau), \sigma_P, P) && \text{(Definition of subst)} \\ &= \sigma - P(\sigma_P - \tau) - P((\sigma - P(\sigma_P - \tau))_P - \sigma_P) && \text{(Definition of subst)} \\ &= \sigma - P(\sigma_P - \tau) - P(\tau - \sigma_P) && ((\sigma - P(\sigma_P - \tau))_P = \sigma_P - \sigma_P + \tau = \tau) \\ &= \sigma \end{aligned}$$

Remark that this operation is a standard example in introductory courses on algorithms and data structures. It is often presented either as a recursive expression (very much in the style of our differential equations) or as a contrived

iterative procedure. This example shows that our compact formulae constitute a clear way of presenting algorithms and that they can be used to eliminate recursion. Moreover, the differential equations are directly implementable algorithms (in functional programming) and our calculus provides a systematic way of reasoning about such programs.

6 Discussion

We have modelled binary trees as formal power series and, using the fact that the latter constitute a final coalgebra, this has enabled us to apply some coalgebraic reasoning. Technically, none of this is very difficult. Rather, it is an application of well known coalgebraic insights. As is the case with many of such applications, it has the flavour of an exercise. At the same time, the result contains several new elements that have surprised us. Although technically Theorem 2 is an easy extension of a similar such theorem for streams, the resulting format for differential equations for trees is surprisingly general and useful. It has allowed us to define various non-trivial trees by means of simple differential equations, and to compute rather pleasant closed formulae for them. We have also illustrated that based on this, coinduction is a convenient proof method for trees. As an application, all of this is new, to the best of our knowledge. (Formal tree series, which have been studied extensively, may seem to be closely related but are not: here we are dealing with differential equations that characterise *single* trees.) In addition to the illustrations of the present differential calculus for trees, we see various directions for further applications: (i) The connection with (various types of) automata and the final coalgebra T_A of binary trees needs further study. For instance, every Moore automaton with input in $2 = \{L, R\}$ and output in A has a minimal representation in T_A . Details of these automata (and on weighted variants of them) are not complicated but have not been worked out here because of lack of space. (ii) The closed formula that we have obtained for the (binary tree representing the) Thue-Morse sequence suggests a possible use of coinduction and differential equations in the area of automatic sequences [2]. Typically, automatic sequences are represented by automata. The present calculus seems an interesting alternative, in which properties such as algebraicity of sequences can be derived from the tree differential equations that define them. (iii) Finally, the closed formulae that we obtain for tree substitution suggest many further applications of our tree calculus to (functional) programs on trees, including the analysis of their complexity.

References

1. Allouche, J.-P., Shallit, J.: The ubiquitous Prouhet-Thue-Morse sequence. In: Ding, C., Helleseht, T., H., N. (eds.) Sequences and their applications, Proceedings of SETA'98, pp. 1–16. Springer, Heidelberg (1999)
2. Allouche, J.-P., Shallit, J.: Automatic sequences: theory, applications, generalizations. Cambridge University Press, Cambridge (2003)

3. Ésik, Z., Kuich, W.: Formal tree series. *Journal of Automata, Languages and Combinatorics* 8(2), 219–285 (2003)
4. Manes, E.G., Arbib, M.A.: *Algebraic approaches to program semantics*. Springer, Heidelberg (1986)
5. Perrin, D., Pin, J.-E.: *Infinite Words*, Pure and Applied Mathematics. Pure and Applied Mathematics, vol. 141. Elsevier, Amsterdam (2004)
6. Rutten, J.J.M.M.: Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comput. Sci.* 308(1-3), 1–53 (2003)
7. Rutten, J.J.M.M.: A coinductive calculus of streams. *Mathematical Structures in Computer Science* 15(1), 93–147 (2005)

A Proof of Theorem 2

Proof (Proof of theorem 2). Consider a well-formed system of differential equations for Σ , as defined above. We define a set \mathcal{T} of terms t by the following syntax:

$$t ::= \underline{\tau} \quad (\tau \in T_A) \mid f(t_1, \dots, t_{r(f)}) \quad (f \in \Sigma)$$

where for every tree $\tau \in T_A$ the set \mathcal{T} contains a corresponding term, denoted by $\underline{\tau}$, and where new terms are constructed by (syntactic) composition of function symbols from Σ with the appropriate number of argument terms. Next we turn \mathcal{T} into an F -coalgebra by defining a function $\langle l, o, r \rangle : \mathcal{T} \rightarrow (\mathcal{T} \times A \times \mathcal{T})$ by induction on the structure of terms, as follows. First we define $o : \mathcal{T} \rightarrow A$ by

$$\begin{aligned} o(\underline{\tau}) &= \tau(\varepsilon) \\ o(f(t_1, \dots, t_{r(f)})) &= c(o(t_1), \dots, o(t_{r(f)})) \end{aligned}$$

(where c is the function used in the equations for f). Next we define $l : \mathcal{T} \rightarrow \mathcal{T}$ and $r : \mathcal{T} \rightarrow \mathcal{T}$ by $l(\underline{\tau}) = \underline{\tau_L}$ and $r(\underline{\tau}) = \underline{\tau_R}$, and by

$$l(f(t_1, \dots, t_r)) = \overline{t_1} \quad \text{and} \quad r(f(t_1, \dots, t_r)) = \overline{t_2}$$

Here the terms $\overline{t_1}$ and $\overline{t_2}$ are obtained from the terms t_1 and t_2 used in the equations for f , by replacing (every occurrence of) x_i by t_i , $(x_i)_L$ by $l(t_i)$, $(x_i)_R$ by $r(t_i)$, and $[x_i(\varepsilon)]$ by $[o(t)]$, for $i = 1, \dots, r$. Because T_A is a final F -coalgebra, there exists a unique homomorphism $h : \mathcal{T} \rightarrow T_A$. We can use it to define tree functions $\tilde{f} : (T_A)^r \rightarrow T_A$, for every $f \in \Sigma$, by putting, for all $\tau_1, \dots, \tau_r \in T_A$,

$$\tilde{f}(\tau_1, \dots, \tau_r) = h(f(\underline{\tau_1}, \dots, \underline{\tau_r}))$$

This gives us a set $\tilde{\Sigma}$ of tree functions. One can prove that it is a solution of our system of differential equations by coinduction, using the facts that $h(\underline{\tau}) = \tau$, for all $\tau \in T_A$, and

$$h(f(t_1, \dots, t_r)) = \tilde{f}(h(t_1), \dots, h(t_r))$$

for all $f \in \Sigma$ and $t_i \in \mathcal{T}$. This solution is unique because, by finality of T_A , the homomorphism h is.

A Sketch of a Dynamic Epistemic Semiring

Kim Solin*

Åbo Akademi, Åbo, Finland
kim.solin@abo.fi

Abstract. This paper proposes a semiring formulation for reasoning about an agent’s changing beliefs: a *dynamic epistemic semiring* (DES). A DES is a modal semiring extended with epistemic-action operators. The paper concentrates on the revision operator by proposing an axiomatisation, developing a basic calculus and deriving the classical AGM revision axioms in the algebra.

1 Introduction

Formal reasoning about epistemic and doxastic notions in the 20th century can be traced back to Georg Henrik von Wright’s *An Essay in Modal Logic* [26] and Hintikka’s *Knowledge and Belief* [13]. These books were written as attempts to philosophically clarify our notions of knowledge and belief using modal logic. In the works of von Wright and Hintikka, the ones believing or knowing – the agents – cannot change their beliefs, *i.e.* only static aspects of an agent’s beliefs can be reasoned about. Alchourrón, Gärdenfors and Makinson presented a semiformal framework for reasoning about an agent’s changing beliefs [1] – this constituted the prelude for the vast amount of research on belief revision done in the last two decades. Segerberg was one of the first to make these streams flow together. In an array of papers (see for example [20, 21, 22]) he proposes and develops a fully formal framework for reasoning about an agent’s changing beliefs: dynamic doxastic logic. Dynamic doxastic logic is, as the name discerns, a blend of dynamic and doxastic logic. In more recent times, formal reasoning about knowledge has been put to use in computer science, the standard references seem to be [12, 14, 25]. (The epithet epistemic is usually used in computer science, whereas in philosophical logic one tends to distinguish between epistemic and doxastic. We will henceforth use ‘epistemic’, except when referring to Segerberg’s logic.)

Since Kozen published his axiomatisation of Kleene algebra (an idempotent semiring with the Kleene star as the least fixpoint with respect to the canonical order) [15] and also showed how to use an elaboration of it for program reasoning [16], there has been significant development and application of something that could be called semiring structures (some form of semiring equipped with additional operators). The spirit in this development lies very much in the calculational prospect of abstract algebra – tedious model-theoretic reasoning can

* Supported by TUCS (Turku Centre for Computer Science).

be reduced to simple, perspicuous, symbol-pushing calculations. One important development are the modal semirings by Desharnais, Möller and Struth [10]. A modal semiring is a semiring structure including a domain operator facilitating the definition of modal operators in the sense of dynamic logic.

Our intent in this paper is to let yet another stream run up to Segerberg's uniting work by viewing some aspects of dynamic doxastic logic from the point of modal semirings. In this paper we propose a modal semiring extended with an epistemic-action operator: a *dynamic epistemic semiring*. This structure allows us to reason about an agent's changing beliefs in a transparent, calculational fashion. The carrier elements of the algebra are viewed as epistemic actions – actions working on the agent's beliefs. To check whether the agent believes a proposition we introduce special actions: *epistemic tests*. Epistemic tests work like guards in program theory, *i.e.* programs that check if some predicate holds or not.

In comparison to the semi-formal approach of the AGM trio, and the logical approaches of Segerberg [20, 21, 22]; Baltag, Moss and Solecki [5, 4] and the so called Amsterdam school centered around van Benthem [7] – our approach is abstract-algebraic. The work that comes closest to ours is the elegant algebraic approach to dynamic epistemic reasoning by Baltag, Coecke, and Sadrzadeh [2, 3, 6]. But when they use systems (a combination of a complete sup-lattice, a quantale and a right-module structure on the lattice) as the underlying structure in order to achieve an axiomatisation, we use modal semirings. Baltag, Coecke and Sadrzadeh's approach is perhaps more thoroughly developed to match the original framework by Alchourrón, Gärdenfors and Makinson than ours, whereas our treatment, on the other hand, is more perspicuous in that it has a (at least to our mind) simpler algebraic structure underneath, and it also easily facilitates iterated action.

The structure of the paper is: Sect. 2 defines modal semirings. In Sect. 3 the revision operator is introduced, in Sect. 4 we provide a relational model and in Sect. 5 we develop a basic calculus. In Sect. 6 we derive the classical AGM-axioms for revision in the algebra. The final section contains some concluding remarks. Only the most interesting proofs are included in the paper; App. A contains detailed proofs of the claims that are not proved.

We want to emphasise that the main purpose of *this* paper not is to consider applications or the philosophy of formal epistemology, but to lay an initial foundation to using semirings for reasoning about belief in change. The paper, then, primarily provides a concise technical framework, that connects earlier work on formal epistemic reasoning with the recent work on applications of semirings.

2 Modal Semirings

By an *idempotent semiring* we shall understand a structure over the signature $(+, ;, 0, 1)$ such that the reduct over $(+, 0)$ is a commutative and idempotent monoid, and the reduct over $(;, 1)$ is a monoid such that $;$ distributes over $+$ and has 0 as a left and right annihilator. When no risk for confusion arises $;$ will

be left implicit. An idempotent semiring thus satisfies the following axioms (a, b and c in the carrier set):

$$a + (b + c) = (a + b) + c, \quad (1)$$

$$a + b = b + a, \quad (2)$$

$$a + 0 = a, \quad (3)$$

$$a + a = a, \quad (4)$$

$$a(bc) = (ab)c, \quad (5)$$

$$1a = a = a1, \quad (6)$$

$$0a = 0 = a0, \quad (7)$$

$$a(b + c) = ab + ac \text{ and} \quad (8)$$

$$(a + b)c = ac + bc. \quad (9)$$

We define the *canonical order* \leq on a semiring by $a \leq b \Leftrightarrow_{df} a + b = b$ for all a and b in the carrier set. With respect to the canonical order, 0 is the least element, $;$ as well as $+$ are isotone and $+$ is join.

A *test semiring* [9, 16] is a two-sorted algebra

$$(S, \text{test}(S), +, ;, \neg, 0, 1)$$

such that

- $(S, +, ;, 0, 1)$ is an idempotent semiring,
- $(\text{test}(S), +, ;, \neg, 0, 1)$ is a Boolean algebra (BA) and
- $\text{test}(S) \subseteq S$.

Join and meet in $\text{test}(S)$ are thus $+$ and $;$, respectively, and the complement is denoted by \neg ; 0 is the least and 1 is the greatest element. We shall use a, b, \dots for general semiring elements and p, q, \dots for test elements. On a test semiring we axiomatise a *domain operator* $\ulcorner : S \rightarrow \text{test}(S)$ by

$$a \leq \ulcorner a ; a, \quad (10)$$

$$\ulcorner(pa) \leq p \text{ and} \quad (11)$$

$$\ulcorner(a\ulcorner b) \leq \ulcorner(ab), \quad (12)$$

for all $a \in S$ and $p \in \text{test}(S)$ [9]. Inequalities (10) and (12) can be strengthened to equalities. The domain operator satisfies stability of tests,

$$\ulcorner p = p, \quad (13)$$

and additivity,

$$\ulcorner(a + b) = \ulcorner a + \ulcorner b. \quad (14)$$

Isotony of domain follows from additivity. We will use $p \rightarrow q$ as a shorthand for $\neg p + q$. Note that

$$p \rightarrow q = 1 \Leftrightarrow p \leq q$$

follows from shunting. With the aid of the domain operator, we can define modal operators [10] by

$$\langle a \rangle p =_{df} \ulcorner (ap) \quad \text{and} \quad [a]p =_{df} \neg \langle a \rangle \neg p . \quad (15)$$

Therefore we shall call a test semiring with a domain operator *modal*. The diamond can be read “it is possible to successfully perform a so that p will hold” and the box as “after every successful way of performing a it will be the case that p holds”.¹ The following fact

$$\text{if } p \leq q \text{ and } [a]p = 1, \text{ then } [a]q = 1 \quad (16)$$

is not hard to prove and will be used later.

Example 1. (From [9]) Given two binary relations $S, T \subseteq \Sigma \times \Sigma$, define the binary operators $;$ and $+$ and the unary operator \ulcorner on the relations by the following:

$$(x, y) \in (S; T) \Leftrightarrow_{df} \exists z \in \Sigma \bullet (x, z) \in S \text{ and } (z, y) \in T, \quad (17)$$

$$(x, y) \in (S + T) \Leftrightarrow_{df} (x, y) \in S \text{ or } (x, y) \in T \text{ and} \quad (18)$$

$$(x, x) \in \ulcorner S \Leftrightarrow_{df} (x, y) \in S \text{ for some } y \in \Sigma. \quad (19)$$

Moreover, denote the identity relation by Δ , let \emptyset denote the empty relation and let \neg denote the complement relative to Δ . Then the structure

$$(\wp(\Sigma \times \Sigma), \wp(\Delta), +, ;, \neg, \ulcorner, \emptyset, \Delta)$$

is a modal semiring. ◇

3 Dynamic Epistemic Semirings

When reasoning about an agent’s beliefs and changes in the agent’s beliefs a common way to go is to talk about the agent’s *belief set* and operators that work on this set. The classical framework centered around Alchourrón, Gärdenfors and Makinson [1] had three such operators, namely

- *addition*: add a belief to the agent’s belief set,
- *revision*: add a belief to agent’s belief set and revise the set (without removing the added belief) so that it becomes consistent, and
- *contraction*: remove a belief and ensure that it is not implied by the remainder.

In the present paper we will concentrate on the revision operator – the heart of dynamic epistemic reasoning – but we will not have anything to say about

¹ In the tradition following Dijkstra [11] the box corresponds to the weakest liberal precondition of a with respect to q : if a terminates, p will hold. Our ‘successful’ corresponds to ‘terminating’.

exactly how the revision is done; what we are saying is that if a revision is done, then certain properties will hold.

It is possible for the agent to be ignorant (or indifferent) regarding a certain proposition: the agent might not believe a proposition, but this does not mean that he believes the negation of the proposition either. A non-ignorant agent thus believes in exactly one possible world, so to speak, whereas an ignorant agent possibly does not know exactly in which possible world (or state) he is in, but only in which subset of possible worlds he is in. For an ignorant agent, if the proposition holds or does not hold in all the possible worlds in his current belief set, then the agent believes or does not believe the proposition, respectively. If, however, this subset of possible worlds is a singleton set, then the agent of course knows exactly in which world he is; Baltag and Sadrzadeh call these sets “complete (fully determined) situations” [6].

We shall now define ‘dynamic epistemic semiring’, but first we provide an informal intuition. The carrier set should be seen as consisting of actions on the agent’s beliefs and *upon* these actions we shall work with four operators:

- $+$ is choice: do the left or the right action,
- $;$ is sequential composition of actions,
- $\ulcorner a$ denotes those states of belief from which it is possible to successfully perform the action a , and
- $\otimes p$ revises the agents beliefs with p (it is always possible to successfully perform this action, apart from when p is the contradiction).

We shall have two distinguished actions, the fail action 0 which always fails and the idle action 1 that leaves everything as it is. The tests (defined according to the test semirings and named **et**) should be seen as actions that check whether the agent believes a proposition or not: *epistemic tests*. If the agent believes the proposition, the epistemic test will *hold* and behave like the idle action, otherwise it will behave like the fail action. If $a \leq b$ the action b results in anything that a would result in and possibly more. The diamond and the box can be interpreted as above.

We now propose a set of axioms for the revision operator. This set is to be seen as a mere suggestion, since the main thrust of this paper is not to promote a certain axiomatisation of revision, but to promote the abstract-algebraic method. These axioms are chosen because we find them reasonable and because the standard AGM-axioms for revision follow from them (see Sect 6); depending on one’s needs, another axiomatisation could be chosen and the abstract-algebraic framework should then work equally well.

Definition 2. A *dynamic epistemic semiring* (DES) is a two-sorted algebra

$$\mathcal{D} = (D, \text{et}(D), +, ;, \neg, \otimes, \ulcorner, 0, 1)$$

such that the reduct structure over

$$(D, \text{et}(D), +, ;, \neg, \ulcorner, 0, 1)$$

is a modal semiring and $\otimes : \text{et}(D) \rightarrow D$ is a unary prefix operator such that

$$p \leq \circledast p, \quad (20)$$

$$\circledast p; \circledast p \leq \circledast p, \quad (21)$$

$$\circledast p; q \leq \circledast (pq), \quad (22)$$

$$\circledast (p + q); p = \circledast p, \quad (23)$$

$$\circledast p \leq \circledast q \Rightarrow p \leq q \text{ and} \quad (24)$$

$$\top \circledast p = 1, \ p \neq 0 \quad (25)$$

hold for any $p, q \in \text{et}(D)$ when \leq is the canonical order. \triangleleft

An intuition for the axioms can briefly be given as follows. The first axiom says that successfully performing a choice between p and $\circledast p$ is equal to successfully performing $\circledast p$. The second axiom says that revising with p and then revising with p again can be done away with by revising with p just once. The third axiom says that revising with p and then checking that q holds can be replaced by a revision with the conjunction p and q .² If the agent first revises his beliefs by the disjunction $p+q$ and then – with success – checks if p holds this is equal to revising his beliefs with p ; this is what the fourth axiom says. The fifth one says that if a successful execution of a choice between revising with p or q is equal to directly revising with q then p implies q . The last axiom says that the action $\circledast p$ can always be performed successfully as long as p is not the contradiction.

4 A Relational Model for Revision

We now give a relational interpretation satisfying the axioms of the above structure. However, we want to emphasise that this rough relational model is not to be seen as *the* model for our algebra; it is simply an example of a structure that satisfies our axioms (showing that our axiomatisation has got a model and, so, is sound). It does not do much for intuition, for example, revision in this model is “absolute” in that all previous knowledge is forgotten when a revision is performed – this, however, should not follow from the axioms. There are other, more suitable, relational models for belief revision, but since our main concern in this paper is the abstract algebra, we shall not go into these. That these more “realistic” models can be connected with our algebra follows from the fact that counterparts to the classical AGM-axioms can be derived in our framework (see Sect. 6).

Example 3. Let Σ be any set and let Δ be the identity relation. Define the unary prefix operator

$$\circledast : \wp(\Delta) \rightarrow \wp(\Sigma \times \Sigma)$$

by, for any given $P \subseteq \Delta$,

$$\circledast P =_{df} \{(x, p) \mid x \in \Sigma \text{ and } (p, p) \in P\}. \quad (26)$$

² The inequalities (21) and (22) can in fact be strengthened to equalities.

Then the structure

$$(\wp(\Sigma \times \Sigma), \wp(\Delta), ;, +, \neg, \ulcorner, \circledast, \emptyset, \Delta)$$

is a DES when the operators and the constants are interpreted as above and as in Ex. 1.

5 A Basic Calculus

In this section we develop a basic calculus for DESs. The first two propositions are helpful tools.

Proposition 1. *In any DES the equations*

$$\circledast p; p = \circledast p, \quad (27)$$

$$p \leq p; \circledast p \text{ and} \quad (28)$$

$$\circledast p; \neg p = 0 \quad (29)$$

hold.

Statement (27) above says that revising the agent's beliefs by p and then checking whether the agent believes p is equal to just revising the agent's beliefs with p . The epistemic test will act as an idle action, since the agent certainly believes a proposition after just having added it. Statement (29) says something similar: after revising with p the epistemic test $\neg p$ will fail. Statement (28) says that p could always be replaced by $p; \circledast p$. Note, however, that the other direction does not hold, since revision by p might result in the agent "losing" information. Having p hold before the revision does not mean that the revision is idle: the revision might weaken the agent's beliefs.

The next proposition settles that $\circledast p$ is an idempotent element for $;$ (for any p). This means that repeated revision with the same predicate has no more effect than revising once.

Proposition 2. *In any DES the equation*

$$\circledast p; \circledast p = \circledast p \quad (30)$$

holds.

Next we establish additivity and isotony of \circledast . The intuition is that if the agent is to change his beliefs to a set of states where p or q holds this is equal to a choice between changing it to a set of states where p holds or changing it to a set of states where q holds. The intuition behind the isotony is similar to the one of axiom (24).

Proposition 3. *In any DES the equations*

$$\circledast (p + q) = \circledast p + \circledast q \text{ and} \quad (31)$$

$$p \leq q \Rightarrow \circledast p \leq \circledast q \quad (32)$$

hold.

The following proposition establishes some relations between the constants, the domain operator and the epistemic operators. As shown by the first and the second statement, respectively, revising with the contradiction always fails and this action cannot be performed.

Proposition 4. *In any DES the equations*

$$\otimes 0 = 0 \text{ and} \quad (33)$$

$$\ulcorner \otimes 0 = 0 \quad (34)$$

hold.

6 Deriving the AGM Axioms for Revision

To connect our framework to the classical work on belief revision, we now show that the well-known AGM axioms for the revision operator can be derived in our algebra. First, we encode the axioms following the encoding of the revision axioms into dynamic doxastic logic (see for example Cantwell [8]). We refrain from writing out the intuition of the axioms, but just recall that $p \rightarrow q = 1 \Leftrightarrow p \leq q$ and note that, informally, p should be seen as $\mathbf{B}p$ in the language of dynamic doxastic logic and as $p \in K$ in the language of AGM, when K is the belief set; $[\otimes p]$ should be seen as $[*p]$ in dynamic doxastic logic and as $K * p$ in AGM theory.

The AGM axioms for revision are then encoded in our algebra as follows:

$$[\otimes p]p = 1, \quad (*)2$$

$$[\otimes p]q \leq p \rightarrow q, \quad (*)3$$

$$\neg\neg p; (p \rightarrow q) \leq [\otimes p]q, \quad (*)4$$

$$[\otimes p]\neg 0 = 1, \text{ when } p \neq 0, \quad (*)5$$

$$[\otimes p]r = [\otimes q]r, \text{ whenever } p = q, \quad (*)6$$

$$[\otimes(pq)]r \leq [\otimes p](q \rightarrow r) \text{ and} \quad (*)7$$

$$[\otimes p]q; [\otimes p](q \rightarrow r) \leq [\otimes(pq)]r. \quad (*)8$$

The first classical AGM axiom (*)1 for revision says that when revising a proposition, the agent should believe all its consequences. This is taken care of by the underlying epistemic-test structure, and need thus not be stipulated explicitly (similarly to dynamic doxastic logic, where it follows from the normality of the underlying logic). The epistemic-test structure also entails that the double negation in axiom (*)4 disappears and that $\neg 0$ in axiom (*)5 equals 1.

We now derive axioms (*)2)–(*)8) as calculational derivations in the algebra. The second axiom (*)2) can be shown to follow from the algebra by the calculation

$$\begin{aligned} & [\otimes p]p = 1 \\ \Leftrightarrow & \{\text{definition of box (15)}\} \\ & \neg\ulcorner(\otimes p\neg p) = 1 \\ \Leftrightarrow & \{(29)\} \end{aligned}$$

$$\begin{aligned}
& \neg\top = 1 \\
& \Leftrightarrow \{\text{modal semiring properties}\} \\
& \text{true.}
\end{aligned}$$

The third axiom (*3) follows from

$$\begin{aligned}
& [\otimes p]q \leq \neg p + q \\
& \Leftrightarrow \{\text{definition of box (15)}\} \\
& \neg\top(\otimes p; \neg q) \leq \neg p + q \\
& \Leftrightarrow \{\text{de Morgan}\} \\
& \neg\top(\otimes p; \neg q) \leq \neg(p; \neg q) \\
& \Leftrightarrow \{\text{stability of tests}\} \\
& \neg\top(\otimes p; \neg q) \leq \neg\top(p; \neg q) \\
& \Leftrightarrow \{\text{contraposition}\} \\
& \top(p; \neg q) \leq \top(\otimes p; \neg q) \\
& \Leftarrow \{\text{isotony}\} \\
& p \leq \otimes p \\
& \Leftrightarrow \{\text{axiom}\} \\
& \text{true,}
\end{aligned}$$

and the fourth axiom (*4) follows from (*2), the definition of \rightarrow and isotony of box (16). The fifth axiom follows from

$$\begin{aligned}
& [\otimes p]\neg 0 = 1 \\
& \Leftrightarrow \{\text{definition of box (15), double negation}\} \\
& \neg\top(\otimes p; 0) = 1 \\
& \Leftrightarrow \{\text{axiom}\} \\
& \neg\top 0 = 1 \\
& \Leftrightarrow \{\text{stability of tests, test BA}\} \\
& \text{true,}
\end{aligned}$$

and the sixth axiom (*6) is a direct consequence of the definition of box, the isotony of domain and the isotony of revision. The seventh axiom (*7) follows from the derivation

$$\begin{aligned}
& [\otimes(pq)]r \leq [\otimes p](q \rightarrow r) \\
& \Leftrightarrow \{\text{definition of box (15), contraposition, de Morgan}\} \\
& \top(\otimes p; q \neg r) \leq \top(\otimes(pq) \neg r) \\
& \Leftarrow \{\text{isotony}\} \\
& \otimes p; q \leq \otimes(pq) \\
& \Leftrightarrow \{\text{axiom}\} \\
& \text{true;}
\end{aligned}$$

and the derivation

$$\begin{aligned}
& [\otimes p]q; [\otimes p](q \rightarrow r) \leq [\otimes(pq)]r \\
& \Leftrightarrow \{\text{definition of box (15), de Morgan}\} \\
& \neg(\top(\otimes p; \neg q) + \top(\otimes p; q; \neg r)) \leq \neg\top(\otimes(pq); \neg r) \\
& \Leftrightarrow \{\text{contraposition, additivity of domain}\}
\end{aligned}$$

$$\begin{aligned}
& \ulcorner (\otimes(pq); \neg r) \leq \ulcorner (\otimes p; \neg q + \otimes p; q; \neg r) \\
& \Leftrightarrow \{\text{axiom, absorption}\} \\
& \ulcorner (\otimes(pq); \neg r) \leq \ulcorner (\otimes p; (\neg q + \neg r)) \\
& \Leftarrow \{\text{isotony}\} \\
& \text{true}
\end{aligned}$$

shows that the last axiom (*8) follows.

7 Concluding Remarks

This paper takes a modest first step towards epistemic reasoning with semirings. A possible elaboration is to incorporate real action into the framework (assuming that the beliefs are not a part of the world, real actions are actions on the world), and another possibility is to extend the underlying semiring to a Kleene algebra in order to enable reasoning about iterated action – we pursue this in [24]. In forthcoming work, we intend to look at contraction and addition in a similar framework. How more elaborate models than the relational one presented in this paper (such as hypertheories) relate to our algebra is also an interesting question that we shall return to. Finally, many of the recent abstract-algebraic approaches have proved practical in applications, see for example [27, 17, 23]; in an epistemic context, modal semirings have recently been applied by Möller [18]. It is a promising thought that the dynamic epistemic semiring would prove useful in applications involving epistemic notions.

Acknowledgements. The author is grateful to Bernhard Möller, Larissa Meinicke, Georg Struth and Peter Höfner for helpful remarks. He also highly appreciated the remarks of the anonymous referees – due to lack of time, however, not all suggestions could be followed.

References

1. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic* 50(2), 510–530 (1985)
2. Baltag, A., Coecke, B., Sadrzadeh, M.: Epistemic actions as resources. In: *Proceedings of the Logics for Resources Processes Programs Workshop at LiCS 2004* (2004)
3. Baltag, A., Coecke, B., Sadrzadeh, M.: Algebra and sequent calculus for epistemic actions. *Electronic Notes in Theoretical Computer Science* 126, 27–52 (2005)
4. Baltag, A., Moss, L.: Logic for epistemic programs. *Synthese* 139(2), 165–224 (2004)
5. Baltag, A., Moss, L., Solecki, S.: The logic of common knowledge, public announcements, and private suspicions. In: Gilboa, I. (ed.) *Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge (TARK’98)*, pp. 43–56 (1998)
6. Baltag, A., Sadrzadeh, M.: The algebra of multi-agent dynamic belief revision. *Electronic Notes in Theoretical Computer Science* 157(4), 37–56 (2006)
7. van Benthem, J.: *Dynamic logic for belief revision*. ILLC Technical Report, University of Amsterdam, (2006)

8. Cantwell, J.: Non-Linear Belief Revision. Foundations and Applications. Dissertation. Uppsala: Acta Universitatis Upsaliensis (2000)
9. Desharnais, J., Möller, B., Struth, G.: Kleene algebra with domain. *ACM Transactions on Computational Logic* 7(4), 798–833 (2006)
10. Desharnais, J., Möller, B., Struth, G.: Modal Kleene algebra and applications. *Journal on Relational Methods in Computer Science* 1(1), 93–131 (2004)
11. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall, Englewood Cliffs (1976)
12. Fagin, R., Halpern, J., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge, MA (1995)
13. Hintikka, J.: Knowledge and Belief: an Introduction to the Logic of the Two Notions. Cornell University Press, Ithaca, N. Y (1962)
14. Meyer, J.-J.C., van der Hoek, W.: Epistemic Logic for Computer Science and Artificial Intelligence. In: Meyer, J.-J. (ed.) *Cambridge Tracts in Theoretical Computer Science* 41, Cambridge University Press, Cambridge (1995)
15. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation* 110(2), 366–390 (1994)
16. Kozen, D.: Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems* 19(3), 427–443 (1997)
17. Möller, B.: Lazy Kleene algebra. In: Kozen, D. (ed.) *MPC 2004. LNCS*, vol. 3125, pp. 252–273. Springer, Heidelberg (2004)
18. Möller, B.: Knowledge and games in nodal semirings. *Technischer Bericht 2007-3*, Institut für Informatik, Universität Augsburg (February 2007)
19. de Rijke, M.: Meeting some neighbours: A dynamic modal logic meets theories of change and knowledge representation. In: van Eijk, J., Visser, A. (eds.) *Logic and information flow*, pp. 170–196. MIT Press, Cambridge, MA (1994)
20. Segerberg, K.: Proposal for a theory of belief revision along the lines of Lindström and Rabinowicz. *Fundamenta Informatica* 32, 183–191 (1997)
21. Segerberg, K.: Two traditions in the logic of belief: bringing them together. In: Ohlback, H.J., Reyle, U. (eds.) *Logic, Language and Reasoning*, pp. 134–147. Kluwer, the Netherlands (1999)
22. Segerberg, K.: The basic dynamic doxastic logic of AGM. In: Williams, M.-A., Rott, H. (eds.) *Frontiers in Belief Revision*, pp. 57–84. Kluwer, the Netherlands (1999)
23. Solin, K.: On two dually nondeterministic refinement algebras. In: Schmidt, R.A. (ed.) *RelMiCS/AKA 2006. LNCS*, vol. 4136, pp. 373–387. Springer, Heidelberg (2006)
24. Solin, K.: Dynamic Epistemic Semirings. Manuscript, Åbo Akademi, March, 2007. Preliminary version as *Technischer Bericht 2006-17*, Institut für Informatik, Universität Augsburg, (June 2006)
25. Wooldridge, M.: An Introduction to Multiagent Systems. John Wiley & Sons, England (2002)
26. Wright, G.H., von Wright, G.H.: An Essay in Modal Logic. North-Holland, Amsterdam (1951)
27. von Wright, J.: von: From Kleene algebra to refinement algebra. In: Boiten, E.A., Möller, B. (eds.) *MPC 2002. LNCS*, vol. 2386, Springer, Heidelberg (2002)

A Detailed Proofs

Soundness with Respect to the Relational Model

The only axioms that need to be verified are those concerning \otimes , the rest have been verified in [9]. The first axiom is verified by

$$\begin{aligned}
 & (p, p) \in P \\
 \Rightarrow & \{\text{property of binary relations}\} \\
 & (p, p) \in \{(x, p) \mid x \in \Sigma \text{ and } (p, p) \in \Sigma\} \\
 \Leftrightarrow & \{\text{definition of } \otimes\} \\
 & (p, p) \in \otimes P,
 \end{aligned}$$

the second by

$$\begin{aligned}
 & (x, y) \in \otimes P; \otimes P \\
 \Leftrightarrow & \{\text{definition of } ;\} \\
 & \exists z \in \Sigma \bullet (x, z) \in \otimes P \text{ and } (z, y) \in \otimes P \\
 \Rightarrow & \{\text{definition of } \otimes\} \\
 & (x, y) \in \otimes P,
 \end{aligned}$$

the third by

$$\begin{aligned}
 & (x, y) \in \otimes P; Q \\
 \Leftrightarrow & \{\text{definition of } ;\} \\
 & \exists z \in \Sigma \bullet (x, z) \in \otimes P \text{ and } (z, y) \in Q \\
 \Leftrightarrow & \{\text{definition of } \otimes P\} \\
 & \exists z \in \Sigma \bullet (x, z) \in \otimes P, (z, z) \in P \text{ and } (z, y) \in Q \\
 \Rightarrow & \{P, Q \subseteq \Delta\} \\
 & \exists z \in \Sigma \bullet (x, z) \in \otimes P, (z, z) \in P \cap Q \text{ and } z = y \\
 \Rightarrow & \{P, Q \subseteq \Delta, \text{ so } P \cap Q = P; Q\} \\
 & (x, y) \in \otimes(P; Q)
 \end{aligned}$$

the fourth by

$$\begin{aligned}
 & (x, y) \in \otimes(P \cup Q); P \\
 \Leftrightarrow & \{\text{definition of } ; \text{ and } P \subseteq \Delta\} \\
 & (x, y) \in \otimes(P \cup Q) \text{ and } (y, y) \in P \\
 \Leftrightarrow & \{\text{definition of } \otimes\} \\
 & (x, y) \in \otimes P,
 \end{aligned}$$

the fifth by

$$\begin{aligned}
 & \otimes P \subseteq \otimes Q \\
 \Leftrightarrow & \{\text{definition of } \otimes\} \\
 & \{(x, p) \mid x \in \Sigma \text{ and } (p, p) \in P\} \subseteq \{(x, q) \mid x \in \Sigma \text{ and } (q, q) \in Q\} \\
 \Rightarrow & \{\text{property of binary relations}\} \\
 & P \subseteq Q
 \end{aligned}$$

and, finally,

$$\begin{aligned}
 & \ulcorner \otimes P \\
 \Leftrightarrow & \{ \text{definition of } \otimes \} \\
 & \ulcorner \{ (x, p) \mid x \in \Sigma \text{ and } (p, p) \in P \} \\
 \Leftrightarrow & \{ \text{definition of } \ulcorner \text{ and } P \neq \emptyset \} \\
 & \Delta
 \end{aligned}$$

verifies the last axiom.

Proof of Proposition 1

The first statement follows from axiom (23) setting $q = 0$. The second statement follows from epistemic tests being a BA, isotony of $;$ and axiom (20). The third statement is proved by

$$\begin{aligned}
 & \otimes p \neg p \\
 = & \{ (27) \} \\
 & \otimes pp \neg p \\
 = & \{ \text{epistemic tests BA} \} \\
 & \otimes p 0 \\
 = & \{ 0 \text{ strict} \} \\
 & 0.
 \end{aligned}$$

Proof of Proposition 2

The statement is proved by

$$\begin{aligned}
 & \otimes p \\
 = & \{ (27) \} \\
 & \otimes pp \\
 \leq & \{ (20) \} \\
 & \otimes p \otimes p
 \end{aligned}$$

and axiom (21).

Proof of Proposition 3

The first statement (additivity) is proved by

$$\begin{aligned}
 & \otimes (p + q) \\
 = & \{ (27) \} \\
 & \otimes (p + q)(p + q) \\
 = & \{ ; \text{ distributive over } + \} \\
 & \otimes (p + q)p + \otimes (p + q)q \\
 = & \{ (23) \text{ and commutativity of } + \} \\
 & \otimes p + \otimes q,
 \end{aligned}$$

the second one (isotony) is straightforward from the first.

Proof of Proposition 4

The first part is proved by

$$\begin{aligned} & \circledast 0 \\ &= \{(27)\} \\ & \circledast 00 \\ &= \{0 \text{ strict}\} \\ & 0. \end{aligned}$$

The second part follows from the first statement and the fact that $\top 0 = 0$.

A Modal Distributive Law (abstract)

Yde Venema^{*}

Institute for Logic, Language and Computation, Universiteit van Amsterdam
Plantage Muidergracht 24, 1018 TV Amsterdam, Netherlands
yde@science.uva.nl

In our talk we consider the coalgebraic, or *cover* modality ∇ which, in modal logic, goes back to the work of Fine on normal forms. The connective, taking a *set* of formulas as its argument, can be defined in terms of the usual box and diamond operators:

$$\nabla\Phi := \Box \bigvee \Phi \wedge \bigwedge \Diamond\Phi,$$

where $\Diamond\Phi$ denotes the set $\{\Diamond\varphi \mid \varphi \in \Phi\}$. Conversely, the standard modalities can be defined in terms of the cover modality, from which it follows that we may equivalently base our modal language on ∇ as a primitive symbol:

$$\begin{aligned}\Diamond\varphi &\equiv \nabla\{\varphi, \top\} \\ \Box\varphi &\equiv \nabla\emptyset \vee \nabla\{\varphi\}.\end{aligned}$$

The main purpose of the talk is to underline the fundamental importance of the following equivalence, which we call a *distributive* law since it shows how conjunctions distribute over nabla's (and disjunctions):

$$\nabla\Phi \wedge \nabla\Phi' \equiv \bigvee_{Z \in \Phi \bowtie \Phi'} \nabla\{\varphi \wedge \varphi' \mid (\varphi, \varphi') \in Z\}. \quad (1)$$

Here $\Phi \bowtie \Phi'$ denotes the set of relations $R \subseteq \Phi \times \Phi'$ that are *full* in the sense that for all $\varphi \in \Phi$ there is a $\varphi' \in \Phi'$ with $(\varphi, \varphi') \in R$, and vice versa.

As an important corollary of (1) and the equidefinability of ∇ and $\{\Box, \Diamond\}$, every modal formula can be rewritten into a *distributive normal form* in which the *only* conjunctions are of the form $\bigwedge P \wedge \nabla\Phi$, where P is a set of literals, i.e., proposition letters and negations of proposition letters.

We then move on to some consequences of this fact. First we show that formulas in distributive normal form allow a direct, inductive definition of uniform interpolants. And second, we briefly review the work of Janin and Walukiewicz on automata for the modal μ -calculus (which is where the notion of distributive normal form stems from).

Finally, we turn to some of our recent work in this area. We discuss our axiomatization of (positive) modal logic based on the cover modality, and, time permitting, sketch its connection with the Vietoris construction in topology. We finish with putting the cover modality into the wider, coalgebraic perspective, introduced by Moss, and discuss the meaning of the associated distributive law in this more general setting.

^{*} The research of the author has been made possible by VICI grant 639.073.501 of the Netherlands Organization for Scientific Research (NWO).

Ant Colony Optimization with Adaptive Fitness Function for Satisfiability Testing

Marcos Villagra^{1,2} and Benjamín Barán²

¹ Catholic University of Asuncion
Asuncion, P.O. Box 1683, Paraguay
`marcos_villagra@uca.edu.py`

² National University of Asuncion
San Lorenzo, P.O. Box, 2169, Paraguay
`bbaran@pol.una.py`

Abstract. Ant Colony Optimization (ACO) is a metaheuristic inspired by the foraging behavior of ant colonies that has been successful in the resolution of hard combinatorial optimization problems. This work proposes MaxMin-SAT, an ACO alternative for the satisfiability problem (SAT). MaxMin-SAT is the first ACO algorithm for SAT that implements an Adaptive Fitness Function, which is a technique used for Genetic Algorithms to escape local optima. To show effectiveness of this technique, three different adaptive fitness functions are compared: Stepwise Adaptation of Weights, Refining Functions, and a mix of the previous two. To experimentally test MaxMin-SAT, a comparison with Walksat (a successful local search algorithm) is presented. Even though MaxMin-SAT cannot beat Walksat when dealing with phase transition instances, experimental results show that it can be competitive with the local search heuristic for overconstrained instances.

Keywords: SAT, Ant Colony Optimization, Local Search, Adaptive Fitness Function.

1 Introduction

The satisfiability problem in propositional logic or SAT is the problem of finding an assignment of values to Boolean variables for a given propositional formula to make it true. SAT was the first problem proved to be \mathcal{NP} -complete [1], and as such is amongst the most important problems in computer science.

There exist several algorithms to solve SAT. These algorithms can be divided in two classes: *Complete* algorithms and *Incomplete* algorithms. Complete algorithms state whether a given formula is true or not and return an assignment of values to variables. One of the most efficient algorithms is the Davis-Putnam method [2, 3]. Nowadays, the best performing complete algorithms are based on this method [4]. Incomplete algorithms try to find an assignment that approximates the satisfiability of the formula. These algorithms are mainly based on local search, where Walksat [5] is one of the most famous.

Ant Colony Optimization (ACO) is a metaheuristic proposed by Dorigo et al. [6] that is inspired by the foraging behavior of ant colonies. ACO has empirically shown its effectiveness in the resolution of several different \mathcal{NP} -hard combinatorial optimization problems [7]. Despite all this success, ACO cannot compete against state-of-the-art local search algorithms for SAT [8]. The scientific community is more interested in solving Constraint Satisfaction Problems (CSP) than SAT¹. Hence, there are more ACO algorithms solving CSPs [9,10,11]. One of them is Ant-Solver [10], which uses a generic CSP heuristic for choosing the next variable to assign a value. Also, it includes a novel pheromone mechanism based on the solution constructed so far and a preprocessing stage. Roli [12] proposed one of the first ACO algorithms for SAT, named ACO-MAXSAT. This algorithm was based on the Ant Colony System [13], and uses a heuristic based on the *breakcount* of the Walksat algorithm [5], i.e., number of satisfied clauses that would become unsatisfied. This approach showed to be inefficient since the heuristic need to be calculated each time a variable assignment is made.

Evolutionary algorithms also had problems solving SAT [14]. In general, population-based metaheuristics have not shown to be competitive when compared against local search algorithms [8]. Previous works showed classical genetic algorithms being unsuitable for the MAX-SAT fitness function [15]. However, other authors suggest that evolutionary algorithms can yield good results for SAT when equipped with additional techniques like adaptive fitness functions, problem-specific variation operators, and local optimization [14]. In that sense, this work proposes an ACO alternative for SAT called MaxMin-SAT, derived from the $\mathcal{MAX} - \mathcal{MIN}$ Ant System [7], that implements several adaptive fitness functions [14]. This is the first time that adaptive fitness function techniques are used with ACO algorithms, which is expected to get similar improvements than the ones obtained for genetic algorithms [14].

The remainder of this work is organized as follows. Section 2 presents the SAT problem. Section 3 introduces the ACO metaheuristic and the way to code SAT for this kind of algorithms. In section 4, the concept of adaptive fitness function is introduced. Section 5 presents experimental results. Finally, the conclusions and future work are left for section 6.

2 The SAT Problem

SAT is defined by a set of Boolean variables $X = \{x_1, \dots, x_n\}$ and a Boolean formula or sentence $\phi : \{0, 1\}^n \rightarrow \{0, 1\}$ where n represents the number of variables. The objective is to find a variable assignment $m = \langle x_1, \dots, x_n \rangle \in \{0, 1\}^n$ (i.e., a model) where $\phi(m) = 1$. The formula is called *satisfiable* if such a model exists, and *unsatisfiable* otherwise. A literal is a variable (also called atom or symbol) or its negation. A clause is a disjunction of literals, i.e., literals connected by an \vee operator. The formula ϕ is in conjunctive normal form (CNF) if $\phi(m) = c_1(m) \wedge \dots \wedge c_p(m)$, where each c_i is a clause and p is the number of clauses. The family of k -CNF sentences has exactly k literals per clause. SAT

¹ SAT is a particular case of CSP.

can be assumed having CNF formulae without loss of generality [16], and k -SAT only contains sentences in k -CNF. While 2-SAT is solvable in polynomial time, k -SAT is \mathcal{NP} -complete for $k \geq 3$ [17].

MAX-SAT is an optimization variant of SAT. Given a set of clauses, MAX-SAT is the problem of finding a model m that maximizes the number of satisfied clauses $f(m)$. In *weighted* MAX-SAT each clause c_i has a weight w_i assigned to it, while *unweighted* MAX-SAT uses $w_i = 1$ for all i . The following equation defines the MAX-SAT objective function:

$$f(m) = \sum_{i=1}^p w_i \cdot c_i(m) \quad (1)$$

where $c_i(m) = 1$ if the clause c_i is satisfied under the model m , and 0 otherwise.

Thus, when searching for a solution to MAX-SAT, one is actually looking for models to SAT formulae. Therefore, solutions refer to models and vice versa. There are only some differences between particular cases of SAT and MAX-SAT, where MAX- k -SAT is \mathcal{NP} -hard for $k \geq 2$ [17].

In 3-SAT, rapid transitions in solvability can be observed [18]. This phenomenon is called *phase transition*. If the problem has a very few number of clauses it is said to be *underconstrained* and it is easy to find a model. If there is a large number of clauses it is said to be *overconstrained* and a good heuristic will be able to quickly cut-off most of the search tree. Phase Transition occurs when going through instances that are underconstrained to instances that are overconstrained [19]. Hard instances are between these two kinds of instances, when 50% of them are satisfiable with approximately $\frac{p}{n} \approx 4.3$ [19]. This point is known as the *crossover point* [19].

3 Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic inspired by the foraging behavior of ant colonies [6]. ACO algorithms make use of simple agents called *ants* which iteratively construct candidate solutions. The ants are guided by (artificial) pheromone trails and problem-dependent heuristic information. This work presents MaxMin-SAT, derived from the $\mathcal{MAX} - \mathcal{MIN}$ Ant System algorithm proposed in [7], and adjusted to solve MAX-SAT. MaxMin-SAT follows the classical ACO algorithm scheme for static combinatorial optimization problems [6], as shown in algorithm 1.

The MAX-SAT problem is coded as a fully connected graph (called the construction graph) where artificial ants lay pheromone and search for a minimum path cost. Each node represents an assignment $\langle x_i, v_i \rangle$, where x_i is a Boolean variable and $v_i \in \{0, 1\}$ is the value assigned to the variable. Figure 1 shows the construction graph for an instance with three variables.

A model is represented as a path of length n on the construction graph, $m = \{\langle x_1, v_1 \rangle, \dots, \langle x_n, v_n \rangle\}$ such that for each $\langle x_i, v_i \rangle \in m$, then $\langle x_i, v_j \rangle \notin m$ for $v_i \neq v_j$. This constraint avoids cycles and inconsistencies in the solution [12].

Algorithm 1. MaxMin-SAT algorithmic scheme

Set parameters and initialize pheromone values

repeat

 for all ants in the colony **do**

Construct an assignment

end for

Update pheromone values

until end condition is met

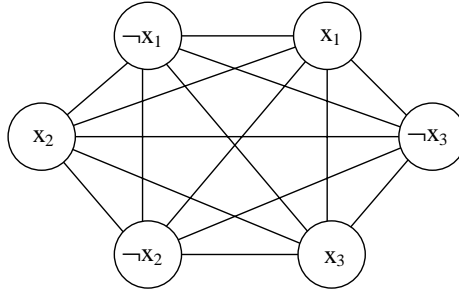


Fig. 1. Construction graph for a SAT instance with $n = 3$ variables. The correspondence between nodes and assignments are: $\langle x_1, 0 \rangle \equiv \neg x_1$, $\langle x_1, 1 \rangle \equiv x_1$, $\langle x_2, 0 \rangle \equiv \neg x_2$, $\langle x_2, 1 \rangle \equiv x_2$, $\langle x_3, 0 \rangle \equiv \neg x_3$, $\langle x_3, 1 \rangle \equiv x_3$.

Pheromone is deposited on nodes, where τ_i represents the quantity of pheromone on node i . The *heuristic information* (also known as visibility [6]) is denoted by η_i , and represents the desirability for an ant to choose node i . As proposed in the $\mathcal{MAX} - \mathcal{MIN}$ Ant System [7], lower and upper bounds are imposed on the pheromones with $0 < \tau_{min} < \tau_{max}$. The goal is to favor a larger exploration of the search space by preventing the relative differences between pheromone values from becoming too extreme during processing. Also, pheromone values are set to τ_{max} at the beginning of the algorithm, thus, achieving a higher exploration of the search space during the first cycles.

3.1 Construction of Assignments by Ants

The colony contains q ants. Let \mathcal{N} be the set of unvisited nodes, each ant a constructs a model m by choosing a node i in the construction graph according to the following probabilistic rule² [6]:

$$p_i^a = \begin{cases} \frac{\tau_i^\alpha \cdot \eta_i^\beta}{\sum_{j \in \mathcal{N}} \tau_j^\alpha \cdot \eta_j^\beta} & \text{if } i \in \mathcal{N} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

² Known as *random-proportional rule*.

The heuristic information η is the well-known *Most Constrained Variable* (MCV) heuristic used for CSPs [20], i.e., variables that appear in most clauses are more desirable by ants. Parameters α and β define the relative influence between the heuristic information and the pheromone levels. The construction of complete assignments by ants is depicted in algorithm 2.

Algorithm 2. Assignment construction algorithmic scheme

Construction of model m

```

 $m \leftarrow \emptyset$ 
while  $|m| < n$  do
    Choose an assignment  $\langle x, v \rangle$  with probability given by equation 2
     $m \leftarrow m \cup \{\langle x, v \rangle\}$ 
end while

```

3.2 Updating Pheromone Values

After every ant has constructed a model, pheromone values are updated according to ACO rules: (i) all pheromone values are decreased uniformly in order to simulate evaporation, allowing ants to forget bad assignments, and then (ii) the best ant of the iteration deposits pheromone. At the end of each cycle, the quantity of pheromone on each node $\langle x, v \rangle$ is updated as follows [7]:

$$\begin{aligned}
 \tau(\langle x, v \rangle) &\leftarrow \rho \cdot \tau(\langle x, v \rangle) + \Delta\tau(\langle x, v \rangle) \\
 \text{if } \tau(\langle x, v \rangle) < t_{min} &\text{ then } \tau(\langle x, v \rangle) \leftarrow \tau_{min} \\
 \text{if } \tau(\langle x, v \rangle) > t_{max} &\text{ then } \tau(\langle x, v \rangle) \leftarrow \tau_{max} \\
 \text{with } \Delta\tau(\langle x, v \rangle) &= \begin{cases} f(x) & \text{if } \langle x, v \rangle \in m^* \\ 0 & \text{if } \langle x, v \rangle \notin m^* \end{cases}
 \end{aligned} \tag{3}$$

where $(1 - \rho)$ is the evaporation parameter such that $0 \leq \rho \leq 1$ and m^* is the current best solution. The quantity of pheromone laid is directly proportional to the objective function $f(x)$, i.e., the number of satisfied clauses. The limits on the pheromone levels were established according the recommendations given in [7]: $\tau_{max} = \frac{p}{1-\rho}$ and $\tau_{min} = \frac{\tau_{max}}{2p}$.

4 Adaptive Fitness Functions

This work focuses on adaptive fitness functions (AFF) techniques for ACO using the MaxMin-SAT algorithm. Particularly, three kinds of AFFs are studied: Stepwise Adaptation of Weights (SAW) [21], Refining Functions (RF) [22], and a mix of the previous two.

4.1 Stepwise Adaptation of Weights (SAW)

The SAW principle uses the fitness function presented in equation 1. Each weight is initialized to 1, and in some stages some of them are increased in order to identify the clauses that are difficult to satisfy. In this work, every time 250 models are constructed, weights are updated according to the following formula:

$$w_i = w_i + 1 - c_i(m^*) \quad (4)$$

The number 250 was chosen following the works of Eiben and van der Hauw [23]. This adaptation scheme increases only those weights that correspond to unsatisfied clauses [23]. Thus, the weights bias the search towards clauses that are difficult to satisfy.

4.2 Refining Functions (RF)

The idea of refining functions was inspired by the observation that many different models have the same quality considering the MAX-SAT objective function (eq. 1), which makes impossible for an algorithm to distinguish between them [22]. Additional heuristic knowledge can be captured in a *refining function* $r : \{0, 1\}^n \rightarrow [0, 1]$ that maybe used within the following *refined fitness function* [14] which replaces equation 1:

$$f_{REF}(m) = f(m) + \theta \cdot r(m) \quad (5)$$

where the influence of r is controlled by $\theta > 0$. For this work θ was experimentally tuned to 1, what assures that if $f(m_1) > f(m_2)$ then $f_{REF}(m_1) > f_{REF}(m_2)$ given that $r(m) < 1$. Following the recommendation given in [23], the refining function used for this work is given by:

$$r(m) = \frac{1}{2} \left(1 + \frac{\sum_{i=1}^n K(x_i)u_i}{1 + \sum_{i=1}^n abs(u_i)} \right) \quad (6)$$

where $K : \{0, 1\} \rightarrow \{-1, 1\}$ is defined by $K(1) = 1$ and $K(0) = -1$, and $u_i \in \mathbb{R}$ is the weight associated to variable x_i . High positive weights indicate that the corresponding variables are favored to be 1, while negative weights express a preference to 0 [23].

In order to escape from local optima, the adaptation of u_i is defined as:

$$u_i = u_i - K(m_i^*) \cdot |U_i(m^*)| \quad (7)$$

where $U_i(m^*)$ is the set of unsatisfied clauses containing variable x_i when considering model m^* and $|\cdot|$ denotes cardinality. This scheme adjusts the weights u_i towards the complement of m^* and is referred as RF. Also, a hybridized scheme that incorporates SAW to accelerate the adaptation process is considered. The scheme, referred as RFSAW, updates the variable weights according to:

$$u_i = u_i - K(m_i^*) \sum_{c_j \in U_i(m^*)} w_j \quad (8)$$

where the clause weights are adjusted according to SAW. Note that RF is a special case of RFSAW with $w_i = 1$ [23]. In this work, variable weights are adapted during each iteration of the MaxMin-SAT algorithm.

5 Experimental Results

Performance of MaxMin-SAT using the different adaptive fitness function mentioned above is presented in this section. The well-known local search algorithm Walksat is used as a point of reference, given that ACO-MAXSAT performed very poorly in our experiments, coincident with results reported in [12].

5.1 Environment

Considering the incomplete nature of Walksat³ and the three proposed versions of MaxMin-SAT (SAW, RF, RFSAW), each algorithm runs 10 times on each benchmark. For the 3 ACO solvers, the number of iterations was limited to 10000, and for Walksat to 10000 tries.

For the experimental test, 50 randomly generated instances were created following the rules given in [24]. This set of 50 instances was divided in one set of 25 instances with $n = 50$ and $p = 215$, and another set of 25 instances with $n = 50$ and $p = 323$. Clearly, the first set contains phase transition instances, and the second set, overconstrained instances.

The algorithms were implemented using C++ compiled with gcc v4.1.0, and tested on a computer with a 4GHz Pentium 4 processor and 1GB of RAM. Walksat was imposed a maximum number of 10^6 flips per try, and noise parameter of 0.5. The parameters for MaxMin-SAT were experimentally tuned on a different set of random instances and set to: $q = 10$, $\rho = 0.95$, $\alpha = \beta = 1$.

5.2 Comparative Results

Three criterions are used to evaluate each version of MaxMin-SAT. The first one is the mean value of the objective function; the second is the success rate (SR); and the last criterion is the number of iterations of the ACO algorithms, or number of tries for Walksat.

Experimental results for phase transition and overconstrained instances can be seen in tables 1 and 2 respectively. When a problem is not solved, the number of unsatisfiable clauses of the best result is presented in parentheses.

Table 1 shows that Walksat clearly outperforms the ACO algorithms for the phase transition instances, as it is well known in the literature [12]. In fact, to this day, there is not a metaheuristic able to undoubtedly outperform Walksat on phase transition instances. However, table 2 shows that MaxMin-SAT can be competitive with Walksat for overconstrained instances, reporting a first relative success of an ACO based algorithm in the resolution of overconstrained MAX-SAT instances.

When comparing the three MaxMin-ACO alternatives proposed in this paper, it is clear that the adaptive fitness function RF outperforms the other two alternatives, as shown in tables 1 and 2.

³ The version of Walksat is v46.

Table 1. Experimental results for the phase transition instances

Phase Transition												
	SAW			RF			RFSAW			Walksat		
instance	mean	SR	iterations	mean	SR	iterations	mean	SR	iterations	mean	SR	tries
1	211		(1 clause)	212.4		(2 clauses)	210.8		(2 clauses)	215	1	3.8
2	212.4	0.2	276	212.4	0.1	119	212.1	0.1	81	215	1	1
3	212.5		(1 clause)	213.2	0.1	90	214	0.4	449	215	1	1
4	211.8		(1 clause)	212.6		(1 clause)	212.5		(1 clause)	215	1	1
5	209.6		(2 clauses)	211.9		(2 clauses)	208.6		(4 clauses)	214		(1 clause)
6	207.2		(4 clauses)	210.1		(2 clauses)	208.9		(2 clauses)	214		(1 clause)
7	209.8	0.1	102	211.2		(3 clauses)	210.3		(1 clause)	215	1	1.1
8	209.2		(3 clauses)	212.6		(2 clauses)	211		(1 clause)	215	1	1
9	207.2		(4 clauses)	211.7		(1 clause)	210.6		(1 clause)	215	1	1.1
10	210.8		(1 clauses)	211.7		(1 clause)	212.1		(1 clause)	215	1	1
11	212.2		(1 clause)	213.5		(1 clause)	212.1		(1 clause)	215	1	1
12	209.2		(4 clauses)	211.3		(2 clauses)	208.9		(2 clauses)	214		(1 clause)
13	211		(1 clause)	211.8		(1 clause)	211.1		(1 clause)	215	1	1.2
14	211.3		(1 clause)	212.1		(2 clauses)	211		(1 clause)	215	1	1.1
15	212.5	0.1	113	213.2		(1 clause)	212.9		(1 clause)	215	1	
16	210.2		(3 clauses)	212.1		(2 clauses)	209.8		(2 clauses)	214		(1 clause)
17	211		(1 clause)	211.5		(1 clause)	211.3	0.1	761	215	1	1.2
18	210.6	0.1	129	212.7	0.1	103	211.3	0.1	93	215	1	1
19	213.2	0.4	140.33	213.4	0.1	248	212.3		(1 clause)	215	1	1
20	210.7		(1 clause)	212.7		(1 clause)	211.9	0.1	130	215	1	1
21	208		(1 clause)	213		(1 clause)	211.6		(1 clause)	215	1	1.2
22	210.4		(1 clause)	211		(2 clauses)	208.9		(3 clauses)	215	1	4.9
23	212.3		(1 clause)	213.2		(1 clause)	212.4		(1 clause)	214		(1 clause)
24	211.3		(2 clauses)	212.9		(1 clause)	212.1		(2 clauses)	214		(1 clause)
25	210.9		(1 clause)	211.9		(2 clauses)	211.1		(1 clause)	214		(1 clause)

6 Conclusions and Future Work

Since, in the literature, there is no successful Ant Colony Optimization (ACO) algorithm for MAX-SAT, this work presents a new MAX-SAT solver based on Ant Colony Optimization which implements a technique known as Adaptive Fitness Function (AFF). This technique was selected to be used for the first time with ACO algorithms given its success for genetic algorithms. Three kinds of AFFs were compared, the Stepwise Adaptation of Weights (SAW), Refining Functions (RF), and a mix of the previous two (RFSAW). Experimental results showed that still population-based metaheuristic can not beat Walksat; however, it can improve a lot the quality of the solutions found, to the point of becoming competitive for overconstrained instances. This fact suggests that adaptive fitness functions techniques for ACO seems a promising approach. Also, it was experimentally proved that the best performing AFF seems to be RF, which found better approximations than the other two AFFs.

Still a lot of work remains to be done, since ACO can not compete with Walksat in phase transition instances, hence, other techniques can be added together with AFFs. An effective and efficient heuristic for selection of variables and clauses during the construction of models by ants can be considered. Research in hybrid ACO algorithms implementing techniques taken from complete methods is being carried out at this moment, with improvements on the model

Table 2. Experimental results for the overconstrained instances

Overconstrained											
instance	SAW			RF			RFSAW			Walksat	
	mean	SR	iterations	mean	SR	iterations	mean	SR	iterations	mean	SR tries
1	304.4	(10 clauses)		313.8	(7 clauses)		304.2	(14 clauses)		317	(6 clauses)
2	307.1	(6 clauses)		314.3	(7 clauses)		309.9	(6 clauses)		318	(5 clauses)
3	308.4	(10 clauses)		314.6	(7 clauses)		305.6	(12 clauses)		317	(6 clauses)
4	312	(8 clauses)		315.7	(5 clauses)		313.2	(6 clauses)		318	(5 clauses)
5	303	(15 clauses)		312.8	(8 clauses)		301.1	(17 clauses)		316	(7 clauses)
6	307.4	(13 clauses)		315.2	(6 clauses)		308	(8 clauses)		318	(5 clauses)
7	305.2	(11 clauses)		314.6	(7 clauses)		306	(13 clauses)		318	(5 clauses)
8	305.3	(11 clauses)		314.5	(6 clauses)		305.5	(8 clauses)		317	(6 clauses)
9	304.2	(13 clauses)		312.4	(8 clauses)		304.8	(13 clauses)		316	(7 clauses)
10	307	(11 clauses)		313.8	(7 clauses)		307.8	(9 clauses)		318	(5 clauses)
11	308	(12 clauses)		314.9	(7 clauses)		305.4	(16 clauses)		318	(5 clauses)
12	304.5	(12 clauses)		313.8	(7 clauses)		305.6	(10 clauses)		317	(6 clauses)
13	305.3	(13 clauses)		315.6	(6 clauses)		310.7	(9 clauses)		318	(5 clauses)
14	312.6	(5 clauses)		316	(4 clauses)		313.3	(5 clauses)		320	(3 clauses)
15	307.9	(9 clauses)		313.9	(7 clauses)		307.3	(9 clauses)		317	(6 clauses)
16	311.3	(8 clauses)		314.8	(6 clauses)		311.1	(7 clauses)		319	(4 clauses)
17	311.4	(6 clauses)		318.2	(4 clauses)		313.4	(6 clauses)		320	(3 clauses)
18	301.1	(16 clauses)		311.9	(9 clauses)		300.8	(16 clauses)		316	(7 clauses)
19	303.5	(12 clauses)		312.2	(10 clauses)		305.9	(12 clauses)		316	(7 clauses)
20	305.6	(13 clauses)		313.7	(8 clauses)		305.3	(11 clauses)		317	(6 clauses)
21	303.2	(13 clauses)		312.4	(8 clauses)		303.7	(14 clauses)		317	(6 clauses)
22	304.6	(13 clauses)		313.1	(8 clauses)		304.6	(12 clauses)		316	(7 clauses)
23	302.6	(16 clauses)		311.5	(10 clauses)		303.6	(10 clauses)		316	(7 clauses)
24	305.5	(11 clauses)		313.9	(7 clauses)		307.2	(11 clauses)		318	(5 clauses)
25	309.8	(9 clauses)		314.8	(7 clauses)		310.1	(8 clauses)		318	(5 clauses)

construction procedure and new ways to handle pheromone values. Also, ways to build a parallel team of algorithms cooperating together seems a promising approach according the first experiments.

References

1. Cook, S.: The complexity of theorem proving procedures. In: Proceedings of the 3rd. ACM Symposium on Theory of Computing, Shaker Heights, Ohio, pp. 151–156 (1971)
2. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM 7(3), 201–215 (1960)
3. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM 5, 394–397 (1962)
4. Prasad, M., Biere, A., Gupta, A.: A survey of recent advances in SAT-based formal verification. Software Tools for Technology Transfer 7(2), 156–173 (2005)
5. Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the AAAI’94. vol. 1, pp. 337–343 (1994)
6. Dorigo, M., Maniezzo, V., Coloni, A.: The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics 26(1), 1–13 (1996)
7. Stützle, T., Hoos, H.: $\mathcal{MAX} - \mathcal{MIN}$ ant system. Future Generation Computer Systems 16(8), 889–914 (2000)

8. Stützle, T., Hoos, H., Roli, A.: A review of the literature on local search algorithms for MAX-SAT. Technical Report AIDA-01-02, Technische Universität Darmstadt (2006)
9. Schoofs, L., Naudts, B.: Ant colonies are good at solving constraint satisfaction problems. In: Proceedings of the 2000 Congress on Evolutionary Computation. vol. 2 1190–1195 (2000)
10. Pimont, S., Solnon, C.: A generic ant algorithm for solving constraint satisfaction problems. In: Proceedings of ANTS 2000 (2000)
11. Solnon, C.: Ants can solve constraint satisfaction problems. IEEE Transactions on Evolutionary Computation 6(4), 347–357 (2002)
12. Roli, A.: Metaheuristics and Structure in Satisfiability Problems. PhD thesis, Facoltà de Ingegneria, Università degli Studi di Bologna (2003)
13. Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evolutionary Computation 1(1), 53–66 (1997)
14. Gottlieb, J., Marchiori, E., Rossi, C.: Evolutionary algorithms for the satisfiability problem. Evolutionary Computation 10(1), 35–50 (2002)
15. Rana, S., Whitley, D.: Genetic algorithm behavior in the maxsat domain. In: PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, pp. 785–794. Springer, Heidelberg (1998)
16. Tseitin, G.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic 2, 115–125 (1968)
17. Garey, M., Johnson, D.: Computers and Intractability: a Guide to the Theory of \mathcal{NP} -completeness. W.H. Freeman & Company, San Francisco, California (1979)
18. Cheeseman, P., Kanelfy, B., Taylor, W.: Where the really hard problems are. In: Proceedings of IJCAI'91, San Mateo, California, pp. 331–337. Morgan Kaufmann, San Francisco (1991)
19. Crawford, J., Auton, L.: Experimental results on the crossover point in satisfiability problems. In: Proceedings of the 11th National Conference on Artificial Intelligence (1993)
20. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2003)
21. Eiben, A., van der Hauw, J.: Solving 3-SAT with adaptive genetic algorithms. In: Eiben, A. (ed.) Proceedings of the 4th IEEE Conference on Evolutionary Computation, Piscataway, pp. 81–86. IEEE Press, New Jersey, USA (1997)
22. Gottlieb, J., Voss, N.: Representations, fitness functions and genetic operators for the satisfiability problem. In: Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (eds.) AE 1997. LNCS, vol. 1363, pp. 53–68. Springer, Heidelberg (1998)
23. Gottlieb, J., Voss, N.: Adaptive fitness functions for the satisfiability problem. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) Parallel Problem Solving from Nature-PPSN VI. LNCS, vol. 1917, pp. 621–630. Springer, Heidelberg (2000)
24. Hoos, H., Stützle, T.: An online resource for research on SAT. In: Gent, P., Maaren, H., Walsh, T. (eds.) Proceedings of the SAT 2000, IOS Press, Amsterdam (2000)

Author Index

- Ayala-Rincón, Mauricio 177
- Barán, Benjamín 352
- Baran, Joachim 1
- Barringer, Howard 1
- Béchet, Denis 12
- Bedregal, Benjamín C. 26, 307
- Bensaid, Hicham 38
- Bernardi, Raffaella 53
- Bissell-Siders, Ryan 72
- Bolander, Thomas 83
- Brasoveanu, Adrian 101
- Broda, Sabine 120
- Caferra, Ricardo 38
- Callejas-Bedregal, Roberto 26
- Damas, Luís 120
- Dikovsky, Alexander 131
- Dimuro, G.P. 307
- Foret, Annie 12, 147
- Gabbay, Murdoch J. 162
- Galdino, André L. 177
- Gardner, Philippa 189
- Hansen, René Rydhof 83
- Hartmann, Sven 203
- Hirschowitz, André 218
- Isihara, Ariya 238
- Kauffman, Louis H. 248
- LaBean, Thomas H. 297
- Link, Sebastian 203
- Lomonaco Jr., Samuel J. 248
- Maggesi, Marco 218
- Mathijssen, Aad 162
- Moortgat, Michael 53, 264
- Muñoz, César 177
- Oliva, Paulo 285
- Peltier, Nicolas 38
- Reif, John H. 297
- Reiser, R.H.S. 307
- Rutten, Jan 322
- Santiago, R.H.N. 307
- Santos, Hélida S. 26
- Silva, Alexandra 322
- Solin, Kim 337
- Venema, Yde 351
- Villagra, Marcos 352
- Zarfaty, Uri 189