

Lecture Notes in Artificial Intelligence 2014

Subseries of Lecture Notes in Computer Science

Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer

Berlin

Heidelberg

New York

Barcelona

Hong Kong

London

Milan

Paris

Singapore

Tokyo

Michael Moortgat (Ed.)

Logical Aspects of Computational Linguistics

Third International Conference, LACL'98
Grenoble, France, December 14-16, 1998
Selected Papers



Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editor

Michael Moortgat
Utrecht Institute of Linguistics OTS
Trans 10, 3512 JK Utrecht, The Netherlands
E-mail: Michael.Moortgat@let.uu.nl

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Logical aspects of computational linguistics : third international
conference ; selected papers / LACL '98, Grenoble, France, December 14 - 16,
1998. Michael Moortgat (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ;
Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2144 : Lecture notes in
artificial intelligence)
ISBN 3-540-42251-X

CR Subject Classification (1998): I.2, F.4.1

ISBN 3-540-42251-X Springer-Verlag 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-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

<http://www.springer.de>

© Springer-Verlag Berlin Heidelberg 2001
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna
Printed on acid-free paper SPIN: 10782159 06/3142 5 4 3 2 1 0

Preface

The conference series Logical Aspects of Computational Linguistics (LACL) aims at providing a forum for the presentation and discussion of current research in all the formal and logical aspects of computational linguistics. The LACL initiative started with a workshop held in Nancy (France) in 1995. Selected papers from this event have appeared as a special issue of the *Journal of Logic Language and Information*, Volume 7(4), 1998. In 1996, LACL shifted to the format of an international conference. LACL'96 and '97 were both held in Nancy (France). The proceedings appeared as volumes 1328 and 1582 of the Springer Lecture Notes in Artificial Intelligence.

This volume contains selected papers of the third international conference on Logical Aspects of Computational Linguistics (LACL'98), held in Grenoble, France, from December 14 to 16, 1998. The conference was organized by the University Pierre Mendès-France (Grenoble 2) together with LORIA (Laboratoire Lorrain d'Informatique et Applications, Nancy). On the basis of 33 submitted 4-page abstracts, the Program Committee selected 19 contributions for presentation. In addition to the selected papers, the program featured three invited talks, by Maarten de Rijke (ILLC, Amsterdam), Makoto Kanazawa (Chiba University, Japan), and Fernando Pereira (AT&T Labs). After the conference, the contributors were invited to submit a full paper for the conference proceedings.

I thank the members of the Program Committee for their conscientious work in refereeing the submitted abstracts and the full papers, and the authors of the invited papers and the regular presentations for their stimulating contributions to the conference. Special thanks go to the Organizing Committee, in particular Alain Lecomte, whose efficiency contributed greatly to the success of LACL'98.

April 2001

Michael Moortgat

Organization

Program Committee

Chair Michael Moortgat (OTS Utrecht)
Johan van Benthem (ILLC Amsterdam)
Gosse Bouma (Groningen)
Vijay Shanker (Delaware)
Erhard Hinrichs (Tübingen)
Mary Dalrymple (Xerox, Palo Alto)
Ruy de Queiroz (Recife)
François Lamarche (LORIA Nancy)
Christian Retoré (IRISA Rennes)
Uwe Reyle (IMS Stuttgart)

Organizing Committee

Chair Alain Lecomte (Université Grenoble 2 & Project Calligramme, LORIA)
Denis Vernant (Groupe de Recherches Philosophie, Langage et Cognition)
Jean-Michel Adam (Département Informatique & Mathématiques)

The local Organizing Committee thanks Catherine Finkel (Université Pierre Mendès-France), Claude Jeannin and Amélie de Paoli (UFR Sciences de l'Homme et de la Société) for their logistic support during the workshop, and François Lamarche (LORIA) and Christian Retoré (IRISA) for passing on their experience gained during the organization of previous LACL conferences.

Sponsoring Institutions

La Ville de Grenoble
France Télécom – CNET
Xerox Research Center Europe
Université Pierre Mendès-France Grenoble 2
INRIA Rhône-Alpes

Table of Contents

Invited Paper

Deductions with Meaning	1
<i>Christof Monz, Maarten de Rijke (ILLC, Amsterdam)</i>	

Contributed Papers

Computational Solutions for Structural Constraints (Learning Structural Permissions in Categorical Grammar)	11
<i>Marcelo Finger (University of São Paulo)</i>	
Hypothetical Reasoning and Basic Non-constituent Coordination in Type-Logical Grammar	31
<i>Nissim Francez (Technion-IIT, Haifa)</i>	
Computational and Structural Aspects of Openly Specified Type Hierarchies	48
<i>Stephen J. Hegner (Umeå University)</i>	
Anaphora and Quantification in Categorical Grammar	70
<i>Gerhard Jäger (ZAS, Berlin)</i>	
An LTAG Perspective on Categorical Inference	90
<i>Aravind K. Joshi, Seth Kulick, Natasha Kurtonina (University of Pennsylvania)</i>	
Dominance Constraints: Algorithms and Complexity	106
<i>Alexander Koller, Joachim Niehren (Universität des Saarlandes), Ralf Treinen (Université Paris-Sud)</i>	
Strict Compositionality and Literal Movement Grammars	126
<i>Marcus Kracht (Freie Universität Berlin)</i>	
Categorical Minimalism	143
<i>Alain Lecomte (Université Pierre Mendès France, Grenoble 2 & LORIA, Nancy)</i>	
Sequential Construction of Logical Forms	159
<i>Wilfried Meyer-Viol (King's College, London)</i>	
Derivational Minimalism Is Mildly Context-Sensitive	179
<i>Jens Michaelis (Universität Potsdam)</i>	

Dominance Constraints in Context Unification	199
<i>Joachim Niehren, Alexander Koller (Universität des Saarlandes)</i>	
A Revision System of Circular Objects and Its Applications to Dynamic Semantics of Dialogues	219
<i>Norihiko Ogata (Osaka University)</i>	
Lexicalized Proof-Nets and TAGs	230
<i>Sylvain Pogodalla (XRCE, Meylan)</i>	
Lambek Calculus Proofs and Tree Automata	251
<i>Hans-Joerg Tiede (Illinois Wesleyan University)</i>	
Grammars with Composite Storages	266
<i>Christian Wartena (Universität Potsdam)</i>	
Author Index	287

Deductions with Meaning

Christof Monz and Maarten de Rijke

Institute for Logic, Language and Computation (ILLC)
University of Amsterdam, Plantage Muidergracht 24,
1018 TV Amsterdam, the Netherlands.
{christof, mdr}@wins.uva.nl

Abstract. In this paper, we consider some of the problems that arise if automated reasoning methods are applied to natural language semantics. It turns out that the problem of ambiguity has a strong impact on the feasibility of any theorem prover for computational semantics. We briefly investigate the different aspects of ambiguity and review some of the solutions that have been proposed to tackle this problem.

1 Introduction

One of the concluding slogans of the FraCaS project on Frameworks for Computational Semantics is that ‘[t]here can be no semantics without logic’ [11]. We take this to mean that formalisms for semantic representation should be developed hand-in-hand with inference methods for performing reasoning tasks with representations and algorithms for representation construction.

Clearly, to be usable in the first place, representation formalisms need to come equipped with construction methods, and this explains the need for algorithmic tools. But what about the need for inference methods? At least three types of reasons can be identified. For *cognitive* purposes one may want to test the truth conditions of a representation against (a model of) speakers’ intuitions—this amounts to a model checking or theorem proving task. Also, the whole issue of what it is to understand a discourse may be phrased as a model generation task. *Computationally*, we need various reasoning tasks and AI-heuristics to help resolve quantifier scope ambiguity, or to resolve anaphoric relations in information extraction and natural language queries. And last, but not least, the very *construction* of semantic representations may require inference tools to be used in checking for consistency and informativity. At the end of the day, the main purpose of a semantic representation is that we can *do* something with it, both algorithmically and in terms of inference tasks.

Now, the present times are exciting ones for anyone with an interest in inference for natural language semantics. On the one hand, there is work in semantics that has little or no attention for inferential aspects. This is certainly the case for a lot of work in dynamic semantics and underspecified representation, and in the recent *Handbook of Logic and Language* [6] inferential methods for semantic representations are largely absent, despite the fact that a substantial part of the book is devoted to representational matters.

At the same time, there is a growing body of work aimed at developing inference methods and tools for natural language semantics, fed by a growing realization that these are ‘the heart of the enterprise’ [7, page viii]. This is manifested not only by various research initiatives (see below), but also by the fact that a number of textbooks and monographs on natural language semantics and its inferential and algorithmic aspects are in preparation [7,11], and by a recent initiative to set up a special interest group on Computational Semantics (see <http://www.coli.uni-sb.de/~patrick/SIGICS.html> for details).

In this note we survey some of the ongoing work on inference and natural language semantics; we identify commonalities, as well as possibilities and the main logical challenges we are confronted with in the field.

2 Putting Semantics to Work

2.1 Lines of Attack

It has often been claimed that classical reasoning based on first-order logic (FOL) is not appropriate as an inference method for natural language semantics. We are pragmatic in this matter: try to stick to existing formats and tools and see how far they get you, and only if they fail, one should develop novel formats and tools. Traditional inference tools (such as theorem provers and model builders) are reaching new levels of sophistication, and they are now widely and easily available. Blackburn and Bos [7] show that the ‘conservative’ strategy of using first-order tools can actually achieve a lot. In particular, they use first-order theorem proving techniques for implementing van der Sandt’s approach to presupposition. We refer the reader to the DORIS system, which is accessible on the internet at <http://www.coli.uni-sb.de/~bos/atp/doris.html>.

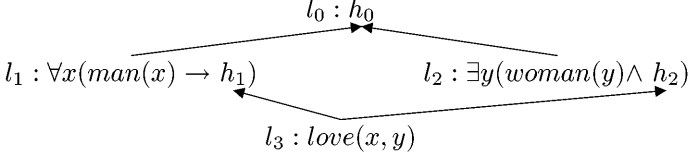
Although one may want to stick to first-order based tools as much as possible, for reasons of efficiency, or simply to get ‘natural representations’ it may pay to move away from the traditional first-order realm. Such a move may be particularly appropriate in two of the areas that currently pose the biggest challenges for computational semantics: *ambiguity* and *dynamics* [11, Chapter 8]. Let us consider some samples of deductive approaches in each of these two areas.

Reasoning with Quantifier Ambiguity. While the problem of ambiguity and underspecification has recently enjoyed a considerable increase in attention from computational linguists and computer scientists, the focus has mostly been on semantic aspects, and ‘reasoning with ambiguous sentences is still in its infancy’ [13]. Lexical ambiguities can be represented pretty straightforwardly by putting the different readings into a disjunction. It is also possible to express quantificational ambiguities by a disjunction, but quite often this involves much more structure than in the case of lexical ambiguities, because quantificational ambiguities are not tied to a particular atomic expression. For instance, the only way to represent the ambiguity of (1.a) in a disjunctive manner is (2).

- (1) Every man loves a woman.
 (2) $\forall x (man(x) \rightarrow \exists y (woman(y) \wedge love(x, y)))$
 $\vee \exists y (woman(y) \wedge \forall x (man(x) \rightarrow love(x, y)))$

Obviously, there seems to be some redundancy, because some subparts appear twice. Underspecified approaches such as the Core language Engine (CLE, [3]) or Underspecified Discourse Representation Theory (UDRT, [35]) allow us to represent quantifier ambiguities in a non-redundant way. The corresponding underspecified representation for (1) is given in (3).¹

(3)



This concise representation of the possible readings should allow us to avoid the state explosion problem. For representing the semantics of a natural language sentence this can be seen immediately, but to which extent theorem proving profits from underspecified representations is not easily determined. Up to now there is no proof theory which can directly work with underspecified representation. All of the approaches we are aware of rely, to some extent, on disambiguation. That is: first disambiguate an underspecified representation and then apply the rules of your proof theory. Once disambiguation has been carried out, this amounts, more or less, to classical proof theory, see, for instance, [17].

In [29] we have proposed a tableau calculus that interleaves disambiguation steps with deduction steps so that the advantages of an underspecified representation can, at least partially, be retained.

In addition, it is sometimes not necessary to compute all disambiguations, because there exists a strongest (or weakest) disambiguation. If there exists such a strongest (or weakest) disambiguation it suffices to verify (or falsify) this one, because it entails (or is entailed by) all other disambiguations. E.g., (4) has six readings which are listed in (5). For each reading we put the order of the quantifiers and negation sign as a shorthand in front of it.

(4) Every boy didn't see a movie

- (5) $(\forall\exists\neg) \forall x(boy(x) \rightarrow \exists y(movie(y) \wedge \neg see(x, y)))$
 $(\forall\neg\exists) \forall x(boy(x) \rightarrow \neg\exists y(movie(y) \wedge see(x, y)))$
 $(\neg\forall\exists) \neg\forall x(boy(x) \rightarrow \exists y(movie(y) \wedge see(x, y)))$
 $(\exists\forall\neg) \exists y(movie(y) \wedge \forall x(boy(x) \rightarrow \neg see(x, y)))$
 $(\exists\neg\forall) \exists y(movie(y) \wedge \neg\forall x(boy(x) \rightarrow see(x, y)))$
 $(\neg\exists\forall) \neg\exists y(movie(y) \wedge \forall x(boy(x) \rightarrow see(x, y)))$

¹ Actually, the underspecified representation in (3) differs slightly from the way underspecified representations are defined in [35], where the holes are not explicitly mentioned. Our representation is a bit closer to [9], but the differences between the frameworks are mainly notational.

In (6), we give the corresponding entailment graph which has as its elements the readings in (5). Two readings φ and ψ are connected by \Rightarrow if $\varphi \models \psi$.

$$(6) \quad \begin{array}{ccc} (\exists\forall\neg) & (\forall\neg\exists) & (\neg\forall\exists) (\exists\neg\forall) \\ \Downarrow & & \Downarrow \\ (\forall\exists\neg) & & (\neg\exists\forall) \end{array}$$

Unfortunately, this graph is not very dense. There are only two pairs of readings that stand in the entailment relation. Nevertheless, it allows for some improvement of the calculus, as it allows us to filter out some readings. $(\exists\forall\neg)$ and $(\forall\exists\neg)$ are two of the readings of (4), where $(\exists\forall\neg)$ entails $(\forall\exists\neg)$. On the other hand, if we are able to derive a contradiction for $(\forall\exists\neg)$, then we know that $(\exists\forall\neg)$ is contradictory, too. In [29] we have shown how the subset of readings which is sufficient can be identified.

But there is more to reasoning with quantificational ambiguity than just developing a calculus for it. In the presence of multiple readings of premises and conclusions, fundamental logical notions such as *entailment* receive new dimensions. Should *all* possible readings of the conclusion follow (in the traditional sense) from *all* possible readings of the premises for the ambiguous conclusion to qualify as a consequence of an ambiguous premise? Basic research in this direction has been carried out by a number of people [12,36,17,20]. Ultimately, the aim here is to obtain insights into the development and implementation of theorem provers for underspecified representations.

Reasoning with Pronoun Ambiguity. A number of calculi have been proposed for reasoning with dynamic semantics. [21,37,40] present natural deduction style calculi for Discourse Representation Theory, and [42] presents a tableau calculus. In the area of Dynamic Predicate Logic (DPL, [19]) and its many variations, [43] presents some ground-tableau calculi and [15] a sequent calculus. All of these approaches presuppose that pronouns are already resolved to some antecedent. Therefore, the problem of pronoun ambiguity does not arise within the calculus but the construction algorithm of the semantic representations. In order to employ the aforementioned calculi it is necessary that the semantic representation is disambiguated, but this might result in a huge number of readings, where the advantage of underspecified representation is lost. Again, it seems reasonable to interleave disambiguation and deduction steps, where disambiguation is only carried out if this is demanded by the deduction method.

The *resolution method* [39] has become quite popular in automated theorem proving, because it is very efficient and it is easily augmentable by lots of strategies which restrict the search space, see e.g., [23]. On the other hand, the resolution method has the disadvantage of presupposing that its input has to be in *clause form*, where clause form is the same as CNF but a disjunction is displayed as a set of literals (the *clause*) and the conjunction of disjunctions is a set of clauses. Probably the most attractive feature of resolution is that it has only one single inference rule, the resolution rule.

Applying the classical resolution method to a dynamic semantics introduces a problem: transforming formulas to clause form causes a loss of structural information. Therefore, it is sometimes impossible to distinguish between variables that can serve as antecedents for a pronoun and variables that cannot. [27,28] provide a resolution calculus that uses *labels* to encode the information about accessible variables. Each pronoun is annotated with a label that indicates the set of accessible antecedents.

There is a further problem with resolution calculus as it was presented in [27,28] is that it requires backtracking in order to be complete. Unfortunately, backtracking is hard to implement efficiently and it spoils some of the appeal of preferring resolution over tableau methods.

A tableau calculus for pronoun ambiguity has been introduced in [30]. This tableau calculus has a number of advantages over a resolution-based approach to pronoun resolution, as mentioned above. First of all, it is possible to interleave the computation of accessible variables with deduction, since preservation of structure is guaranteed in our signed tableau method. This is not possible in resolution, because it is assumed that the input is in conjunctive normal form, which destroys all structural information needed for pronoun binding. There, accessible antecedents can only be computed by a preprocessing step, cf. [27,28].

But the major advantage is that no backtracking is needed if the choice of an antecedent for a pronoun does not allow us to close all open branches; we simply apply pronoun resolution again, choosing a different antecedent.

2.2 Lessons Learned

The brief sketches of recent work on inference and natural language semantics given above show a number of things. First, all traditional computational reasoning tasks (theorem proving, model checking, model generation) are needed, but often in novel settings that work on more complex data structures. Dealing with ambiguity is one of the most difficult tasks for theorem provers, and we have seen in the previous section how we can tackle this problem. On the other hand, so far we have only looked at theorem proving for quantifier and pronoun ambiguity, separately; but what kind of problems arise if one tries to devise a theorem prover for a language containing both kinds of ambiguity? We will have a closer look at this later on.

Second, there are novel logical concerns both at a fundamental and at an architectural level. The former is illustrated by the proliferation of notions of entailment and by the need for incremental, structure preserving proof procedures. As to the latter, to move forward we need to develop methods for integrating specialized inference engines, possibly operating on different kinds of information, with other computational tools such as statistical packages, parsers, and various interfaces. We propose to use combinations of small specialized modules rather than large baroque systems. Of course, similar strategies in design and architecture have gained considerable attention in both computer science [8], and in other areas of applied logic and automated reasoning [10].

Combining Ambiguities. What happens if we combine both kinds of ambiguity and try to reason efficiently with formulas that contain ambiguous quantifier scopings *and* unresolved pronouns? Especially, which of the proof strategies that we used for dealing with the respective ambiguities can be adopted, and which ones raise problems?

When combining different kinds of ambiguity, ambiguities do not simply multiply, but they also interfere. Below, a short example is given, where (7) and its two readings in (8), an instance of quantifier ambiguity, is followed by (9), which contains an unresolved pronoun.

- (7) Every man loves a woman.
- (8) a. $\forall x (man(x) \rightarrow \exists y (woman(y) \wedge love(x, y)))$
 b. $\exists y (woman(y) \wedge \forall x (man(x) \rightarrow love(x, y)))$
- (9) But she is already married.

Here, (9) allows us to resolve the quantifier ambiguity of (7). Therefore, an appropriate calculus has to account for this. (9) filters out (8.a), because it does not provide an antecedent for the pronoun *she* in (9). This is easily seen, as (7) was uttered in the empty context, and (8.a) does not provide any antecedents. This implies that (8.a) cannot be a possible reading.

The preceding discussion so far hints at another problem that occurs if we try to reason in a combined framework. Considering only quantifier ambiguity, it was possible to neglect a reading φ if it entailed another reading ψ . Is this still possible if there are pronouns occurring in the proof which remain to be resolved? Reconsidering (7), the reading (8.b) entails (8.a), and it is sufficient to use only (8.a) in the proof. But if (7) is followed by (9), then (8.a) does not provide any antecedents for the pronoun in (9) and the pronoun remains unresolved. In fact, according to the discussion above, (8.a) would be filtered out, just because it cannot provide any antecedent; but then, no reading is left. (8.b) is filtered out, because it is stronger than (8.a), and (8.a) is filtered out for the reasons just given.

An obvious way out is to prefer weaker readings over stronger ones without throwing the stronger reading away. Only if the weaker reading does not cause any unresolvedness of pronouns, one can fully dispense with the stronger reading. For a longer discussion of this problem and some ways to solve this, the reader is referred to [33].

Incrementality. Implementations in computational semantics that employ theorem provers normally state the inference tasks in a non-incremental way. For instance, DORIS filters out those readings of a natural language discourse that do not obey local informativity or local consistency constraints. In this process of filtering out readings, the system is often faced with very similar reasoning tasks involving very similar sets of premises and conclusions. In DORIS, these tasks are treated independently of each other, and every inference task is started from scratch. The set of formulas which are treated multiple times grows with the length of the discourse. Of course, this redundancy significantly decreases

the efficiency of the implementation, and it will prevent the system from scaling up.

[31,32] introduce a way of stating these inference tasks such that redundant applications of inference rules can be avoided. This is accomplished by taking context and the way contextual information is threaded through a discourse explicitly into account. The approach in [31,32] is based on formal theories of context, see, e.g., [4,5,25].

3 Further Directions and Challenges

The findings of the previous sections are supported by a number of further and novel developments in more applied areas adjacent to natural language semantics. We will restrict ourselves to three examples.

First, in syntactic analysis, partial or underspecified approaches to parsing are becoming increasingly popular [1]. Just like underspecified representations in semantics, a partial parse fully processes certain phrases, but leaves some ambiguities such as modifier attachment underspecified. Given this similarity, it is natural to ask whether underspecified semantics can somehow be combined with partial parsing. An ongoing project at ILLC studies to which extent one can, for instance, use semantic information into account to resolve syntactic ambiguities; see <http://www.illc.uva.nl/~mdr/Projects/Derive/> for details. Note that combinations of underspecified representation and packed syntactic trees (parse forests) have been considered before [41,14], but no methods for using semantic information to resolve syntactic ambiguities are reported there.

Second, assuming that underspecified representations can usefully be combined with partial parsing, we may be able to improve methods in Information Extraction (IE). Common approaches to IE suffer from the fact that they either give only a very shallow analysis of text documents, as in approaches using word vectors, or that they are domain dependent, as in the case of template filling. More general techniques using some kind of logical representation could circumvent these disadvantages. Now, IE techniques provide the right data structures, but to access the information one needs the right retrieval algorithms. Logic-based Information Retrieval (IR) has been around, at least theoretically, since the mid 1980's [38]. An ongoing project at ILLC investigates to which extent underspecified reasoning and representation can be used for IR; again, see <http://www.illc.uva.nl/~mdr/Projects/Derive/> for details. We do not believe that these techniques can compete with IR methods for very large data collections, where logic-based techniques seem to be intractable, but we are confident about substantial quality improvements for smaller domains.

In this context, it seems interesting to investigate to which extent Description Logics can be employed to represent the content of a document. [24] consider a fragment of Montague Semantics ([26]) that can be expressed in Description Logics. Formulas belonging to this fragment have to be quantifier-free, meaning that they do not contain any lambda abstractions. For instance, (10.b), which

is the semantic representation of (10.a) belongs to the fragment, but (11.b), representing (11.a), does not.

(10)a. Mary read a book.

b. (Mary read (some book))

(11)a. Mary read a book that John bought.

b. (Mary read (some ($\lambda x (x \text{ book} \wedge (\text{John} (\text{bought } x))))))))$

For this quantifier-free fragment, [24] provides an inference procedure which decides satisfiability in polynomial time. More generally, Description Logics are concerned representations and inference algorithms for fragments of first- and higher-order logics in which quantification is of a restricted, or *guarded* nature; see, for instance, [18,22] for further uses of Description Logics in computational semantics. One of the important advantages of using Description Logics is that very efficient inference tools are available, such as DLP [34].

Finally, and coming from a completely different direction, there is work on the use of dynamic semantics to explain the meaning of programs in hybrid programming languages such as *Alma-0* [2] that combine the imperative and declarative programming paradigms. [16] shows how dynamic predicate logic provides an adequate semantics for a non-trivial fragment of *Alma-0*, and how inference tools for dynamic predicate logic become verification tools for the hybrid programming language.

4 Conclusions

In this note we have identified some of the main concerns of doing inference for natural language semantics. One of the most difficult tasks in this context is the problem of reasoning with ambiguity. We have seen that it is possible to devise calculi which can deal with a particular kind of ambiguity, but that things get much more complicated if one tries to devise a calculus which can deal with different kinds of ambiguity. We have illustrated these concerns by means of samples from ongoing research initiatives, and, in addition, we have listed what we take to be some of the main challenges and most promising research directions in the area.

Acknowledgments. Christof Monz was supported by the Physical Sciences Council with financial support from the Netherlands Organization for Scientific Research (NWO), project 612-13-001. Maarten de Rijke was supported by the Spinoza Project ‘Logic in Action’ at ILLC, the University of Amsterdam.

References

1. Abney, S. Tagging and partial parsing. In *Corpus-Based Methods in Language and Speech*. Kluwer Academic Publishers, 1996.
2. Apt, K. R., J. Brunekreef, V. Partington, and A. Schaerf. *Alma-0: An imperative language that supports declarative programming*. *ACM Toplas*, 1998. In press.

3. Alshawi, H., editor. *The Core Language Engine*. MIT Press, Cambridge, Massachusetts, 1992.
4. Attardi, G., and M. Simi. Building proofs in context. In *Proc. of Meta '94*, LNCS 883, pages 410–424. Springer, 1994.
5. Attardi, G., and M. Simi. Proofs in context. In Doyle, J. and P. Torasso, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Fourth International Conference*. Morgan Kaufmann, 1994.
6. Benthem, J. van, and A. ter Meulen, editors. *Handbook of Logic and Language*. North-Holland, 1997.
7. Blackburn, P., and J. Bos. *Representation and Inference for Natural Language*. Available from <http://www.coli.uni-sb.de/~bos/comsem/>, 1998.
8. Booch, G. *Object-Oriented Design*. Benjamin/Cummings, 1991.
9. Bos, J. Predicate logic unplugged. In Dekker, P., and M. Stokhof, editors, *Proceedings of the 10th Amsterdam Colloquium*, pages 133–142. ILLC, University of Amsterdam, 1996.
10. Baader, F., and K. Schulz, editors. *Frontiers of Combining Systems*, Applied Logic Series. Kluwer Academic Publishers, 1996.
11. Crouch, D., C. Fox, J. van Genabith, and S. Glasbey, editors. *The FraCaS Book*. Unpublished manuscript, 1998.
12. Deemter, K. van. Towards a logic of ambiguous expressions. In van Deemter and Peters [13].
13. Deemter, K. van, and S. Peters, editors. *Semantic Ambiguity and Underspecification*. CSLI Publications, 1996.
14. Dörre, J. Efficient construction of underspecified semantics under massive ambiguity. In *Proc. of the 35th Annual Meeting of the ACL*, 1997.
15. Eijck, J. van. Axiomatizing dynamic logics for anaphora, 1998. CWI, Amsterdam.
16. Eijck, J. van. Programming with dynamic logic. Manuscript, CWI, Amsterdam, 1998.
17. Eijck, J. van, and J. Jaspars. Ambiguity and reasoning. Technical Report CS-R9616, CWI, Amsterdam, 1996.
18. Franconi, E. A treatment of plurals and plural quantifications based on a theory of collections. *Minds and Machines*, 3(4):453–474, 1993.
19. Groenendijk, J., and M. Stokhof. Dynamic Predicate Logic. *Linguistics and Philosophy*, 14:39–100, 1991.
20. Jaspars, J. Minimal logics for reasoning with ambiguous expressions. CLAUS-report 94, Universität des Saarlandes, 1997.
21. Kamp, H., and U. Reyle. A calculus for first order Discourse Representation Structures. *Journal of Logic, Language and Information*, 5:297–348, 1996.
22. Küssner, U. Description logic unplugged. In Franconi, E., et al., editors, *Proceedings of the 1998 International Description Logics Workshop (DL'98)*, 1998.
23. Loveland, D. W. *Automated Theorem Proving: A Logical Bases*. North-Holland, Amsterdam, 1978.
24. McAllester, D. A., and R. Givan. Natural language syntax and first-order inference. *Artificial Intelligence*, 56(1):1–20, 1992.
25. McCarthy, J., and S. Buvač. Formalizing context (expanded notes). In Aliseda, A., R. van Glabbeek, and D. Westerståhl, editors, *Computing Natural Language*, pages 13–50. Stanford University, 1997.
26. Montague, R. The proper treatment of quantification in ordinary english. In Hintikka, J., J. Moravcsik, and P. Suppes, editors, *Approaches to Natural Language. Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*. Reidel, Dordrecht, 1973.

27. Monz, C., and M. de Rijke. Labeled resolution for discourse semantics. In *Proc. LD'98*. Dept. of Computer Science, Univ. of Freiburg, 1998.
28. Monz, C., and M. de Rijke. A resolution calculus for dynamic semantics. In Dix, J., et al., editors, *Proc. JELIA'98*, LNAI 1489, pages 184–198. Springer, 1998.
29. Monz, C., and M. de Rijke. A tableaux calculus for ambiguous quantification. In Swart, H. de, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, TABLEAUX'98, LNAI 1397, pages 232–246. Springer, 1998.
30. Monz, C., and M. de Rijke. A tableau calculus for pronoun resolution. In Murray, N. V., editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, TABLEAUX'99, LNAI. Springer, 1999.
31. Monz, C. Computing presuppositions by contextual reasoning. In *Proceedings of the AAAI-99 Workshop on Reasoning in Context for AI Applications*, 1999.
32. Monz, C. Contextual inference in computational semantics. Submitted for publication, 1999.
33. Monz, C. Underspecified theorem proving with different kinds of ambiguity. Institute for Computational Linguistics, University of Stuttgart, 1999.
34. Patel-Schneider, P. DLP system description. In Franconi, E., et al., editor, *Proceedings of the 1998 International Description Logics Workshop (DL'98)*, pages 87–89, 1998.
35. Reyle, U. Dealing with ambiguities by underspecification: Construction, representation, and deduction. *Journal of Semantics*, 10(2):123–179, 1993.
36. Reyle, U. On reasoning with ambiguities. In *Proceedings of the 6th Meeting of the EACL*, pages 1–8, Dublin, 1995.
37. Reyle, U., and D. Gabbay. Direct deductive computation on Discourse Representation Structures. *Linguistics and Philosophy*, 17(4):343–390, 1994.
38. Rijsbergen, K. van. A non-classical logic for information retrieval. *The Computer Journal*, 29:481–485, 1986.
39. Robinson, J. A. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.
40. Saurer, W. A natural deduction system for Discourse Representation Theory. *Journal of Philosophical Logic*, 22:249–302, 1990.
41. Schiehlen, M. Semantic construction from parse forests. In *Proc. COLING'96*, 1996.
42. Sedogbo, C., and M. Eytan. A tableau calculus for DRT. *Logique et Analyse*, 31:379–402, 1986.
43. Veltman, F. Dynamic predicate logic, 1997. ILLC, Amsterdam.

Computational Solutions for Structural Constraints

Learning Structural Permissions in Categorical Grammar

Marcelo Finger^{*}

Department of Computer Science, University of São Paulo, Rua do Matão 1010,
05508-900, São Paulo, Brazil, mfinger@ime.usp.br

Abstract. The need to solve structural constraints arises when we investigate computational solutions to the question: “*in which logics is a given formula deducible?*” This question is posed when one wants to learn the structural permissions for a Categorical Grammar deduction.

This paper is part of a project started at [10], to deal with the question above. Here, we focus on *structural constraints*, a form of Structurally-Free Theorem Proving, that deal with an unknown transformation X which, when applied to a given set of components $P_1 \dots P_n$, generates a desired structure Q . The constraint is treated in the framework of the combinator calculus as $XP_1 \dots P_n \rightarrow Q$, where the transformation X is a combinator, the components P_i and Q are terms, and \rightarrow reads “reduces to”.

We show that in the usual combinator system not all admissible constraints have a solution; in particular, we show that a structural constraint that *represents right-associativity* cannot be solved in it nor in any consistent extension of it. To solve this problem, we introduce the notion of a *restricted combinator system*, which can be consistently extended with *complex combinators* to represent right-associativity. Finally, we show that solutions for admissible structural constraints always exist and can be efficiently computed in such extension.

1 Introduction

This paper introduces the concept of *structural constraints* and studies how to efficiently solve them. The problem of solving structural constraints arises when one tries to automatically “learn” the structural permissions that can be used in inferences in Categorical Grammar. Lest we raise false expectations, we hasten to point out that structural constraints are just one step in the learning of structural permissions, motivated by our initial work in structurally-free theorem proving [10].

Linguistic inference in Categorical Grammar is normally performed in one of a variety of substructural logics. E.g. in [14] the repertoire included four logics (**L**, **NL**, **NLP** and **LP**) parameterized by whether the structural rules of (right-)

^{*} Partly supported by the Brazilian Research Council (CNPq), grant PQ 300597/95-5.

associativity and commutativity were allowed or not. We want here to address the problem of automatically learning from an example which structural rules (and hence which logic) we are dealing with.

More specifically, we address the following question: given an antecedent and a desirable consequent, can we *automatically* determine in which substructural logics the latter is inferable from the former? This activity has been termed *Structurally-Free Theorem Proving* (SFTP).

SFTP is a variation of traditional theorem proving introduced in [10], which emerges from the work on Structurally Free Logics (SFL) by Dunn and Meyer [9, 7]. SFL is a family of logics free from any structural presupposition — whence its name. SFL includes combinators (*i.e.* λ -terms without free variable) as special kind of atoms. Combinators represent faithfully the usual structural inference rules (contraction, commutativity, etc.) In SFL, these rules are replaced by *combinator rules* such that each SFL-deducible sequent contains some combinators that indicate the structural rules needed for its deduction. SFL is thus strongly related to the family of *propositional substructural logics* [8]; in such a family, one logic differs from the others by the *structural rules* it allows in the deductions, *i.e.* by the rules through which the premises in a proof can be rearranged.

Possible applications of SFTP to computational linguistics include:

- Automated learning of structural permissions in the framework of Categorical Grammar. In particular, SFTP allows one to compute, from a positive example, the structural rules needed to make this example true, and where they should be applied.
- Bridging the gap between the deductive approach to Categorical Grammar of [14] and Steedman’s Combinatory Categorical Grammar [17], taking advantage of the presence of combinators as first class elements in SFTP.

Admittedly, SFTP techniques have to be perfected and refined before these goals can be fully achieved. In [10] it was shown that SFTP can be efficiently solved in a fragment containing connectives $\{\bullet, /\}$ if it can be solved for the $\{\bullet\}$ -fragment; the solution for the $\{\bullet\}$ -fragment was only hinted at.

This paper intends to provide a solid basis for SFTP by showing that it can be efficiently solved in the $\{\bullet\}$ -fragment, thus complementing the work of [10]. It turns out that solving SFTP in the $\{\bullet\}$ -fragment is equivalent to solving a *structural constraint*, that is, an expressions in the λ -calculus and combinator calculus [1] of the form

$$XP_1 \dots P_n \multimap Q$$

where X is the unknown *structural transformation*, $P_1 \dots P_n$ and Q are *pure terms*, *i.e.* terms built only with variables, and \multimap is β - or combinator-reduction. A structural constraint is *admissible* if each variable in Q occurs in some P_i .

For example, to know the structural rules needed to deduce

$$p_1 \bullet p_2 \vdash p_2 \bullet (p_1 \bullet p_1)$$

it suffices to find a combinator that solves the structural constraint

$$Xx_1x_2 \multimap x_2(x_1x_1).$$

The solution for this constraint, $X = W(BC(C(B)))$ as computed in Section 3.1, represents the structural rules for commutativity (C), left-associativity (B) and contraction (W). Moreover, it is possible from this solution to reconstruct a deduction for the initial sequent [10].

It turns out, however, that the path to find solutions to any admissible structural constraint is not a simple one. Some admissible structural constraints, do not have a solution in the combinator calculus (or, equivalently, in the λ -calculus), *e.g.*

$$Xx_2(x_1x_1) \not\rightarrow x_1x_2.$$

If we simply extend the combinator calculus to cope with such deficiency, it easily becomes inconsistent (Lemma 3).

The main result of the paper is showing that we can construct a consistent setting in which *all admissible structural constraints have efficiently computable solutions*, so SFTP can be solved in the \bullet -fragment.

By overcoming the inconsistency problems (see below), this work paves the way to the extension of SFTP techniques of [10] to the more usual $\{\bullet, /, \backslash\}$ -fragment of Categorical Grammar.

We proceed as follows. After introducing the combinator calculus and SFL in Section 2, we can formally define structural constraints in Section 3 and show that the class of *simple constraints* have simple, efficient computational solutions. We then present an algorithm (Algorithm 1) that solves simple structural constraints.

However, Section 4 shows that the full class of *complex structural constraints* does *not* have a solution in the standard combinator system nor in any extension of it. To solve this problem, we introduce in 4.1 the notion of a *restricted combinator system* and show that the problems of standard combinators are absent from it. Finally, we introduce in 4.2 *complex combinators* and show how all admissible complex structural constraints has a solution using complex combinators in 4.4; this solution is shown to be computable in time $O(N^2)$.

2 Background

2.1 Combinator Systems

Given a set \mathbb{C} of basic combinators, a *combinator system* is a pair $(\mathcal{C}_c, \rightarrow_c)$, where the set \mathbb{C} is the *basis* of the system, \mathcal{C}_c are the *combinator terms based on* \mathbb{C} , and \rightarrow_c is the reduction relation associated to \mathbb{C} . We usually refer to \mathbb{C} as *the* combinator system. When the basis \mathbb{C} is clear from the context, the system is represented just by $(\mathcal{C}, \rightarrow)$; in this section, we use by default the set of *primitive* combinators $\mathbb{C}_0 = \{K, S, W, C, B, I\}$ described below.

To define the set of terms, consider $V = \{x, y, z, \dots\}$ a countable set of variables; both variables and combinators are *atomic* terms. The set \mathcal{C} of *combinator terms* is the smallest set that includes all atomic terms, and if P and Q are terms, then the application (PQ) is also a term.

A term P is *pure* if it contains only variables. A (*compound*) *combinator* is a term that contains no variables. $Var(Q)$ represents the set of free variables occurring in the term Q . We abbreviate $\mathbf{x} \equiv x_1 \dots x_n$; $\mathbf{P} \equiv P_1 \dots P_n$; and

$$Q\mathbf{P} \equiv QP_1 \dots P_n \equiv ((\dots((QP_1)P_2)\dots)P_n),$$

that is, application associates to the left. \mathbf{x} also represents the set $\{x_1, \dots, x_n\}$; context always separates both uses. $P(\mathbf{x})$ represents a term P where $Var(P) = \mathbf{x}$.

In $\mathbf{P} \equiv P_1 \dots P_n$ we always assume that P_1 is an atomic term; at the end of Section 4 we show how this restriction can be overcome.

Each combinator $X \in \mathbb{C}$ is associated to a *basic reduction rule* of the form $X\mathbf{P} \circ \twoheadrightarrow Q$ defining a binary relation between terms, where $X\mathbf{P}$ is called a *redex* (reducible expression). Each $X \in \mathbb{C}$ is assumed to be *functional*, *i.e.* if \mathbf{P} is pure, there is at most one Q such that $X\mathbf{P} \circ \twoheadrightarrow Q$. Figure 1 presents a basic reduction rule for combinators in $\mathbb{C}_0 = \{K, S, W, C, B, I\}$; the choice of letters is historical.

$BPQR \circ \twoheadrightarrow P(QR)$	$CPQR \circ \twoheadrightarrow PRQ$	$IP \circ \twoheadrightarrow P$
$WPQ \circ \twoheadrightarrow PQQ$	$SPQR \circ \twoheadrightarrow PR(QR)$	$KPQ \circ \twoheadrightarrow P$

Fig. 1. Reduction Rules for Primitive Combinator

For a given set \mathbb{C} the *reduction relation* \twoheadrightarrow is the smallest binary relation containing $\circ \twoheadrightarrow$ and closed under:

- reflexivity: $P \twoheadrightarrow P$;
- transitivity: $P \twoheadrightarrow Q$ and $Q \twoheadrightarrow R$ implies $P \twoheadrightarrow R$;
- congruence: if $P \twoheadrightarrow P'$ then $PQ \twoheadrightarrow P'Q$ and $QP \twoheadrightarrow QP'$.

The number of redexes in a term P is represented by $NRedex(P)$. P is in *normal form* (nf) if $NRedex(P) = 0$.

The (*weak*) *equality* on terms ($=_{w, \mathbb{C}}$, or only $=_w$ when \mathbb{C} is clear) is a binary relation obtained by the reflexive, symmetric, transitive and congruent closure of the basic reduction rules. A combinator system is *inconsistent* if $P =_w Q$ for all $P, Q \in \mathbb{C}$; otherwise it is *consistent*. A system is CR (*i.e.* it has the Church-Rosser property) if whenever $P \twoheadrightarrow Q$ and $P \twoheadrightarrow Q'$ there exists R such that $Q \twoheadrightarrow R$ and $Q' \twoheadrightarrow R$. It is well known that the \mathbb{C}_0 -system is CR and consistent [1].

Combinators can be defined as terms in the λ -calculus without free-variables, as shown in Figure 2. Also, any λ -term has an equivalent combinator term built up using only the combinators S and K [1]. In this work, we will treat combinators as primitive entities, not as derived λ -terms, as when they were originally proposed [16].

A combinator X is *definable* in a basis \mathbb{C} if there exists a $X' \in \mathbb{C}_c$ such that for all pure terms \mathbf{P}, Q , $X\mathbf{P} \twoheadrightarrow Q$ iff $X'\mathbf{P} \twoheadrightarrow Q$. A set of combinators is *independent*

$B \equiv \lambda xyz.x(yz)$	$C \equiv \lambda xyz.xzy$	$I \equiv \lambda x.x$
$W \equiv \lambda xy.xyy$	$S \equiv \lambda xyz.xz(yz)$	$K \equiv \lambda xy.x$

Fig. 2. λ -calculus combinators

if none of the combinators is definable in terms of the others. The primitive combinators in Figure 1 are not independent from each other. All combinators are definable in terms of S and K . Also, S is definable in terms of W , B and C as $B(BW)(BC(BB))$. K is independent from all others.

So, among the primitive combinators there are two independent, *combinatorially complete* sets, namely $\{K, S\}$ and $\{K, W, C, B\}$; that is, all combinators can be defined in terms of them. The combinator I is definable in both sets (as SKK and WK) but is very useful and is normally added to the bases; note that if we drop the combinator K , $\{W, C, B, I\}$ is also independent, though not complete.

2.2 Structurally Free Theorem Proving (SFTP) and Structurally Free Logics (SFL)

In the family of logics $SFL(\mathbb{C})$, combinators in \mathbb{C} are treated as special propositional atoms [7]. Atomic formulas are propositional letters or combinators (primitive or compound). For the purpose of this paper, we consider a propositional fragment containing only the connective \bullet , known as *multiplicative conjunction* or *product* or *fusion*. A formula is *pure* if it does not contain a combinator.

To be able to refrain from structural presupposition, sequent deductions in SFL have to deal with sequents of the form $\Gamma \vdash \varphi$ where φ is a formula and Γ is a *structure* defined as:

- every formula is a structure;
- if Γ and Δ are structures, so is (Γ, Δ) .

Structures associate to the left, so $\Gamma, \Delta, \Sigma \equiv ((\Gamma, \Delta), \Sigma)$. In this setting, the sequent rules for \bullet are:

$$\frac{\Gamma \vdash \varphi \quad \Delta \vdash \psi}{\Gamma, \Delta \vdash \varphi \bullet \psi} (\vdash \bullet) \quad \text{and} \quad \frac{\Gamma[\varphi, \psi] \vdash \chi}{\Gamma[\varphi \bullet \psi] \vdash \chi} (\bullet \vdash)$$

where $\Gamma[\varphi, \psi]$ indicates that the structure (φ, ψ) occurs in Γ and $\Gamma[\varphi \bullet \psi]$ is obtained by substituting $\varphi \bullet \psi$ for (φ, ψ) in Γ . Besides those connective rules, there is the *Axiom* rule stating that $\varphi \vdash \varphi$.

SFL has *combinator rules* instead of structural rules. All potential uses of structural rules in proofs are accounted by the introduction of a combinator. For example, the usual left-associativity rule (below on the left) is replaced in SFL by the B -introduction rule (below on the right):

$$\frac{\Gamma[\Theta, (\Delta, \Sigma)] \vdash \chi}{\Gamma[\Theta, \Delta, \Sigma] \vdash \chi} (l\text{-assoc}) \quad \frac{\Gamma[\Theta, (\Delta, \Sigma)] \vdash \chi}{\Gamma[B, \Theta, \Delta, \Sigma] \vdash \chi} (B \vdash)$$

Note that the structure inside $[]$ in the lower part of the $(B \vdash)$ -rule, can be seen as a redex that reduces into that in the upper part of the rule. Similarly, each primitive combinator is associated to a combinator rule and a structural rule. Combinator B accounts for left-associativity, C for commutativity, I for identity, W for contraction, K for thinning (or monotonicity) and S is also a type of contraction, called factoring. There can be rules associated to compound combinators too. Figure 3 in Section 4.2 illustrates several combinator rules and their associated structural rules.

Structurally-free theorem proving (SFTP) [10] takes as input an intuitionistic (*i.e.* combinator-free) sequent $\Gamma_1, \dots, \Gamma_n \vdash \varphi$, and tries find a combinator structure X such that $X, \Gamma_1, \dots, \Gamma_n \vdash \varphi$ is deducible in SFL. The combinators occurring in X encode the structural rules needed for the deduction of the original sequent, and so can tell us in which substructural logics it is deducible. In the \bullet -fragment, a sequent $X, \Gamma_1, \dots, \Gamma_n \vdash \varphi$ has an associated structural constraint $XP_{\Gamma_1} \dots P_{\Gamma_n} \multimap Q_\varphi$ obtained by deleting \bullet and (structure composition) $'$, (*i.e.* replacing \bullet and $'$ by application) and considering atomic propositions as variables.

However, not every deducible \bullet -sequent $\Gamma_1, \dots, \Gamma_n \vdash \varphi$ in intuitionistic logic has an equivalent $X, \Gamma_1, \dots, \Gamma_n \vdash \varphi$ in $SFL(\mathbb{C}_0)$. For example, the rule of right-associativity

$$\frac{\Theta, \Delta, \Sigma}{\Theta, (\Delta, \Sigma)} \text{ (right-associativity)}$$

has no corresponding combinator in \mathbb{C}_0 . Section 4 shows how a different combinator system completely represents all intuitionistic deductions and allows for the efficient computation of its associated combinator.

3 Structural Constraints

Let \mathbb{C} be a generic basis and let X be a metavariable standing for an unknown combinator. A *structural constraint* on X has the format:

$$\forall \mathbf{x} \mathbf{y} (X \mathbf{P}(\mathbf{x}) \multimap Q(\mathbf{y})) \quad (1)$$

The “ $\forall \mathbf{x} \mathbf{y}$ ” in (1) means that X must also be a solution for any substitution of $\mathbf{x} \mathbf{y}$. Constraint (1) is *admissible* if $Var(\mathbf{P}) = \mathbf{x} \supseteq \mathbf{y}$. It is a *simple constraint* if $\mathbf{P} \equiv \mathbf{x}$; otherwise it is a *complex* structural constraint.

Simple structural constraints have the format $X \mathbf{x} \multimap Q$ and are easily solvable.

Lemma 1. *A simple structural constraint has a solution iff it is admissible.*

For a proof, see [1, Corollary 2.1.24]. Section 3.1 below presents computational solutions. Note that there may be more than one solution for (1). In particular, any X' that reduces to X will also be a solution.

We start by concentrating on the solution for simple constraints and leave the solution for complex constraints to Section 4.

3.1 A Solution for Simple Constraints

We start by concentrating on the computation of solutions of simple structural constraints of the form $\mathbf{X}\mathbf{x} \rightarrow Q$, which is equivalent to finding a combinator corresponding to $\lambda\mathbf{x}.Q$. There are many algorithms in the literature that solve this problem:

- Curry’s initial formulation used the basis $\{\mathbf{K}, \mathbf{S}, \mathbf{I}\}$ [4], but the number of combinators in it was exponential with the size of \mathbf{x} .
- To reduce this inefficiency, Curry *et al* [4,5] introduced the combinators \mathbf{C} and \mathbf{B} in the algorithm, reducing the space complexity to be quadratic with the size of \mathbf{x} .
- More recent results by Turner, using so called non-standard bases, were able to reduce space to a linear dependency [15]. Also, the time complexity was reduced to $N \log N$.

However, none of these methods generates a solution that is extensible to the solution of complex structural constraint. So we introduce a solution of our own for the simple case and in Section 4 we will show how it generalises to the case of complex constraints.

3.2 List Combinators

We first introduce some abbreviations. The first is the *deferred combinator* presented originally in [4]. For $\mathbf{X} \in \mathbb{C}_0 - \{\mathbf{I}\}$ define $\mathbf{X}_{(i)}$ as:

$$\mathbf{X}_{(1)} \equiv \mathbf{X} \qquad \mathbf{X}_{(i+1)} \equiv \mathbf{B}\mathbf{X}_{(i)}$$

Intuitively, $\mathbf{X}_{(i)}$ defers to the right the application of \mathbf{X} by $i - 1$ positions¹. For example,

$$\begin{aligned} \mathbf{C}_{(1)}x_0x_1x_2x_3x_4 &\equiv \mathbf{C}x_0\underline{x_1x_2x_3x_4} &\rightarrow x_0\underline{x_2x_1x_3x_4} \\ \mathbf{C}_{(2)}x_0x_1x_2x_3x_4 &\equiv \mathbf{B}\mathbf{C}x_0x_1\underline{x_2x_3x_4} &\rightarrow x_0x_1\underline{x_3x_2x_4} \\ \mathbf{C}_{(3)}x_0x_1x_2x_3x_4 &\equiv \mathbf{B}(\mathbf{B}\mathbf{C})x_0x_1x_2\underline{x_3x_4} &\rightarrow x_0x_1x_2\underline{x_4x_3}, \text{etc.} \end{aligned}$$

We can now deal with the infinite basis $\mathbb{C}_0^* = \{\mathbf{X}_{(i)} \mid \mathbf{X} \in \mathbb{C}_0 - \{\mathbf{I}\}, i \geq 1\}$. An important property of the combinators in \mathbb{C}_0^* is that they are *head-invariant*.² \mathbf{X} is head-invariant if, for any redex of the form $\mathbf{X}x\mathbf{P}$ with $x \notin \text{Var}(\mathbf{P})$, there exists \mathbf{T} such that $\mathbf{X}x\mathbf{P} \rightarrow x\mathbf{T}$ and $x \notin \text{Var}(\mathbf{T})$.

The second abbreviation we introduce is that of a *list combinator*. The notation for a list is $\langle \mathbf{X}_1, \dots, \mathbf{X}_n \rangle$, with $\langle \rangle$ representing the empty list and $\mathbf{X}_j \in \mathbb{C}_0^*$. We define:

$$\begin{aligned} \langle \rangle &\equiv \mathbf{I} \\ \langle \mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n \rangle &\equiv \mathbf{X}_1 \langle \mathbf{X}_2, \dots, \mathbf{X}_n \rangle \end{aligned}$$

¹ [4] actually sets $\mathbf{X}_{(0)} \equiv \mathbf{X}$; our 1-based notation suits list combinators better.

² [4] calls these *regular* combinators; “head-invariant” was taken in similarity with the notion of *head normal form* in [1].

Note that $\langle X \rangle \equiv X \langle \rangle \equiv XI$. This definition is isomorphic to that of a list data structure, the first clause defining the empty list and the second list construction. The list combinator $\langle X_1, X_2, \dots, X_n \rangle$ actually abbreviates the term

$$X_1(X_2(\dots (X_n(I)) \dots))$$

We may think of the combinators in a list as sequentially applied left-to-right on a 1-based input string. For example

$$\begin{aligned} \langle W_{(3)}, C_{(2)}, B_{(3)} \rangle x_1 x_2 x_3 &\rightarrow \langle C_{(2)}, B_{(3)} \rangle x_1 x_2 x_3 x_3 \rightarrow \langle B_{(3)} \rangle x_1 x_3 x_2 x_3 \\ &\rightarrow \langle \rangle x_1 x_3 (x_2 x_3) \quad \rightarrow x_1 x_3 (x_2 x_3) \end{aligned}$$

so $\langle W_{(3)}, C_{(2)}, B_{(3)} \rangle$ defines S . Note that if X is head-invariant, $X_{(n)}$ and $\langle X_{(n+1)} \rangle$ are interdefinable. The solutions we will compute for simple constraints will always be list combinators, and that will guarantee its extensibility to complex constraints.

If X_{L_1} and X_{L_2} are two list combinators, their concatenation is represented by $X_{L_1} \cdot X_{L_2}$ and defined inductively as the usual list concatenation:

$$\langle \rangle \cdot X_L = X_L \quad \langle X_1, X_2, \dots, X_n \rangle \cdot X_L = X_1(\langle X_2, \dots, X_n \rangle \cdot X_L)$$

As a result, $\langle X_{n_1}, \dots, X_{n_p} \rangle \cdot \langle X_{m_1}, \dots, X_{m_q} \rangle = \langle X_{n_1}, \dots, X_{n_p}, X_{m_1}, \dots, X_{m_q} \rangle$ and \cdot is associative. We write $X_1 \cdot \langle X_2, \dots, X_n \rangle$ to represent $\langle X_1, X_2, \dots, X_n \rangle$.

Lemma 2 (Concatenation Lemma). *Let X_{L_1}, X_{L_2} be list combinators, and let P, Q be pure terms. Then $(X_{L_1} \cdot X_{L_2})P \rightarrow Q$ iff there exist pure O such that*

$$X_{L_1}P \rightarrow O \quad \text{and} \quad X_{L_2}O \rightarrow Q$$

Proof. By induction on the length of X_{L_1} . The base case is $X_{L_1} = \langle \rangle$, which holds trivially. For the inductive case, first consider (\Rightarrow) . Assume that $(X_1 \cdot X_{L_1}) \cdot X_{L_2}P \rightarrow Q$. Then there exists a T with $X_1 xP \rightarrow xT$, such that

$$(X_1 \cdot X_{L_1}) \cdot X_{L_2}P = X_1 \cdot (X_{L_1} \cdot X_{L_2})P = X_1(X_{L_1} \cdot X_{L_2})P \rightarrow (X_{L_1} \cdot X_{L_2})T$$

This last \rightarrow -step holds because X_1 is head-invariant. By the Church-Rosser property, $(X_{L_1} \cdot X_{L_2})T \rightarrow Q$ and then, by induction hypothesis, there exist pure term O such that

$$X_{L_1}T \rightarrow O \quad \text{and} \quad X_{L_2}O \rightarrow Q.$$

But because X_1 is head-invariant, $(X_1 \cdot X_{L_1})P \rightarrow X_{L_1}T \rightarrow O$.

For (\Leftarrow) , assume $X_1 \cdot X_{L_1}P \rightarrow O$ and $X_{L_2}O \rightarrow Q$. From X_1 being head-invariant it follows that $X_1 \cdot X_{L_1}P \rightarrow X_{L_1}T$ and, by Church Rosser, $X_{L_1}T \rightarrow O$. So applying the induction hypothesis we obtain

$$(X_{L_1} \cdot X_{L_2})T \rightarrow Q$$

Again, because X_1 is head-invariant, $X_1 \cdot (X_{L_1} \cdot X_{L_2})P \rightarrow (X_{L_1} \cdot X_{L_2})T \rightarrow Q$, finishing the proof. \square

Finally, define P^b , the *flattening* of a term P , inductively as:

$$x^b = x \quad (Px)^b = P^b x \quad (P(QR))^b = ((PQ)R)^b$$

P^b removes all the non-implicit parenthesis of P , *e.g.* $(x(y(zx)))^b = xyzx$. P^b has the form $x_1 \dots x_m$, preserving the order of the x_i occurring in P .

We can now introduce a four-step algorithm³ to solve the simple constraint $Xx \rightarrow Q(y)$, for $y \subseteq x$, by transforming Q into x .

Algorithm 1 (Simple Constraint Solver).

Input: x and $Q(y)$, with $y \subseteq x$.

Output: A list combinator X_L such that $X_L x \rightarrow Q$.

B-phase: Flatten Q into Q^b . Generate $X_B \equiv \langle B_{(a_1)} \dots, B_{(a_\alpha)} \rangle$ as follows:

```

 $X_B := \langle \rangle$ ;       $T := Q$ ;
While  $T$  is of the form  $z_1 \dots z_i (O_1 O_2) O_3$ 
   $T := z_1 \dots z_i O_1 O_2 O_3$ ;
   $X_B := B_{(i+1)} \cdot X_B$ ;
return  $X_B$ ;

```

At the end of the B-phase, $T = Q^b$.

C-phase: Order Q^b into Q_{\prec} by any order \prec on y . Generate the list combinator $X_C \equiv \langle C_{(b_1)}, \dots, C_{(b_\beta)} \rangle$ as follows:

Apply the bubble sort algorithm [11] to Q^b . Each bubble inversion between y_{b_i} and $y_{b_{i+1}}$ adds a $C_{(b_i)}$ to X_C .

At the end of the C-phase $Q_{\prec} \equiv y_1 \dots y_1 \dots y_m \dots y_m$ and $X_C Q_{\prec} \rightarrow Q^b$.

W-phase: Eliminate duplicates from Q_{\prec} , obtaining y . Generate the list combinator $X_W \equiv \langle W_{(c_1)}, \dots, W_{(c_\gamma)} \rangle$ as follows:

```

 $X_W := \langle \rangle$ ;       $i := 1$ ;
 $T := Q_{\prec}$ ; ( $Q_{\prec}$  is of the form  $y_1 \dots y_1 \dots y_m \dots y_m$ )
While  $i \leq m$ 
  If  $t_i = t_{i+1}$  then
    Delete  $t_{i+1}$  from  $T$ ;
     $X_W := W_{(i)} \cdot X_W$ ;
  Else increment  $i$ ;
return  $X_W$ ;

```

At the end of the W-phase, $T = y_1 \dots y_m = y$, without repetitions.

K-phase: Add extra variables to y to make it identical to x . Generate the list combinator $X_K \equiv \langle K_{(d_1)}, \dots, K_{(d_\delta)} \rangle$ as follows:

```

Let  $x = x_1 \dots x_n$ ,  $y = y_1 \dots y_m$ ,  $m \leq n$ ;
 $X_K := \langle \rangle$ ;       $i := 1$ ;
While  $i \leq m$ 
  If  $x_i \neq y_i$  then
    Delete  $x_i$  from  $x$ ; (corresponds to adding  $x_i$  to  $y$ )
     $X_K := X_K \cdot \langle K_{(i)} \rangle$ ;

```

³ It must be noticed that such a method was hinted at, without details, proof of correctness or list notation, by [4, Section 5B1].

Else increment i ;
return X_K ;

At the end of the K-phase, $x = y$.

Return: $X_K \cdot X_W \cdot X_C \cdot X_B$.

Theorem 1 (Simple Constraint Solution). *Any admissible simple structural constraint of the form $Xx \rightarrow Q(y)$, $y \subseteq x$, has as a solution a list combinator computed by Algorithm 1.*

Proof. It suffices to note that, in Algorithm 1:

- In the B-phase, $X_B Q^b \rightarrow Q$.
- In the C-phase, $X_C Q_{\prec} \rightarrow Q^b$.
- In the W-phase, $X_W y \rightarrow Q_{\prec}$.
- In the K-phase, $X_K x \rightarrow y$.

Then, by three consecutive applications of the Concatenation Lemma we get $(X_K \cdot X_W \cdot X_C \cdot X_B)x \rightarrow Q(y)$. \square

As an example of the application of Algorithm 1, consider the structural constraint of Section 1, $Xxy \rightarrow y(xx)$. B-step gives yx and $\langle B_{(2)} \rangle$; C-step gives xy and $\langle C_{(2)}, C_{(1)} \rangle$; W-step leads to xy and $\langle W_{(1)} \rangle$; and K-step gives $\langle \rangle$. Therefore, $\langle W_{(1)}, C_{(2)}, C_{(1)}, B_{(2)} \rangle =_w W(BC(CB))$ solves the constraint.

It is worth noting that the list format highlights the type of combinators that *really correspond* to structural rules; *e.g.* the list combinator $\langle W_{(2)}, C_{(1)} \rangle$ represent the usage of contraction (W) and commutativity (C), but not of associativity (B), even though its “explicit” format is BW(CI).

With respect to the complexity of Algorithm 1, steps B, W and K traverse the input string only once; therefore those phases are linear with the number N of variables (distinct or not) in the input term. Concatenation is also linear. Step-C is the bubble sort algorithm and its complexity is $\mathcal{O}(N^2)$ [11]. It is the dominating part of the algorithm, whose temporal complexity is then $\mathcal{O}(N^2)$. We could reduce the complexity of the algorithm to $\mathcal{O}(N \log N)$ by replacing combinator C with a “more efficient” version of it, but we will not do it here.

A final remark on Algorithm 1 is that its output is not the only list combinator that solves the constraint. For instance, $\langle W_{(1)}, B_{(1)}, C_{(1)} \rangle$ is also a solution to $Xxy \rightarrow y(xx)$.

4 Complex Structural Constraints

We now face the problem of solving complex structural constraints of the form

$$XP \rightarrow Q.$$

Unfortunately, Lemma 1 does not generalise to complex constraints, and there are admissible complex constraints without a solution in C_0 . In fact, the combinator rule $(B \vdash)$ represents the structural rule for left-associativity, but there is no combinator that represents right-associativity. Let us formally show that.

A *context* $M[\]$ is a combinator term with “holes” or, more formally, is a combinator built extending the set of atomic terms with “[]”. If $M[\]$ is a context and P is a term, $M[P]$ is obtained by placing P in the holes of $M[\]$.

A combinator system \mathbb{C} *represents right-associativity* iff there is a context $M[\]$ in it such that $M[P(QR)] \rightarrow (PQ)R$, for any terms P , Q and R .

Lemma 3. *Suppose there is a combinator system \mathbb{C} that contains combinators K and I and represents right-associativity. Then:*

- (a) *It is not CR.*
- (b) *It is inconsistent.*

Proof. Suppose there is a context $M[\]$ such that $M[P(QR)] \rightarrow (PQ)R$. Make $P \equiv K$, $Q \equiv I$ and $R \equiv xx$. If we start reducing the context we get

$$M[K(I(xx))] \rightarrow KI(xx) \rightarrow I$$

but if we start reducing the redex $I(xx)$ we get

$$M[K(I(xx))] \rightarrow M[K(xx)] \rightarrow Kxx \rightarrow x$$

Since both I and x are in normal form, the system cannot be CR, proving (a).

The proof of (b) follows directly, because we have just shown that $x =_w I =_w y$ for any x and y . \square

Since the \mathbb{C}_0 -system is well-known to be both CR and consistent [1], neither \mathbb{C}_0 nor any consistent extension of it can represent right-associativity. However, we do want to have a consistent system that represents right-associativity, without throwing away any of the combinators in \mathbb{C}_0 (e.g. getting rid of K would invalidate Lemma 3).

To overcome this problem, we propose to restrict the set of combinator terms available for representing structural rules.

4.1 The Restricted Combinator Systems

We define the set of *restricted* terms $\mathcal{C}_c^1 \subset \mathcal{C}_c$:

- every term in \mathbb{C} -normal form is in \mathcal{C}_c^1 .
- if $Q \in \mathcal{C}_c^1$ and $P \rightarrow_c Q$ such that $NRedex_c(P) \leq 1$, then $P \in \mathcal{C}_c^1$.

It follows that there is at most one redex in a term in \mathcal{C}_c^1 . We chose the set \mathcal{C}_c^1 as the appropriate restriction on allowed terms to represent structural rules, concentrating on *restricted combinator systems* $(\mathcal{C}_c^1, \rightarrow_c)$. As usual, we write $(\mathcal{C}_c^1, \rightarrow)$ when the basis is clear from the context. Equality is defined as usual and restricted to $\mathcal{C}_c^1 \times \mathcal{C}_c^1$.

Note that the definition above accepts a combinator as restricted only if all its reductions to a normal form have one redex. In this sense, the term $SIII$ is not a restricted combinator, even though it has a single redex, for $SIII \rightarrow II(II)$, which has two redexes. So, apparently, to verify whether a term is a restricted

combinators, one will have to try to reduce it totally. However, as we will see in Lemma 5, the handy class of *list combinators* is guaranteed to be of the restricted format and one can easily verify pertinence to that class. But first, some nice properties of restricted combinators.

Lemma 4. *Consider the restricted combinator system $(\mathcal{C}_c^1, \rightarrow_c)$ for a given basis \mathbb{C} . Then:*

- (a) \mathcal{C}_c^1 is closed under \rightarrow_c , i.e. if $P \in \mathcal{C}_c^1$ and $P \rightarrow_c Q$ then $Q \in \mathcal{C}_c^1$.
- (b) The combinator system $(\mathcal{C}_c^1, \rightarrow_c)$ is CR.
- (c) $(\mathcal{C}_c^1, \rightarrow_c)$ is consistent.

Proof. (a) Since all combinators in \mathbb{C} are functional, this follows from the definition.

(b) The number of redexes in any term is at most 1, so the system is trivially CR.

(c) Since the system is CR, any two distinct nf-terms are not equal. \square

What is remarkable about this proof is that its only requirement of \mathbb{C} is that its combinators be functional. This means we can now extend \mathbb{C}_0 with functional combinators and still remain (restrictedly) consistent. In the rest of this paper, we deal only with *restricted combinator systems* of the form $(\mathcal{C}^1, \rightarrow)$. We will look for solutions to complex structural constraints in such restricted framework.

One might think that constraining terms to at most one redex is too restrictive. This is not the case. The problem we are trying to solve is to be able to represent the use of structural rules in every intuitionistic deduction by means of combinators. It happens that we cannot do that with generic combinators, but we can do that with restricted ones, as shown in [10]; see Figure 3 below. The weakness of the terms reverts into a benefit, for it allows for more efficient algorithms.

4.2 Inverse Combinators and Complex Combinators

Our idea is to systematically add new combinators to our system to cope with \mathbb{C}_0 's deficiency in capturing all possible intuitionistic structural rules. Thus, we systematically introduce functional inverses of all combinators.

For each combinator in $\mathbb{C}_0 - \{K\}$ we define a corresponding *primitive inverse combinator*⁴, giving their basic reduction rules as follows:

$$\begin{array}{lll} B^{-1}P(QR) \circ \rightarrow PQR & W^{-1}PQQ \circ \rightarrow PQ & S^{-1}PR(QR) \circ \rightarrow PQR \\ C^{-1}PRQ \circ \rightarrow PQR & I^{-1}P \circ \rightarrow P & \end{array}$$

The idea behind primitive inverse combinators is to have a combinator $X^{-1}\mathbf{P} \rightarrow \mathbf{x}$ corresponding to a standard primitive combinator defined by $X\mathbf{x} \rightarrow \mathbf{P}$. Note that the primitive inverses are also head-invariant.

⁴ [7] defines the distinct notion of *dual combinators*, which are combinators that operate right-to-left: $z(y(x\bar{C})) \rightarrow y(zx)$. Unfortunately, duals were represented in [7] by the $^{-1}$ symbol normally used for inverses; but inverses are *not* duals.

<p><i>identity:</i></p> $\frac{\Gamma[\Phi] \vdash \chi}{\Gamma[l, \Phi] \vdash \chi} \quad (l \vdash)$ <p><i>left-associativity:</i></p> $\frac{\Gamma[\Phi, (\Psi, \Xi)] \vdash \chi}{\Gamma[B, \Phi, \Psi, \Xi] \vdash \chi} \quad (B \vdash)$ <p><i>contraction:</i></p> $\frac{\Gamma[\Phi, \Psi, \Psi] \vdash \chi}{\Gamma[W, \Phi, \Psi] \vdash \chi} \quad (W \vdash)$	<p><i>commutativity:</i></p> $\frac{\Gamma[\Phi, \Xi, \Psi] \vdash \chi}{\Gamma[C, \Phi, \Psi, \Xi] \vdash \chi} \quad (C \vdash)$ <p><i>right-associativity:</i></p> $\frac{\Gamma[\Phi, \Psi, \Xi] \vdash \chi}{\Gamma[B^{-1}, \Phi, (\Psi, \Xi)] \vdash \chi} \quad (B^{-1} \vdash)$ <p><i>expansion:</i></p> $\frac{\Gamma[\Phi, \Psi] \vdash \chi}{\Gamma[W^{-1}, \Phi, \Psi, \Psi] \vdash \chi} \quad (W^{-1} \vdash)$ <p><i>thinning:</i></p> $\frac{\Gamma[\Phi] \vdash \xi}{\Gamma[K, \Phi, \Psi] \vdash \xi} \quad (K \vdash)$
--	--

Fig. 3. Combinator rules and their associated structural roles

Combinator B^{-1} represents right associativity, and therefore cannot be represent in terms of the combinators in \mathbb{C}_0 . Combinator W^{-1} demands two terms to be identical in the redex, and is also not definable in \mathbb{C}_0 . S^{-1} is also not definable in \mathbb{C}_0 . However, C^{-1} has exactly the same behaviour as C , so we make $C^{-1} \equiv C$; similarly, $l^{-1} \equiv l$.

The inverse of K was ruled out. If K^{-1} were allowed, it would be possible to build a combinator that reduces to any term. In fact, since for any P and Q $KPQ \circ \Rightarrow P$, then $K^{-1}P \circ \Rightarrow PQ$, so $K^{-1}l \Rightarrow Q$ for any Q . This means that for any P and Q , $P =_w K^{-1}l =_w Q$; so K^{-1} is not functional and it trivializes reduction, making the system inconsistent.

We can add these new combinators to \mathbb{C}_0 and use it as a basis.

Definition 1 (Complex Combinators). Let $\mathbb{C}_1 = \mathbb{C}_0 \cup \{S^{-1}, B^{-1}, W^{-1}\}$ be the set of primitive complex combinators. The complex combinator system is the restricted system $(\mathcal{C}_{\mathbb{C}_1}^1, \rightarrow_{\mathbb{C}_1})$ based on \mathbb{C}_1 ; we will represent it only as $(\mathcal{C}^1, \rightarrow)$ when the context of complex combinators is clear. A complex combinator is a term in \mathcal{C}^1 built only with the primitive combinators in \mathbb{C}_1 .

The set of relevant structural rules representable by combinator rules in $\text{SFL}(\mathbb{C}_1)$ is illustrated in Figure 3.

A *standard combinator* is one without the occurrence of an inverse. Equality for complex combinator terms is defined in the usual way. We also write $X_{(i)}^{-1}$ for $(X^{-1})_{(i)}$. We then define $\mathbb{C}_1^* = \{X_{(i)} \mid X \in \mathbb{C}_1 - \{l\}, i \geq 1\}$, and build complex list combinators X_L with the elements of \mathbb{C}_1^* ; as in the standard case, $X_L \in \mathcal{C}^1$.

We now extend the notion of inverses to a larger class of complex combinators.

Definition 2 (Complex Inverses). Let X and X^\bullet be a K -free complex combinators. X^\bullet is an inverse combinator of X whenever

$$\text{if } XP \rightarrow Q \text{ then } X^\bullet Q \rightarrow P$$

for all pure terms P and Q , $\text{Var}(P) = \text{Var}(Q)$.

It follows directly from Definition 2 that if X^\bullet is an inverse of X then X is an inverse of X^\bullet . Not all complex combinators in \mathcal{C}^1 have an inverse; apart from combinators in which K occurs, the divergent combinators such as WWW and $Wl(Wl)$ do not have an inverse⁵ according to Definition 2. Nor is the inverse unique: $\langle C \rangle$ has inverses $\langle C \rangle$, $\langle C, C, C \rangle$, $\langle C, C, C, C, C \rangle$, etc.

By dealing with complex list combinators based on \mathbb{C}_1^* , we are guaranteed to stay inside the restricted framework, and therefore we have a consistent and CR system to represent structural rules, as shown below.

Lemma 5. Let $X_L \equiv \langle X_1, \dots, X_n \rangle$ such that each $X_i \in \mathbb{C}_1^*$. Then $X_L \in \mathcal{C}^1$. If P is pure, $X_L P \in \mathcal{C}^1$.

Proof. By induction on n . For the base case, just note that $\langle \rangle \equiv I \in \mathcal{C}^1$. Then note that if $X_2 \in \mathcal{C}^1$ and X_1 is head-invariant then $X_1 X_2 \in \mathcal{C}^1$ because $X_1 X_2$ is normal form, which takes care of the induction case. A similar induction shows that $X_L P \in \mathcal{C}^1$. \square

We next prove that a K -free complex list combinators always has an inverse that is easily computed.

4.3 Computing List Inverses

We start by defining a function X^{-1} on each K -free $X \in \mathbb{C}_1^*$ (so far, X^{-1} was defined only for $X \in \mathbb{C}_0$). Let $X \in \mathbb{C}_1 - \{I, K\}$ and $n \geq 1$. Then:

$$\begin{aligned} (X_{(n)})^{-1} &= (X^{-1})_{(n)} \equiv X_{(n)}^{-1} \\ (X_{(n)}^{-1})^{-1} &= X_{(n)} \end{aligned}$$

It is not difficult to see that for K -free $X \in \mathbb{C}_1^*$, X^{-1} is an inverse of it. Extend this definition to list combinators in the following way. Let X_L be a K -free list combinator and $X \in \mathbb{C}_1^*$:

$$\begin{aligned} (\langle \rangle)^{-1} &= \langle \rangle \\ (\langle X \rangle \cdot X_L)^{-1} &= X_L^{-1} \cdot \langle X^{-1} \rangle \end{aligned}$$

In other words, $\langle X_1, \dots, X_n \rangle^{-1} = \langle X_n^{-1}, \dots, X_1^{-1} \rangle$.

We still need to show that X_L^{-1} is actually an inverse of X_L . First, we need to extend the Concatenation Lemma for complex list combinators.

⁵ but note that these combinators are not equivalent to list combinators.

Lemma 6. *The Concatenation Lemma holds for complex combinators. That is, for \mathbb{C}_1^* -based complex combinator lists X_{L_1} and X_{L_2} , and pure terms P and Q , $(X_{L_1} \cdot X_{L_2})P \rightarrow Q$ iff there exists pure O such that $X_{L_1}P \rightarrow O$ and $X_{L_2}O \rightarrow Q$.*

Proof. Just notice that the proof of Concatenation Lemma (Lemma 2) uses two basic properties of the elements of the list: the fact that they are head-invariant and that they possess the Church-Rosser property. Both of them are present in complex list combinators due to the fact that members of \mathbb{C}_1^* are head-invariant and Lemma 4. So that proof of Lemma 2 applies to restricted complex list combinators. \square

Theorem 2. *Let $X_L \equiv \langle X_1, \dots, X_n \rangle$ be a K -free complex list combinator. Then X_L^{-1} is an inverse of X_L .*

Proof. By induction on the length n of the list. The base case trivially deals with the empty list. For the inductive case, let P and O be any pure terms such that $X_L P \rightarrow O$.

Since $X_L \equiv \langle X_1 \rangle \cdot \langle X_2, \dots, X_n \rangle$, by the Concatenation Lemma, there must exist a pure Q such that $\langle X_1 \rangle P \rightarrow Q$ and $\langle X_2, \dots, X_n \rangle Q \rightarrow O$.

From the induction hypothesis, we get $\langle X_2, \dots, X_n \rangle^{-1} O \rightarrow Q$. Also note that $\langle X_1^{-1} \rangle Q \rightarrow P$. Then, by the Concatenation Lemma,

$$X_L^{-1} O \equiv (\langle X_2, \dots, X_n \rangle^{-1} \cdot \langle X_1^{-1} \rangle) O \rightarrow P$$

finishing the proof. \square

Algorithm 2 (List Inverter).

Input: a K -free list combinator of the form $\langle X_1, \dots, X_n \rangle$.

Output: $\langle X_n^{-1}, \dots, X_1^{-1} \rangle$.

4.4 Solutions for Complex Constraints

Given an admissible structural constraint⁶ $X P \rightarrow Q$, the idea is to find x such that $X_{L_1} x \rightarrow P$ and $X_{L_2} x \rightarrow Q$. The solution will be $X = X_{L_1}^{-1} \cdot X_{L_2}$.

Algorithm 3 (Structural Constraint Solver).

Input: pure terms P and Q , $Var(Q) \subseteq Var(P)$.

Output: a complex list combinator X_L satisfying $X_L P \rightarrow Q$.

Let $x = Var(P)$. Establish an arbitrary order between the $x_i \in x$, $x_1 \prec \dots \prec x_n$. Then:

1. Using Algorithm 1, compute the (standard) list combinator X_{L_1} such that $X_{L_1} x \rightarrow P$.

⁶ According to the assumption made in Section 2, in $P \equiv P_1 \dots P_n$ the term for P_1 is atomic; see at the end of this Section how to avoid such a restriction.

2. Using the list inverter (Algorithm 2), compute $X_{L_1}^{-1}$.
3. Again using list the simple constraint solver (Algorithm 1), compute the list combinator X_{L_2} such that $X_{L_2}\mathbf{x} \rightarrow \mathbf{Q}$.

Return $X_L = X_{L_1}^{-1} \cdot X_{L_2}$.

Steps 1 and 3 have the complexity of the simple constraint solver. Step 2 is list inverter, with linear time complexity. So Algorithm 3 has complexity $\mathcal{O}(N^2)$, where N is the maximum of the lengths of \mathbf{P} and \mathbf{Q} .

Theorem 3 (Complex Solution). *A complex structural constraint $X\mathbf{P} \rightarrow \mathbf{Q}$ has a \mathbb{C}_1^* -list solution iff it is admissible; this solution is computed by Algorithm 3.*

Proof. If $\text{Var}(\mathbf{Q}) \not\subseteq \text{Var}(\mathbf{P})$ then there is a variable $x \in \text{Var}(\mathbf{P}) - \text{Var}(\mathbf{Q})$, and since no combinator creates new variables there cannot exist a solution for $X\mathbf{P} \rightarrow \mathbf{Q}$.

Now assume $\text{Var}(\mathbf{Q}) \subseteq \text{Var}(\mathbf{P})$. Let $\mathbf{x} = \text{Var}(\mathbf{P})$. We know from Theorem 1 that there exists a list combinator X_{L_1} such that $X_{L_1}\mathbf{x} \rightarrow \mathbf{P}$. From $\mathbf{x} = \text{Var}(\mathbf{P})$, it follows that X_{L_1} is K-free. From Theorem 2, we know that X_{L_1} has an inverse such that $X_{L_1}^{-1}\mathbf{P} \rightarrow \mathbf{x}$.

Again by Theorem 1, we know there exists a list combinator X_{L_2} such that $X_{L_2}\mathbf{x} \rightarrow \mathbf{Q}$ (this one may not be K-free). And finally, by the Concatenation Lemma, we get that $(X_{L_1}^{-1} \cdot X_{L_2})\mathbf{P} \rightarrow \mathbf{Q}$, solving the intended structural constraint. To finalise, just notice that $(X_{L_1}^{-1} \cdot X_{L_2})$ is computed by Algorithm 3. \square

Let us show a few examples of the application of Algorithm 3. First, the solution for $Xx(yz) \rightarrow xyz$ is $\langle \mathbf{B}_{(2)}^{-1} \rangle$ (which is the list combinator that defines \mathbf{B}^{-1}), where the algorithm above gives X_{L_1} as $\langle \mathbf{B}_{(2)} \rangle$ and X_{L_2} as $\langle \rangle$.

The solution for constraint $Xy(xx) \rightarrow xy$ is $\langle \mathbf{B}_{(2)}^{-1}, \mathbf{C}, \mathbf{C}_{(2)}, \mathbf{W}^{-1} \rangle$ (exactly the inverse of the solution for $Xxy \rightarrow y(xx)$). This solution was found by arbitrarily fixing the order $x \prec y$; had we fixed $y \prec x$ the solution would have been $\langle \mathbf{B}_{(2)}^{-1}, \mathbf{W}_{(2)}^{-1}, \mathbf{C} \rangle$. This shows that the solution is sensitive to the order chosen. However, note that the primitive combinators used in both solutions are the same. This should be always the case, except that the combinator \mathbf{C} may be eliminated from some solutions, when a particular choice of \prec avoids reordering.

The constraint $X(xy) \rightarrow x$ violates the assumption made in Section 2 that in $\mathbf{P} \equiv P_1 \dots P_n$, the term for P_1 is atomic; in this case $\mathbf{P} \equiv P_1 \equiv (xy)$, and Algorithm 3 gives the wrong answer $\langle \mathbf{K}_{(2)} \rangle$ for it computes $X_{L_1} = \langle \rangle$ instead of $\langle \mathbf{B} \rangle$. However, a very simple alteration in the algorithm can treat this case by changing step 1 with 1'.

- 1'. Take a new variable $w \notin \mathbf{x}$ and compute $X'_{L_1}, X'_{L_1}w\mathbf{x} \rightarrow w\mathbf{P}$. Now compute X_{L_1} by decrementing i in each $X_{(i)}$ in X'_{L_1} ; the rest of the algorithm remains the same.

It is not hard to see that we have computed a X_{L_1} such that $X_{L_1}^{-1}\mathbf{P} \rightarrow \mathbf{x}$ even when P_1 is not atomic. In the case of $X(xy) \rightarrow x$, we compute $X'_{L_1}wxy \rightarrow w(xy)$, obtaining $\langle \mathbf{B}_{(2)} \rangle$, and X_{L_1} is $\langle \mathbf{B} \rangle$; then by computing $X_{L_2}xy \rightarrow x$ we get $\langle \mathbf{K}_{(2)} \rangle$ and the final answer is $\langle \mathbf{B}^{-1}, \mathbf{K}_{(2)} \rangle$.

4.5 Minimal Solutions

Algorithm 3 does not give us “minimal” solutions. For example, given the constraint $Xx(yz)ab \rightarrow x(yz)(ab)$, Algorithm 3 finds the solution $X = \langle B_{(2)}^{-1}, B_{(2)}, B_{(3)} \rangle$. However, $X = \langle B_{(3)} \rangle$ is also a solution. It is a preferred solution according to the following definition.

Let X_1 and X_2 be two solutions to $XP = Q$ such that

$$\begin{aligned} X_1P &\rightarrow X_{1,1}P_{1,1} \rightarrow \dots \rightarrow X_{1,n}P_{1,n} \rightarrow Q \\ X_2P &\rightarrow X_{2,1}P_{2,1} \rightarrow \dots \rightarrow X_{2,m}P_{2,m} \rightarrow Q \end{aligned}$$

Then solution X_1 is *preferable* to X_2 if $\{P_{1,1}, \dots, P_{1,n}\} \subset \{P_{2,1}, \dots, P_{2,m}\}$.

The preferred solution in the example above is easily computed with a preprocessing of the input. The non-preferred solution is computed due to the fact that the variables y and z occur in $x(yz)ab$ and $x(yz)(ab)$ only as the subterm yz . We substitute the subterm yz by a new variable, say y' , so that we now solve the constraint $Xxy'ab \rightarrow xy'(ab)$, obtaining $X = \langle B_{(3)} \rangle$. Such a preprocessing was included in our implementation.

However, the “minimal solution” is not unique. For example, consider the constraint $Xx(xy) \rightarrow xy$. The simplification above does not apply, for x occurs both inside and outside xy . Algorithm 3 yields $X = \langle B_{(2)}^{-1}, W_{(1)}^{-1} \rangle$. However, if we do apply the simplification above, making $y' = xy$, we get the solution $X = \langle K_{(1)} \rangle$, which is also correct. There is no a priori reason to prefer either $\langle B_{(2)}^{-1}, W_{(1)}^{-1} \rangle$ or $\langle K_{(1)} \rangle$, so in this case the notion of “minimal” solution depends on further assumptions. There are several candidates for what constitutes a “minimal” solution:

1. smallest number of K 's (avoid weakening at all cost);
2. smallest number of W 's and W^{-1} 's (avoid contraction/expansion at all cost);
3. smallest combinator size; etc.

Neither of the above guarantees a unique minimal solution for all cases. For condition (1), note that it is always possible to have at most one K in the solution of $XP \rightarrow Q$: if $Var(P) = Var(Q)$ then Algorithm 3 gives a K -free solution. Otherwise, solve:

$$X_1P \rightarrow x(K) \quad \text{and} \quad X_2x \rightarrow Q$$

where $x = Var(Q)$ and $K = Var(P) - Var(Q)$. In this case, neither X_1 nor X_2 contains a K , and $X = X_1 \cdot \langle K_{(n)} \rangle \cdot X_2$, where $n = |x|$.

For (2), we can eliminate all the occurrences of W^{-1} from the solution in the following way: for each variable occurring more than once in P , let K above contain all but one of its occurrences. Then no W^{-1} occurs in X .

Substitution of a common term by a new variable (even in cases where its variables occur elsewhere) decreases the number of W 's in the solution, but no minimal number is guaranteed this way. To obtain the combinator with the smallest size, one would have to try all the possible orders in Algorithm 1.

4.6 Implementation

An implementation of the Complex Structural Constraint Solver was done in Java 1.1 and is available in the web at the following address:

<http://www.ime.usp.br/~mfinger/sftp/java/sftp.html>

5 Comparisons with the Literature

The piece of work in the literature that mostly approximate ours is that of Natasha Kurtonina [12]. There, an algorithm for computing a first-order formula expressing the model-theoretic constraint associated to a CG axiom is presented. The approach, however, is semantical, based on the three-place relation for substructural model frames. In that sense, Kurtonina's work is complementary to ours, which is purely syntactical.

A similar but distinct problem of *solvability* is found in the literature, as studied by Wadsworth [18]. A term M is *solvable* if there are terms N_1, \dots, N_r such that $MN_1 \dots N_r =_w I$; solvability has applications in the study of the definability of partial functions [1].

In their presentation of SFL, Dunn and Meyer [7] introduce the notion of *dual combinators*, which operate right-to-left: $z(y(x\bar{C})) \rightarrow y(zx)$. More recent developments have shown that several combinator systems involving both standard and dual combinators are inconsistent [13,3] (this is indeed the case even for systems that do not represent right-associativity).

What best approaches SFTP in the literature is the notion of a *modular theorem prover* for substructural logics, in which a basic proof procedure is presented for all substructural logics, differing only in a parameterized element. In [6] such parameter is the closure rule for analytic tableaux. In [2], this parameter is the licensing of the application of a Natural Deduction rule.

6 Conclusions

Computing an efficient solution for complex structural constraints shows that structurally free theorem provers can be constructed for the \bullet -fragment of substructural logics. The next step should be to extend the logic SFL to incorporate complex combinators and then investigate larger fragments.

Steps in that direction were given in [10], where the $\{\bullet, /\}$ -fragment is investigated; it was shown that all intuitionistic deductions could be represented by SFL with the combinators B, C, B^{-1}, I, W and K (W^{-1} can always be replaced by K). Note that if we can treat the $\{\bullet, /, \backslash\}$ -fragment, we will be dealing with the basic connectives of the Lambek Calculus, a logic with wide applications in Computation Linguistics. SFTP may then become useful to the automated learning of structural rules involved in language parsing.

Another potential area of applications for SFL and SFTP is to bridge the *logical*⁷ gap existing between two approaches of Categorical Grammar. On one

⁷ Of course, this approach will not be able to solve the *methodological* differences between the two approaches. For instance, the way Steedman's CCG treats the *extraction* phenomenon is very different from how Moortgat's CG does it.

hand, there is the deductive approach, recently summarized in [14]. On the other hand, there is Steedman's Combinatory Categorical Grammar [17], in which certain operations, related to the application of combinators, are introduced in the deductive system. We believe that the presence of combinators as first class element in SFL may be of use here.

Acknowledgments. The author would like to thank Katalin Bimbó for much appreciated discussions and invaluable comments on earlier versions of this paper.

References

1. Barendregt, H. P. *The Lambda Calculus: Its Syntax and Semantics*. Number 103 in North-Holland Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers, 1981.
2. Broda, K., M. D'Agostino, and A. Russo. Transformation methods in lds. In *Logic, Language and Reasoning. An Essay in Honor of Dov Gabbay*. Kluwer Academic Publishers, 1996.
3. Bimbó, K. Investigation into combinatory systems with dual combinators. To appear in *Studia Logica*.
4. Curry, H. B., and R. Feys. *Combinatory Logics*, volume 1. North Holland, 1958.
5. Curry, H. B., J. R. Hindley, and J. P. Seldin. *Combinatory Logics*, volume 2. North Holland, 1972.
6. D'Agostino, M., and D. Gabbay. A Generalization of Analytic Deduction via Labelled Tableaux, part I: Basic Substructural Logics. *Journal of Automated Reasoning*, 13:243–281, 1994.
7. Dunn, J. M., and R. K. Meyer. Combinators and Structurally Free Logic. *Logic Journal of the IGPL*, 5(4):505–538, July 1997.
8. Došen, K. A Historical Introduction to Substructural Logics. In Heister, P. S., and K. Došen, editors, *Substructural Logics*, pages 1–31. Oxford University Press, 1993.
9. Dunn, J. M. Proof Theory and Semantics for Structurally Free Logics. In *Proceedings of the 3rd Workshop of Logic, Language, Information and Computation (WoLLIC96)*, 1996. Abstract only.
10. Finger, M. Towards structurally-free theorem proving. *Logic Journal of the IGPL*, 6(3):425–449, 1998.
11. Knuth, D. E. *The Art of Computer Programming*, volume 3, Sorting and Searching. Addison-Wesley, 1973.
12. Kurtonina, N. *Frames and Labels — A Modal Analysis of Categorical Inference*. PhD thesis, Research Institute for Language and Speech (OTS), Utrecht, and Institute of Logic, Language and Information (ILLC), Amsterdam, 1994.
13. Meyer, R., K. Bimbó, and J. M. Dunn. Dual combinators bite the dust (abstract). forthcoming in *The Journal of Symbolic Logic*, (Presented at the Annual Conference of the Australasian Association for Logic, July 4–6, 1997, University of Auckland, New Zealand.).
14. Moortgat, M. Categorical type logics. In Benthem, J. van, and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 93–178. Elsevier North-Holland/The MIT Press, 1997.

15. Revesz, G. *Lambda Calculus, Combinatorics and Functional Programming*. Number 4 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1988.
16. Schönfinkel, A. Über die Bausteine der Mathematischen Logik. In Heijenoort, J. van, editor, *From Frege to Gödel*. Harvard University Press, Cambridge, Massachusetts, 1967. Reprinted from original version, 1924.
17. Steedman, M. Combinators and grammar. In R. T. Oehrle, E. Bach, and D. Wheeler, editors, *Categorial Grammars and Natural Language Structures*, Studies in Linguistics and Philosophy, pages 417–442. D. Reidel Publishing Company, 1988.
18. Wadsworth, C. P. *Semantics and Pragmatics of the Lambda-calculus*. PhD thesis, Oxford University, 1971.

Hypothetical Reasoning and Basic Non-constituent Coordination in Type-Logical Grammar

Nissim Francez

Computer Science Dept., Technion-IIT, Haifa, Israel
francez@cs.technion.ac.il

Abstract. The paper investigates the connection between non-constituent coordination, as implemented in categorial grammar by means of a polymorphic type-assignment to lexical conjunctions, and hypothetical reasoning in Categorical Logics. A way of extending the logic is suggested, so that coordination can be applied to types depending on undischarged assumptions. By a certain “resource manipulation” of assumptions (of hypothetical reasoning), a late-discharge is facilitated, leading to what is referred to as the *basic non-constituent coordination*, whereby only basic types (and not functional types of any kind) are coordinated. The approach also provides a syntactic counterpart to the usual definition of generalized meet to higher-order functions into a boolean range, stipulated in the literature as the meaning of coordination.

1 Introduction

An acclaimed advantage of Categorical Grammar (CG) as a theory of the syntax of natural language(s) is its smooth syntactic analysis for *non-constituent coordination* (*NCC*); via the Curry-Howard correspondence, the semantics falls out naturally too. Thus, coordinations like

- (1) Mary kissed John and hugged Bill
- (2) Mary kissed and Sue hugged John

are treated on a par. This is done via the type assignment of the universally quantified type $\forall X : ((X \rightarrow X) \leftarrow X)$ (abbreviated below to κ) to lexical conjunctions. This type is used in conjunction with the following *ncc*-rule (which covers constituent-coordination as a special case):

$$(ncc) \quad \frac{\tau, \kappa, \tau}{\tau}.$$

This rule abbreviates an instantiation of the bound type-variable X to τ , and two actual arrow-elimination steps – see below. We use the more self-explanatory arrow-notation for categories, instead of the original slash-notation. To allow the

whole range of *NCC*-derivations (see [3] for the extent of *NCC*), the *ncc*-rule is accompanied by (forward and backward) composition-rules and type-raising rules, leading to *Combinatory Categorical Grammar (CCG)* [15]. As a consequence of the universal quantification on types in the κ type (even if the instantiation of X is restricted only to certain linguistically motivated types) a proliferation of conjoinable types emerges. An alternative route to the composition and type-raising rules is obtained by passing to more expressive formalisms, known as *Type (Categorical) Logics*, such as the various variants of the Lambek calculus \mathcal{L} , which have rules for *hypothetical reasoning*. In such systems, the composition and type-raising rules become¹ derived proof-rules. For a recent survey, see [10].

What is the relationship between *ncc*-derivations and hypothetical reasoning? There seems to be no *explicit* account of it in the literature. In [10], there is an implicit consideration of this combination. The only example (on p. 114, repeated in Section 2) has the following characteristic: when the *ncc*-rule is applied, both coordinated types had their assumptions all discharged; arrow-introduction rules are applied before coordination via *ncc*.

Similarly, all the examples in [11] (Chapter Three, Section 3) have the same characteristic, and the same goes for [2]. Thus, even though the interaction of the *ncc*-rule with hypothetical reasoning is not explicitly stated to satisfy the above characteristic, it seems that this is the intended combination of an *ncc*-rule with \mathcal{L} . Note that “blind application” of the *ncc*-rule in \mathcal{L} to types depending on undischarged assumptions is indeed not clear. Assuming only peripheral abstraction, after conjoining two types (each of which peripherally depending, say, on one assumption), one of the two assumptions ceases to be peripheral and becomes nondischargable for the rest of the derivation. Intuitively, the reason is that the two gaps that serve as assumptions should be filled by *the same filler* for the application of the *ncc*-rule to yield the right result.

When semantics is considered too, the notation of the *ncc*-rule is somewhat misleading; though all occurrences of τ in the *ncc*-rule have the same category type as value (the type τ to which X is instantiated in κ), these occurrences may have different interpretations as the semantic counterparts of their value. Thus, in the “colon notation”, one would write more fully

$$(ncc) \quad \frac{\tau : \alpha, \kappa : \lambda f. \lambda g. f \sqcap g, \tau : \beta}{\tau : \alpha \sqcap \beta}$$

under the usual assumption that *all* semantic domains are closed w.r.t. ‘ \sqcap ’. Indeed, in [3] ‘ \sqcap ’ is not further analyzed. However, [15] defines conjoining functions, following [12] and [7], according to which

$$(\lambda x_1. f(x_1) \sqcap \lambda x_2. g(x_2)) =_{df.} \lambda x. (f(x) \sqcap g(x))$$

¹ Though not the third CCG rule, known as functional substitution, nor the “crossed composition” variants of composition.

(and recursively for Curried functions of higher type). When used for *NCC*, while syntactically functor-categories are conjoined, semantically only result-category interpretations are conjoined. This mismatch suggest some syntactic operation, under which the semantic definition will fall-out by (a suitable extension of) the Curry-Howard correspondence. We mention here that the semantics of [15], as well as our revision presented below, deal only with wide scope of quantifiers (under their usual *Generalized quantifiers* interpretation) w.r.t. coordination (an example is presented later). For full scope control, an orthogonal devise of scope modalities is used, see e.g, [2].

In the present paper, we have two main purposes. On the more technical level, the paper provides an explicit treatment of the combination of the *ncc*-rule with \mathcal{L} . In particular, it provides a way of applying the *ncc*-rule *prior* to the arrow-introduction rules, under a certain side condition manipulating assumptions (thereby also reducing overgeneration), and only afterwards discharging the (manipulated) assumption(s). Admittedly, the proposed rule transcends the traditional framework of type-logical grammar, where only introduction and elimination rules for the various operators are used, in addition to structural rules. This point is discussed more after the exposition of the proposed system.

As for the application of \mathcal{L} as a formalism for grammar formulation for NL, the proposal here provides another view of *NCC*. On the categorial level, the coordinated categories are always constituents. On the phrasal level, the coordinated phrases are what we call *pseudo-constituents*: constituents with gaps. This way, non-constituent coordination is perceived as a filler-gap structure. The key observation is, that several gaps may be associated with the *same* filler. This is an elaboration of the observation in [9], whereby *NCC*-sentences are viewed as having shared substructures. The shared substructures (with their gaps) are pseudo-constituents, and the intended effect of the application of a revised *ncc*-rule is to *identify the two gaps into one gap, by identifying the assumptions (for hypothetical reasoning) introduced by these gaps*. Upon application of an arrow-introduction rule, the conjoined pseudo-constituents have one antecedent, while discharging the assumption resulting from the identification; this represents a *simultaneous* discharge of the original assumptions. Finally, by using a filler for the gap via an arrow-elimination rule, both original gaps can be perceived as having been filled by the same filler. Below, we present the *basic non-constituent coordination thesis*, that in a way eliminates the whole *NCC* as being different in essence from constituent-coordination. It enjoys the additional advantage of restricting coordination to basic categories only, thereby avoiding the proliferation of types over which universal quantification takes place in κ . It also blocks some of the well-known overgenerated *NCCs* by the traditional approach.

2 Preliminaries

We briefly review the basic notions and the notation used. The (associative, product-free) Lambek calculus \mathcal{L} [8] is a calculus for deriving valid *declarative units* of the form $U \triangleright A$, where U is a sequence of categories, and A is a category. The set \mathcal{C} of *types (categories)* is the closure of a finite set \mathcal{B} of *basic types* under ‘ \leftarrow ’ and ‘ \rightarrow ’. The meaning of a declarative unit is that the sequence U reduces to A . Validity is defined in terms of denotations over a string model (see [10]). A natural deduction version of the calculus for \mathcal{L} is presented in Figure 1. Assumptions

$$\begin{array}{c}
 (ax) \ A \triangleright A \\
 (\rightarrow E) \ \frac{U_1 \triangleright B, \ U_2 \triangleright (B \rightarrow A)}{(U_1 U_2) \triangleright A}, \quad (\leftarrow E) \ \frac{U_2 \triangleright (A \leftarrow B), \ U_1 \triangleright B}{(U_2 U_1) \triangleright A} \\
 (\rightarrow I) \ \frac{(BU) \triangleright A}{U \triangleright (B \rightarrow A)}, \quad (\leftarrow I) \ \frac{(UB) \triangleright A}{U \triangleright (A \leftarrow B)}
 \end{array}$$

Fig. 1. The \mathcal{L} -calculus

are enclosed in square brackets and indexed for reference when discharged by arrow-introduction rules. The *ncc*-rule for \mathcal{L} is (*ncc*) $\frac{U_1 \triangleright \tau, \ U_2 \triangleright \kappa, \ U_3 \triangleright \tau}{(U_1 U_2 U_3) \triangleright \tau}$.

Traditional CG is equivalent to the Ajdukiewicz-fragment \mathcal{A} , which does not have the hypothetical reasoning rules of arrow-introduction ($\leftarrow I$) and ($\rightarrow I$). The combinatory-categorical grammar (CCG) is obtained by augmenting \mathcal{A} with additional specific rules: Type-raising, composition, and substitution. Figure 2 presents two of these rules, the (forwards and backwards) type-raising rules and composition-rules. The substitution rule is not needed for the current discussion.

$$\begin{array}{c}
 (\Rightarrow T) \ \frac{U \triangleright A}{U \triangleright (B \rightarrow A) \leftarrow B}, \quad (\Leftarrow T) \ \frac{U \triangleright A}{U \triangleright (B \leftarrow A) \rightarrow B} \\
 (\Leftarrow C) \ \frac{U_1 \triangleright A \leftarrow B, \ U_2 \triangleright B \leftarrow C}{(U_1 U_2) \triangleright A \leftarrow C}, \quad (\Rightarrow C) \ \frac{U_1 \triangleright A \rightarrow B, \ U_2 \triangleright B \rightarrow C}{(U_1 U_2) \triangleright A \rightarrow C}
 \end{array}$$

Fig. 2. Additional CCG rules

For a detailed exposition of *CCG* see [15]. The above rules are validity preserving and can be derived in \mathcal{L} , though not in \mathcal{A} .

We present below two typical derivations using the *ncc*-rule in \mathcal{L} . By mimicking the derivation in [10] (p. 114), we obtain a derivation of *left-NCC*, shown in Figure 3. Note that all hypothetical reasoning and assumption discharge in the derivation in Figure 3 precedes the application of the *ncc*-rule. In the latter, X in κ is instantiated to $(s \leftarrow np)$. An \mathcal{L} -derivation of a right-*NCC* ([3])

$$\begin{array}{c}
 \text{Mary} \quad \frac{\text{kissed} \quad \frac{((np \rightarrow s) \leftarrow np)}{np \rightarrow s} \quad \frac{-}{[np]_1}}{np} \leftarrow E \quad \frac{s}{s \leftarrow np} \leftarrow I_1 \quad \rightarrow E \\
 \hline
 s \leftarrow np \\
 \hline
 \text{Sue} \quad \frac{\text{hugged} \quad \frac{((np \rightarrow s) \leftarrow np)}{np \rightarrow s} \quad \frac{-}{[np]_2}}{np} \leftarrow E \quad \frac{s}{s \leftarrow np} \leftarrow I_2 \quad \rightarrow E \\
 \hline
 s \quad \text{John} \\
 \hline
 s \quad np \quad \leftarrow E
 \end{array}$$

Fig. 3. A left-ncc derivation in \mathcal{L}

(3) Mary gave a book to John and a record to Bill.
 is presented in Figure 4. Note the instantiation of X in κ to the complex type $((vp \leftarrow pp) \leftarrow np) \rightarrow vp$. In Figure 5 we present an example of an incor-

$$\begin{array}{c}
 \frac{-}{[(vp \leftarrow pp) \leftarrow np]_1} \quad \frac{\text{a book}}{np} \leftarrow E \quad \frac{\text{to John}}{pp} \leftarrow E \\
 \hline
 vp \leftarrow pp \quad \rightarrow I_1 \\
 \hline
 ((vp \leftarrow pp) \leftarrow np) \rightarrow vp \\
 \hline
 \text{-----} \\
 \frac{-}{[(vp \leftarrow pp) \leftarrow np]_2} \quad \frac{\text{a record}}{np} \leftarrow E \quad \frac{\text{to Bill}}{pp} \leftarrow E \\
 \hline
 vp \leftarrow pp \quad \rightarrow I_2 \\
 \hline
 ((vp \leftarrow pp) \leftarrow np) \rightarrow vp \\
 \hline
 \text{-----} \\
 \text{Mary} \quad \frac{\text{gave} \quad \frac{((vp \leftarrow pp) \leftarrow np) \rightarrow vp}{(vp \leftarrow pp) \leftarrow np} \quad \frac{\text{and} \quad \kappa \quad ((vp \leftarrow pp) \leftarrow np) \rightarrow vp}{((vp \leftarrow pp) \leftarrow np) \rightarrow vp} \quad ncc}{np} \rightarrow E \\
 \hline
 s
 \end{array}$$

Fig. 4. A right-ncc derivation in \mathcal{L}

rect \mathcal{L} -derivation, arising by applying “blindly” the *ncc*-rule to types depending on assumptions. The second, non-peripheral, abstraction is of unclear nature,

and is marked with ‘?’ . Note that the assumption $[np]_1$ becomes peripherally-nondischageable after the alleged application of the *ncc*-rule.

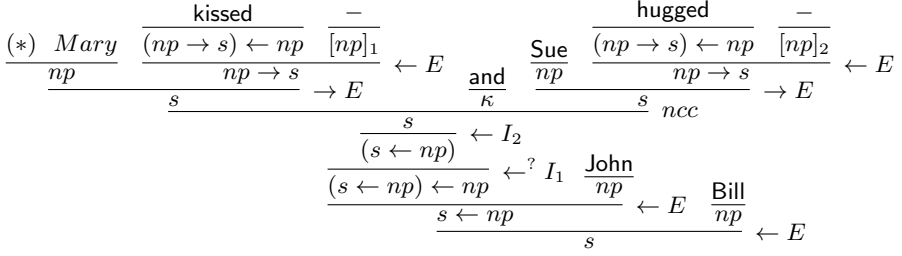


Fig. 5. Incorrect left-*ncc* \mathcal{L} -derivation with *ncc*-rule?

3 Combining Hypothetical Reasoning with a Revised Ncc-Rule

We now provide an explicit account of the combination of hypothetical reasoning with a revised form of the *ncc*-rule, which allows the combination of types depending on undischarged assumptions. The main idea is, that in order for the revised *ncc*-rule to apply, the two copies of X not only are instantiated in κ to the same type, but have to be *alignable* in terms of the (undischarged) assumptions on which these two X s depend² (if any): *In an \mathcal{L} -derivation, two instances of a category X are alignable iff they depend on assumptions of the same type and of equi-peripherality.* Let us restrict for a while the discussion to pseudo-constituents with (at most) a *single*, peripheral gap. Alignment is possible in any of the following cases:

1. Both instances of X do not depend on any assumptions,
2. Both instances of X depend on left-peripheral assumptions (of the same type),
3. Both instances of X depend on right-peripheral assumptions (of the same type).

The revised *ncc*-rule operates only on alignable types, by identifying the (equiperipheral) assumptions on which they depend, and replacing them by a *new* assumption of the same type and bearing the same peripherality to the resulting declarative unit. An important effect of this definition is, that any continuation of derivation that was possible for each of the coordinated declarative

² In this context, whenever dependence on assumptions is mentioned, it is understood as undischarged assumptions.

units (representing pseudo-constituent phrases) is possible for the declarative unit resulting by the revised *ncc*-rule (representing the coordinated pseudo-constituent). In particular, any discharge of an assumption (within the given continuation) is now replaced by the assumption that results from identifying the original assumptions, when an arrow-introduction rule is applied. In the general case of dependence on more than one assumption (presented below), alignment of assumptions has the same effect but for more general dependency on assumptions.

$$\begin{aligned}
 (ncc) \quad & \frac{U_1 \triangleright \tau, U_2 \triangleright \kappa, U_3 \triangleright \tau}{(U_1 U_2 U_3) \triangleright \tau} \quad \tau \text{ assumption} - \text{independent} \\
 (lncc) \quad & \frac{U_1[A]_1 \triangleright \tau, U_2 \triangleright \kappa, U_3[A]_2 \triangleright \tau}{(U_1 U_2 U_3[A]_{1 \equiv 2}) \triangleright \tau} \\
 (rncc) \quad & \frac{[A]_1 U_1 \triangleright \tau, U_2 \triangleright \kappa, [A]_2 U_3 \triangleright \tau}{([A]_{1 \equiv 2} U_1 U_2 U_3) \triangleright \tau}
 \end{aligned}$$

Fig. 6. The revised *ncc*-rules for \mathcal{L}

The simplified revised *ncc*-rule (under the single-gap assumption) constitutes now of three separate rules, presented in Figure 6. The notational convention adopted is, that assumptions indexed by a single index are present in the original resources, while these indexed by an expression containing ‘ \equiv ’ are generated in the proof itself. An assumption $[\tau]_{i \equiv j}$ is the result of identifying the two assumptions $[\tau]_i$ and $[\tau]_j$. We adhere here to this simplified notation because we do not consider examples with iterated application of the revised rule. In a more general setup, where such iterations are considered, a better notation would be identifying $[\tau]_I$ and $[\tau]_J$, for I, J some finite index sets, to produce $[\tau]_{I \cup J}$; then, $[\tau]_{i \equiv j}$ would be merely $[\tau]_{\{i, j\}}$. Here we stick with the simpler notation. As already mentioned, the proposed rule transcends the usual type-logical framework. In a more general framework, it might be sensible to detach the assumption-identification from uses by the revised *ncc*-rule, and give it an autonomous status, associated with the semantic operation of substitution for (free) variables in λ -terms, as shown below. I am currently working on such a more general framework. At this point, however, I have no worked-out additional applications of this operation except within coordination, so it is left as part of the revised coordination rule.

There is no semantic reason for disallowing coordination, say, of $np \rightarrow s$ and $s \leftarrow np$, both having predicates of the form $\lambda x[p(x)]$ as their associated meanings. However, non-alignability would allow derivation of

(4) (*) A man who - slept and Mary kissed - ,

which is syntactically bad³. Note also, that the suggested rule is in accordance with the general point of view of Lambek-calculi as resource-management logics. Here, assumptions are recognized as special resources, and are given their own resource-manipulation rules. This resource-manipulation might be seen as a controlled-modulation (as alluded to in [10]) regarding a limited amount of the contraction structural-rule, absent in its full generality from \mathcal{L} , in the context of NCC .

$$\begin{array}{c}
 \text{Mary} \quad \frac{\text{kissed} \quad -}{(np \rightarrow s) \leftarrow np \quad [np]_1} \leftarrow E \quad \text{and} \quad \frac{\text{hugged} \quad -}{(np \rightarrow s) \leftarrow np \quad [np]_2} \leftarrow E \\
 \frac{np}{np} \quad \frac{np \rightarrow s}{s} \rightarrow E \quad \frac{s}{s \leftarrow np} \leftarrow I_{1=2} \quad \frac{John}{np} \leftarrow E \\
 \hline
 \frac{s \quad [np]_{1=2}}{s} \text{ } l_{ncc}
 \end{array}$$

Fig. 7. Left-ncc derivation with revised ncc-rule

Let us inspect some examples of the application of the revised *ncc*-rule. First, note that derivation of constituent-coordination remains valid, as constituent-types are always alignable, not depending on assumptions. Figure 7 presents the derivation for (2). The *lncc*-rule discharges the two right-peripheral assumptions $[np]_i, i = 1, 2$, and the resulting s depends on the newly-generated assumption (right-peripheral to the whole *NCC*) $[np]_{1=2}$, the latter is then discharged by the arrow-introduction rule. Note that while the conjoined phrases are indeed pseudo-constituents (having an *np*-gap each in the object position), the categories conjoined are s (though depending still on assumptions).

A regularity observed in the above example is, that immediately preceding an application of the revised *ncc*-rule is always an application of an arrow-introduction rule, which discharges the newly introduced assumption $[\tau]_{i=j}$. This suggest a somewhat different formulation of the coordination rule, by which the arrow-introduction is absorbed into the coordination rule. Such a formulation would yield, for the right *NCC*,

$$(r_{ncc}') \quad \frac{[A]_1 U_1 \triangleright \tau, \quad U_2 \triangleright \kappa, \quad [A]_2 U_3 \triangleright \tau}{(U_1 U_2 U_3) \triangleright ([A]_{1=2} \rightarrow \tau)}$$

However, anticipating future applications of the various “components” of the coordination rule, we find it preferable not load too much on this rule, and keep the whole treatment more modular and separable in future. Thus, we stick here with formulation proposed before. Furthermore, as also can be seen from the example derivations, proof-generated assumptions need to be exempt from the “Prawitz normal-form” requirement [13], and functor-categories generated by

³ There are known exceptions, e.g. *who(m) did Mary give - a present and kiss -*.

discharging a combined-assumption generated by the revised *ncc*-rule can be immediately applied to arguments, without the danger of spurious ambiguity.

Turning to the general case of alignable categories, with any number of equal-type, equal-peripherality assumptions, we refer to the revised *ncc*-rule as *ncch* (*ncc* with hypothetical types). The rule is schematic over the numbers of assumptions. It is presented in Figure 8. All superscripts are for distinguishing types; all subscripts, assumed pairwise different, are for numbering assumptions. Note that the special-case rule presented before is obtained, respectively, for the

$$(ncch) \frac{[A^1]_{i_1} \dots [A^n]_{i_n} U_1 [B^1]_{k_1} \dots [B^n]_{k_m} \triangleright \tau, \quad U_2 \triangleright \kappa, \quad [A^1]_{j_1} \dots [A^n]_{j_n} U_3 [B^1]_{l_1} \dots [B^n]_{l_m} \triangleright \tau}{([A^1]_{i_1} \equiv j_1 \dots [A^n]_{i_n} \equiv j_n \quad U_1 U_2 U_3 [B^1]_{k_1} \equiv l_1 \dots [B^n]_{k_m} \equiv l_m) \triangleright \tau}$$

Fig. 8. The revised *ncc*-rule for general alignable types

three cases $n = m = 0$, $n = 0 \wedge m = 1$ and $n = 1 \wedge m = 0$. Note also how the rule identifies corresponding pairs of equi-typed, equi-peripheral assumptions, to generate for each pair a combined assumption with the same type and peripherality. This rule is *not* an iteration of the basic revised *ncc*-rule, as there is only one instance of κ involved in its application. An iterative application of the basic rule would need a number of instances of κ as the number of iterations, as usual in resource conscious logics.

The revised derivation for a right-*ncc* is shown in Figure 9. The derivations henceforth are displayed as a collection of sub-derivations (separated by dotted lines) for ease of formatting.

To see further the effect of the revised *ncc*-rule, consider a derivation of a sentence containing “split constituent” *ncc* (in the nomenclature of [3]):

(5) Mary drove [to Chicago] yesterday and Detroit today.

The square brackets delimit the constituent, while the underline indicates the coordinated phrases. The derivation is presented in Figure 10. We use the derived rule \Rightarrow_C for shortening the derivation. As a more complicated example, in which the revised *ncc*-rule is applied twice, consider the derivation in Figure 11 of a two-sided *NCC* in

(6) Mary gave and Sue sold a book to John and a record to Bill.

An alternative implementation of the idea of identifying assumptions, leading to a late discharge within a pure natural-deduction framework, can be obtained by using a multi-modal logic with a “bracketing modality” (see [10], Ch. 4, and the references there), that would introduce another composition operation, that distributes over products within a bracketed context. This would also relax locally the absence of a contracting structural rule. We do not pursue here further this possibility.

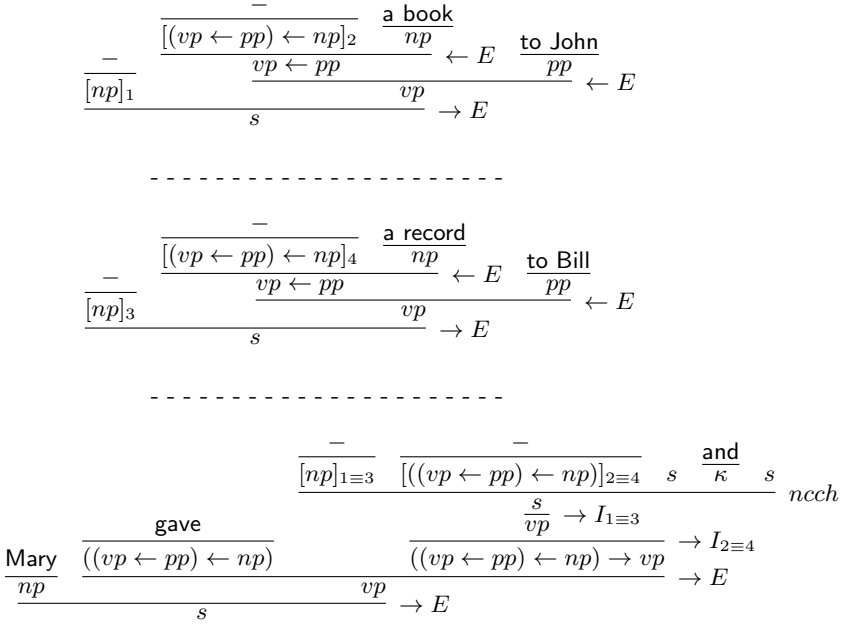


Fig. 9. Right-ncc derivation with revised ncc-rule

Our scheme blocks the derivations in [14], of sentences like

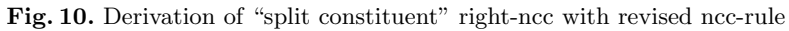
(7) (*) A man who - slept and Mary kissed John.

Any attempt to derive this sentence by applying the revised *ncc*-rule will have to conjoin the phrase - slept of category *s* depending on a left-peripheral assumption, and the phrase Mary kissed John also of category *s*, but not depending on any assumptions. Thus, the alignability condition is violated. At this point, there is no characterization of exactly which over-generations are blocked. However, in view of the interesting non-blocking of the following bad sentence⁴, derivable also by the original *ncc*-rule, it is clear that the current rule is still a coarse approximation to the ultimate coordination rule within type-logical grammar.

(8) (*) The mother of and Bill thought John arrived.

The derivation is shown in Figure 12. Clearly, alignability is not strong enough to block such examples, which depend on *how assumptions are used*, not only where they are located. Thus, a finer equivalence relation among assumptions is needed to block more over-generated phrases; once a better approximation is obtained, an exact characterization should be sought for any remaining non-blockings.

⁴ attributed in [3] to Paul Dekker.



Turning to the semantics of the revised *ncc*-rule (under the assumption of one peripheral gap only), suppose the meanings of the two conjoined types are $\alpha(x_1)$ and $\beta(x_2)$, where x_1, x_2 are the meanings of the undischarged assumptions the respective coordinated types depend upon, say $[...]_1$ and $[...]_2$. Then, the meaning of the generated assumption $[...]_{1\equiv 2}$ is x , where x is *fresh* variable, and the meaning of the resulting conjoined type is $\alpha(x_1) \sqcap \beta(x_2)[x_1 := x, x_2 := x]$, amounting to $\alpha(x) \sqcap \beta(x)$. Here $\alpha[x := y]$ means the substitution of y for all free occurrences of x in α . This semantic rule captures the intention of identifying the two assumptions semantically. After applying an arrow-introduction rule, say $\leftarrow I_{1\equiv 2}$, the obtained meaning of the result becomes $\lambda x.(\alpha(x) \sqcap \beta(x))$, the exact meaning stipulated by Steedman as holding by definition. The substitution of a fresh variable for all occurrences two different free variables can be seen as the extension of the Curry-Howard correspondence to the syntactic operation manifesting itself in the revised *ncc*-rule.

To see the relative scope of quantification and conjunction, consider the variant (9) of (2).

$$\begin{array}{c}
\dfrac{\dfrac{-}{[np]_5} \quad \dfrac{\dfrac{\dfrac{-}{[(vp \leftarrow pp) \leftarrow np]_1}}{vp \leftarrow pp} \quad \dfrac{\text{a book}}{np} \leftarrow E}{vp} \leftarrow E}{s} \quad \dfrac{\text{to John}}{pp} \leftarrow E}{vp} \leftarrow E \\
\\
\dots \\
\dfrac{\dfrac{-}{[np]_6} \quad \dfrac{\dfrac{\dfrac{-}{[(vp \leftarrow pp) \leftarrow np]_2}}{vp \leftarrow pp} \quad \dfrac{\text{a record}}{np} \leftarrow E}{vp} \leftarrow E}{s} \quad \dfrac{\text{to Bill}}{pp} \leftarrow E}{vp} \leftarrow E \\
\\
\dots \\
\dfrac{\dfrac{-}{[np]_{5=6}} \quad \dfrac{\dfrac{-}{[(vp \leftarrow pp) \leftarrow np]_{1=2}} \quad s \quad \dfrac{\text{and}}{\kappa} \quad s}{\dfrac{s}{vp} \rightarrow I_{5=6}}} \quad ncch}{((vp \leftarrow pp) \leftarrow np) \rightarrow vp)} \rightarrow I_{1=2} \\
\\
\dots \\
\dfrac{\text{Mary}}{np} \quad \dfrac{\text{gave} \quad \dfrac{(vp \leftarrow pp) \leftarrow np \quad [((vp \leftarrow pp) \leftarrow np) \rightarrow vp]_3}{vp}}{s}} \rightarrow E \\
\\
\dots \\
\dfrac{\text{Sue}}{np} \quad \dfrac{\text{sold} \quad \dfrac{(vp \leftarrow pp) \leftarrow np \quad [((vp \leftarrow pp) \leftarrow np) \rightarrow vp]_4}{vp}}{s}} \rightarrow E \\
\\
\dots \\
\dfrac{s \quad \dfrac{\text{and}}{\kappa} \quad s \quad \dfrac{\dfrac{-}{[((vp \leftarrow pp) \leftarrow np) \rightarrow vp]_{3=4}}}{\dfrac{s}{s \leftarrow ((vp \leftarrow pp) \leftarrow np) \rightarrow vp}}} \quad ncch}{s \leftarrow (((vp \leftarrow pp) \leftarrow np) \rightarrow vp)} \rightarrow I_{3=4} \\
\\
\dots \\
\dfrac{s \leftarrow (((vp \leftarrow pp) \leftarrow np) \rightarrow vp) \quad (((vp \leftarrow pp) \leftarrow np) \rightarrow vp)}{s} \leftarrow E
\end{array}$$

Fig. 11. Derivation of two-sided ncc with revised ncc-rule

$$\begin{array}{c}
\frac{\frac{\text{the mother of}}{np \leftarrow np} \quad \frac{\overline{\quad}}{[np]_2}}{np} \leftarrow E \quad \frac{\overline{\quad}}{[np \rightarrow s]_1} \rightarrow E \\
\hline
s
\end{array}$$

$$\begin{array}{c}
\frac{\text{Bill thoght} \quad \frac{\overline{\quad}}{[np]_4} \quad \frac{\overline{\quad}}{[np \rightarrow s]_3}}{\frac{s \leftarrow s}{s} \quad s} \leftarrow E \rightarrow E
\end{array}$$

$$\begin{array}{c}
\frac{s \quad \frac{\text{and}}{\kappa} \quad s \quad \frac{[np]_{2 \equiv 4} \quad [np \rightarrow s]_{1 \equiv 3}}{s} \quad ncch}{\frac{s \leftarrow (np \rightarrow s)}{((s \leftarrow (np \rightarrow s)) \leftarrow np)} \leftarrow I_{1 \equiv 3}} \leftarrow I_{2 \equiv 4} \quad \frac{\text{John}}{np} \leftarrow E \quad \frac{\text{arrived}}{(np \rightarrow s)} \leftarrow E \\
\hline
s
\end{array}$$

Fig. 12. Derivation of Dekker's counterexample with revised ncc-rule

(9) Mary kissed and Sue hugged a boy

Interpreting a boy as the usual generalized quantifier $\lambda P[\exists x(\text{boy}(x) \wedge P(x))]$, and resorting to a derivation similar to that in Figure 7 (but with an additional type-raising of the object np), we get the meaning of the coordinated phrase as follows:

$$\exists x(\text{boy}(x) \wedge \text{kiss}(\text{Mary}, x) \wedge \text{hug}(\text{sue}, x))$$

Thus, the existential quantifier takes scope over conjunction. Similarly, for the Geach [5] sentence (10)

(10) Every girl loves, and every boy detests, some saxophonist

We get the reading in which the existential quantifier takes scope over the two universal quantifiers (as well as over the conjunction). The reading where the existential quantifier has the lowest reading can also be obtained, as mentioned before, by using scope modalities. However, the bad reading, where two, possibly different saxophonists, one related to the girls and and the other related to the boys, are involved is not generable due to gaps identification. In [16], Steedman suggest to abandon the GQ interpretation of indefinites in order to block the bad reading of (10), resorting to some referential interpretation. This is not needed in the current approach to *NCC*.

We note while passing that a treatment of *NCC* using open formulae is hinted at in [6]. There, the main argument is about the role of variables in the semantics, and the main issue is pronoun binding. In a brief consideration of an *NCC* example, Jacobson hints towards a conjunction of formulae with

free-variables, but nothing relates it to hypothetical reasoning. The semantics of conjunction is stipulated similarly to Steedman's.

$$\begin{array}{c}
 \frac{\frac{\frac{[np]_1}{-}}{s} \quad \frac{\frac{\frac{[(vp \leftarrow pp) \leftarrow np]_3}{-} \quad \frac{\text{a book}}{np}}{vp \leftarrow pp} \leftarrow E \quad \frac{\text{to John}}{pp} \leftarrow E}{vp} \rightarrow E}{s} \\
 \\
 \text{-----} \\
 \frac{\frac{\frac{[np]_2}{-}}{s} \quad \frac{\frac{\frac{[(vp \leftarrow pp) \leftarrow np]_4}{-} \quad \frac{\text{a record}}{np}}{vp \leftarrow pp} \leftarrow E \quad \frac{\text{to Bill}}{pp} \leftarrow E}{vp} \rightarrow E}{s} \\
 \\
 \text{-----} \\
 \frac{\text{Mary} \quad \frac{\text{gave} \quad \frac{(vp \leftarrow pp) \leftarrow np}{np} \quad \frac{((vp \leftarrow pp) \leftarrow np) \rightarrow (np \rightarrow s)}{np \rightarrow s} \rightarrow E}{np \rightarrow s} \rightarrow E \quad \frac{\frac{[np]_{1 \equiv 2} \quad [(vp \leftarrow pp) \leftarrow np]_{3 \equiv 4} \quad s \quad \frac{\text{and}}{\kappa} \quad s}{np \rightarrow s} \rightarrow I_{1 \equiv 2}}{ncch}}{s}
 \end{array}$$

Fig. 13. An alternative right-ncc derivation with the *ncch*-rule

4 Basic Non-constituent Coordination

Consider the derivation in Figure 13. It uses two left-peripheral assumptions, hence *ncch* is indeed needed. Note that the coordinated type is *s*. This is not accidental! By resorting to *ncch*, one can always introduce sufficiently many assumptions, so as the pseudo-constituents coordinated are open sentences (with a semantics containing free variables for all assumptions). There is one exception, though, for *np*-coordination, having a non-distributive predication, like *Mary and John met*. For a recent account of boolean semantics for *np*-coordination (both collective and distributive) see [17]. We may, therefore, conclude, that nothing is lost if the logic *forbids ncc*-coordination of functional categories, restricting it to base categories only (*s* and *np* in the current version of \mathcal{L} employed here).

Basic non-constituent coordination thesis: *Only (and all) base-categories can be coordinated.*

There is no distinction anymore between constituent and non-constituent coordination, and *ncch* applies equally to both, restoring the attractive similarity between the derivations of (1) and (2).

In [6] there is a short discussion of the role of variables in model-theoretic semantics. Jacobson draws a distinction between a semantics in which “variables do real work”, and the case in which they do not. The distinction is drawn, though, based on two operations supported by variables: being bound (or abstracted) and denoting a value via variable-assignments. We would like to emphasize that in our extension of \mathcal{L} with *ncch*, variables are meanings of resources, and the operations on them correspond to resource manipulation allowed by the substructural logic. Thus, the identification of different variables via substitution, corresponding to assumption-identification by *ncch*, should render the current semantics as belonging to the sort in which “variables do real work”. The real work is obtained as an extension of the traditional Curry-Howard correspondence.

5 Conclusions

In this paper, we presented a new approach to the treatment of non-constituent coordination in type-logical grammar. The approach is based on a rule, deviating from the standard rule landscape of introduction, elimination and structural control rules. It is based on view of assumptions used for hypothetical reasoning as resources amenable to their own resource manipulation; in this case, coalescing different assumptions, of equal type and peripherality, to one, in the context of the coordination rule. This leads to an ability to coordinate types depending on undischarged assumptions, contrary to current practice in the literature. Ultimately, this leads to the so-called basic view of *NCC*, restricting the polymorphic type in the coordination rule to range over basic categories only.

There is room for much further research, in which I am currently engaged.

- There is a good reason to disassociate the assumptions identification from the coordination rule. As this is a syntactic counterpart of substitution in λ -terms, extending the Curry-Howard correspondence, it may have additional applications.
- There is a need for refining the alignability condition used in the current proposal, thereby obtaining a finer equivalence among assumptions for hypothetical reasoning. This will have an improved blocking of over-generated phrases.

One direction to look at is in [1], where a notion of *parallelism* between coordinated phrases is proposed, based on having the same structure of *constituency dependency*. The latter reflects the way assumptions used as a major premiss in applications of row-elimination rules are discharged.

- In [4], Emms shows that having general polymorphism for \forall -types (though with empty untededents in declarative units) leads to undecidability. It should be interesting to check whether confining polymorphism to basic types only affects decidability.
- Additional instances of *NCC* should be investigated. One such topic is *NCC* using subject-control verbs. Thus, we would like to be able to generate a proper interpretation of (11) while we would like to block (12).
 - (11) John urged Bill and persuaded Mary to go
 - (12) (*) John persuaded and promised Mary to go

Acknowledgements. This research started under grant B 62-443 from Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO). I wish to thank NWO for their support. I thank Michael Moortgat for drawing me to type-logical grammars, and Jan van Eijck, Rani Nelken, Mark Steedman and Yoad Winter for various discussions. The anonymous referees are thanked for constructive criticism that led to a considerable improvement of the presentation. In the Technion, this work was partially supported by the fund for the promotion of research at the Technion.

References

1. Barry, G., and M. Pickering. Dependency and constituency in Categorical Grammar. In Lecomte, A., editor, *Word order in Categorical Grammar*. DYANA project workshop in Clermont-Ferrand, 1992.
2. Carpenter, B. A deductive account of scope. In *Proc. of the 13th west coast conference on formal linguistics*, San-Diego, CA, 1994. CSLI publications. Extended version submitted to *Linguistics and Philosophy*.
3. Dowty, D. Non-constituent coordination, wrapping, and multimodal categorical grammars. In *Proc. of the international congress on logic, methodology, and philosophy*, Florence, Italy, August, 1996. Kluwer.
4. Emms, M. An undecidability result for polymorphic Lambek Calculus. In Dekker, P., and M. Stokhof, editors, *Proceedings of Tenth Amsterdam Colloquium*. ILLC, 1995.
5. Geach, P. A program for syntax. In Davidson, D., and G. Herman, editors, *Semantics of Natural Language*. Reidel, Dordrecht, 1972.
6. Jacobson, P. The locality of interpretation: the case of binding and coordination. In *Proceedings of the 6th conference on semantics in Linguistic Theory (SALT)*. Cornell working papers in Linguistic, 1996.
7. Keenan, E., and L. Faltz. *Boolean semantics for natural language*. Reidel, Dordrecht, 1985.
8. Lambek, J. The mathematics of sentence structure. *American mathematical monthly*, 65:154 – 170, 1958.
9. Milward, D. Non-constituent coordination: Theory and practice. In *Proc. of the 15th international conference on Computational linguistics (COLING94)*, pages 935–941, Kyoto, Japan, 1994.

10. Moortgat, M. Categorical type logics. In Johan van Benthem and Alice ter Muelen, editors, *Handbook of Logic and Language*. Elsevier, 1997.
11. Morrill, G. V. *Type Logical Grammar: Categorical Logic of Signs*. Kluwer Academic Publishers, 1994.
12. Partee, B., and M. Rooth. Generalized conjunction and type ambiguity. In Rainer Bäuerle, Christoph Schwartze, and Arnim von Stechow, editors, *Meaning, use and interpretation of language*. De Gruyter, Berlin, 1983.
13. Prawitz, D. *Natural deduction*. Alqvist and Wiksell, Uppsala, 1965.
14. Steedman, M. Categorical grammar (tutorial overview). *Lingua*, 90:221–258, 1993.
15. Steedman, M. *Surface Structure and Interpretation*. MIT Press, Linguistic Inquiry monographs 30, 1996.
16. Steedman, M. Alternating quantifier scope in CCG. *To appear*, 1999.
17. Winter, Y. A unified semantic treatment of singular np coordination. *Linguistics and Philosophy*, 19:337–391, 1996. See also forthcoming (1998) Ph.D thesis.

Computational and Structural Aspects of Openly Specified Type Hierarchies

Stephen J. Hegner

Umeå University
Department of Computing Science
S-901 87 Umeå, Sweden
hegner@cs.umu.se
<http://www.cs.umu.se/~hegner>

Abstract. One may identify two main approaches to the description of type hierarchies. In *total specification*, a unique hierarchy is described. In *open specification*, a set of constraints identifies properties of the hierarchy, without providing a complete description. Open specification provides increased expressive power, but at the expense of considerable computational complexity, with essential tasks being NP-complete or NP-hard. In this work, a formal study of the structural and computational aspects of open specification is conducted, so that a better understanding of how techniques may be developed to address these complexities. In addition, a technique is presented, based upon Horn clauses, which allows one to obtain answers to certain types of queries on open specifications very efficiently.

1 Introduction

In recent years, grammatical formalisms based upon constraint-based parsing within the context of typed feature structures have become central within computational linguistics, the most prominent undoubtedly being HPSG [22]. As a result, numerous computational frameworks specifically designed for constraint-based reasoning on typed feature logics have emerged, among them ALE [5], TFS [24], and CUF [7]. Likewise, systems for representing and managing lexical information in a hierarchical fashion have appeared in recent years [3]. To function efficiently, all of these frameworks must first and foremost be capable of managing the associated type hierarchy effectively.¹

¹ By “type hierarchy,” we mean the static hierarchy of (parameterless) types which underlies the typing mechanism of the feature structures. We do not include the recursively specified types which also form an integral part of grammars which are specified using such systems. In CUF [7], the static types are called *types*, and the (often parameterized) recursive types are called *sorts*, but there is no universal agreement, and other terminology, or even the opposite terminology, is often used. Pollard and Sag [22], for example, use the term *sort* to characterize that which we call type.

All type hierarchies share some basic properties, such as the ability to express inheritance. However, beyond that, there is relatively little agreement as to which properties are appropriate. It seems clear that these differences arise not because some systems embody the “correct” formalism while others do not, but rather because each system places distinct expectations upon the hierarchy. As with all forms of knowledge representation [19], there is a fundamental tradeoff between expressiveness and tractability in formalization of type hierarchies. In this paper, we examine aspects of the tradeoff which occurs when one moves from total specification, in which the entire hierarchy is specified explicitly, to constraint-based open specification² in which the actual hierarchy is not completely specified, but rather may be of the models of a set of constraints.

It is fair to say that total specification is used far more frequently than is open specification. Indeed, the only system which we know of which supports open specification is CUF.³ It appears to be the case, at least in part, that open specification has been avoided because the computational overhead is perceived to be too high. This perception is reinforced by the fact that the question of model existence in such a context is NP-complete [14]. Nonetheless, we feel that open specification can be a useful tool in some contexts, and it is therefore important to understand more about its representational aspects and computational complexity. In this paper, we take some steps towards this end.

The paper is organized as follows. In Sec. 2, a brief summary of the representational aspects of completely specified hierarchies is presented. Such a summary is important because even within that context, there is a critical distinction between distributive and nondistributive hierarchies which must be understood to appreciate fully the various aspects of open specification, since the latter is carried out in a distributive context. In Sec. 3, the basic ideas and structural aspects of open specification are presented. In Sec. 4, representations which are essential for the process of computing solutions to an open specification are developed. Finally, in Sec. 5, some efficient techniques for identifying properties of models in open specifications, based upon Horn clauses, are developed,

This paper may be viewed as a complement to [14]. In that work, the focus was largely upon developing the machinery necessary to show the satisfaction problem to be NP-complete. In this paper, we focus more on structural and algorithmic issues.

² In [21, Sec. 3.1], the term *open specification* is used in a different way, in a discussion contrasting ALE to Troll [12]. We see this terminological overloading as a non-issue, provided authors are careful to indicate which definitions are in force in their writings.

³ In his work Ait-Kaci [1], [2] has used a crown construction to complete hierarchies which may not have all glb's. However, this technique always associates a unique completion with a specification. Thus, we regard it as a means of extracting a total specification from an incomplete specification, rather than one of open modelling.

2 The Rôle of Distributivity

This paper is largely about the open specification of distributive hierarchies. However, to put that work in perspective, it is first necessary to identify the reasons why distributivity is important. In this section, we present a brief overview of the rationale for requiring distributivity in ordinary, complete hierarchies.

2.1 Bounded and Distributive Lattices

A lattice is *bounded* if it has a greatest element \top and a least element \perp . It is always assumed that \top and \perp are distinct, so under this definition, a lattice must have at least two different elements. As a notational convention, a boldface symbol (e.g., \mathbf{L}) will denote the entire algebraic structure, while the roman symbol L will denote just the underlying set of elements. Thus, we write $\mathbf{L} = (L, \vee, \wedge, \top, \perp)$. A lattice is *distributive* [13, p. 30] if it satisfies $(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$ for all elements a , b , and c .

2.2 General Semantics for Total Type Hierarchies

The syntactic component of a type hierarchy is just a finite bounded lattice (not necessarily distributive). However, a type hierarchy has a semantics as well, which identifies a collection of objects, together with a specification of which objects belong to which types. More formally, let $\mathbf{L} = (L, \vee, \wedge, \top, \perp)$ be a finite bounded lattice. A *type semantics* for \mathbf{L} is a pair $\mathbf{S} = (\mathcal{U}, \mathcal{J})$, in which \mathcal{U} is a nonempty set, called the *universe of objects*, and $\mathcal{J} : L \rightarrow 2^{\mathcal{U}}$ is a function which associates a subset of \mathcal{U} to each type in L , subject to the following conditions that $\mathcal{J}(\top) = \mathcal{U}$, $\mathcal{J}(\perp) = \emptyset$, and for $\tau_1, \tau_2 \in T$, $\tau_1 \leq \tau_2$ implies $\mathcal{J}(\tau_1) \subseteq \mathcal{J}(\tau_2)$. The last condition expresses the critical concept of *inheritance*: if $\tau_1 \leq \tau_2$, then every instance of τ_1 is also an instance of τ_2 , and so every property which applies to an object of type τ_2 also applies to (i.e., is *inherited by*) every object of type τ_1 .

The semantics $(\mathcal{U}, \mathcal{J})$ *separates* $\{\tau_1, \tau_2\} \in L$ if $\mathcal{J}(\tau_1) \neq \mathcal{J}(\tau_2)$, and is *totally separating* if it separates every pair $\{\tau_1, \tau_2\} \in L$. Note that $\{\top, \perp\}$ must be separated by any semantics $(\mathcal{U}, \mathcal{J})$, since \mathcal{U} is required to be nonempty.

Formally, a type hierarchy is a pair $\mathfrak{H} = (\mathbf{L}, (\mathcal{U}, \mathcal{J}))$ in which \mathbf{L} is a finite bounded lattice and $(\mathcal{U}, \mathcal{J})$ is a semantics for \mathbf{L} .

2.3 Natural Meet Semantics

The fundamental operation in constraint-based parsing strategies is *unification*; the unification of two objects x and y results in an object $x \sqcup y$ which has all of the attributes of both x and y . In the context of typed unification, this means in particular that $x \sqcup y$ is of both the type of x and the type of y . Thus, if x is known to be of type τ_x , and y of type τ_y , then $x \sqcup y$ must be of type $\tau_x \wedge \tau_y$. It follows that a type hierarchy which is used for constraint-based parsing must have the

following property: A type hierarchy $\mathfrak{H} = (\mathbf{L}, (\mathfrak{U}, \mathfrak{J}))$ satisfies the *natural meet semantics* if the following condition is satisfied for every $\tau_1, \tau_2 \in L$:

$$\mathfrak{J}(\tau_1 \wedge \tau_2) = \mathfrak{J}(\tau_1) \cap \mathfrak{J}(\tau_2) \quad (\text{nat-}\wedge)$$

A lattice \mathbf{L} is said to *admit natural meet semantics* if there is a semantics $(\mathfrak{U}, \mathfrak{J})$ for \mathbf{L} which satisfies condition (nat- \wedge) above.

2.4 Existence of Natural Meet Semantics

Every bounded lattice $\mathbf{L} = (L, \vee, \wedge, \top, \perp)$ admits a totally separating separating natural meet semantics.

Proof. For each $\tau \in L \setminus \perp$, associate a distinct element x_τ , and then put $\mathfrak{U} = \{x_\tau \mid \tau \in L\}$. Define $\mathfrak{J} : L \rightarrow \mathfrak{U}$ by $\tau \mapsto \{x_\sigma \mid \sigma \leq \tau\}$. It is easy to see that this semantics satisfies the condition (sem- \wedge), and it is totally separating by construction. \square

2.5 The Role of Distributivity

In the systems ALE [5], TFS [24], and the ACQUILEX LKB [6], among others, the type hierarchies are not required to be distributive. In [4, pp. 15-17], Carpenter argues that type hierarchies should not be distributive. On the other hand, the CUF system [7] requires a distributive hierarchy. We shall now attempt to sort out how these apparently disparate points of view can exist.

Unification itself makes no use of the join operation; as outlined above, only the meet operation is used. Even though a bounded finite meet semilattice will necessarily have joins of any elements [13, Ch. 1, Sec. 3, Lem. 14], these joins are not used in the unification process, and so it is unnecessary to assign any computationally significant semantics to them. In short, only the natural meet semantics is relevant. Since the concept of distributivity can make sense only in a context in which there is a meaningful join as well as meet, distributivity plays no formal rôle in these contexts. As shown in 2.4, it is always possible to assign a complete meet semantics to a finite bounded lattice, regardless of any further properties such as distributivity or modularity.

CUF, on the other hand, was designed to support *disjunctive unification* [8], in which alternative parses are support via a meaningful semantics on the join operation. This indeed requires a distributive lattice.

2.6 Natural Join Semantics and Natural Semantics

A type hierarchy $\mathfrak{H} = (\mathbf{L}, (\mathfrak{U}, \mathfrak{J}))$ satisfies the *natural join semantics* if the following condition is satisfied for every $\tau_1, \tau_2 \in L$:

$$\mathfrak{J}(\tau_1 \vee \tau_2) = \mathfrak{J}(\tau_1) \cup \mathfrak{J}(\tau_2) \quad (\text{nat-}\vee)$$

The type hierarchy \mathfrak{H} is said to satisfy the *natural semantics* if it satisfies both condition (sem- \wedge) and (sem- \vee). Similarly, a lattice \mathbf{L} is said to *admit natural semantics* if there is a semantics for \mathbf{L} which satisfies both (sem- \wedge) and (sem- \vee).

The definitions of *separating* and *totally separating* natural semantics are analogous to those which apply to meet semantics.

2.7 Birkhoff-Stone Representation of Distributive Lattices

A lattice \mathbf{L} is called a *ring of sets* if there is a set S (called the *basis*) with the property that every $x \in L$ is a subset of S , and, furthermore, that for every $x, y \in L$, $x \vee y = x \cup y$ and $x \wedge y = x \cap y$. In other words, join is union and meet is intersection. In the case of a bounded lattice, it suffices to take $\top = S$. Such a lattice has a “built-in” semantics; let S be the universe of objects, with the set of objects associated with an element $x \in L$ just x itself.

The representation theorem of Birkhoff and Stone [13, Ch. 2, Sec. 1, Thm. 19] states that *a lattice is distributive iff it is isomorphic to a ring of sets*. It is furthermore possible to require that the ring be *nonredundant*, in the precise sense that if $s_1, s_2 \in S$, then there is some $x \in L$ which contains one of $\{s_1, s_2\}$, but not both. (Otherwise, one of $\{s_1, s_2\}$ could be removed, with the resulting lattice isomorphic to the original one.)

2.8 Existence and Algorithmic Aspects

Let $\mathbf{L} = (L, \vee, \wedge, \top, \perp)$ be an arbitrary finite bounded lattice, and let $n = \text{Card}(L)$ denote the cardinality of L .

- (a) \mathbf{L} admits a totally separating natural semantics iff it is distributive. This question is decidable in time $\Theta(n^3)$.
- (b) It is decidable in time $\Theta(n^3)$ whether or not \mathbf{L} admits a natural semantics.
- (c) If \mathbf{L} admits a natural semantics, it may be constructed in time $\Theta(n^3)$.

Proof. The characterization is a consequence of the Birkhoff-Stone representation theorem. The $\Theta(n^3)$ algorithm arises from the fact that the distributive law involves triples of elements; it is only necessary to check each such triple in turn. The question of establishing a natural semantics involves construction of an appropriate quotient lattice via the congruence defined by nondistributive components. The details are not presented in this paper. The important point, however, is that the construction may be carried out in deterministic polynomial time. \square

3 Open Specification of Type Hierarchies

In open specification, a constraint-based specification replaces a complete description of the hierarchy. At least one existing system, CUF [7], has taken this approach. In this section, some basic issues surrounding such specifications are examined, including condition for existence of a model, size of models of models, and characterization of canonical models. Because the principal interest in such a framework lies within the domain of distributive hierarchies, attention has been restricted to that case.

3.1 Augmentation and Interpretations

A set of *clean types* is one which contains neither \top nor \perp . For such a set, define $\text{Aug}_\perp(P) = P \cup \{\perp\}$, $\text{Aug}_\top(P) = P \cup \{\top\}$, and $\text{Aug}(P) = P \cup \{\perp, \top\}$. An *interpretation* of P is a pair $I = (\mathbf{L}, f)$ in which $\mathbf{L} = (L, \vee, \wedge, \top, \perp)$ is a bounded distributive lattice and $f : \text{Aug}(P) \rightarrow L$ is a function which is subject to the conditions that $f(\perp) = \perp$ and $f(\top) = \top$. The collection of all interpretations of P is denoted $\text{Interp}(P)$.

3.2 Open Specifications

Let P be any finite set of clean types. The system of *constraints over P* is defined to be the smallest set $\text{Constraints}(P)$ satisfying the following conditions.

- (c- \leq): If $\tau_1 \in \text{Aug}_\top(P)$, $\tau_2 \in \text{Aug}_\perp(P)$, then $(\tau_1 \leq \tau_2) \in \text{Constraints}(P)$.
- (c- \wedge): If $\tau \in \text{Aug}(P)$ and $S \subseteq P$ is nonempty, then $(\bigwedge S = \tau) \in \text{Constraints}(P)$.
- (c- \vee): If $\tau \in \text{Aug}(P)$ and $S \subseteq P$ is nonempty, then $(\bigvee S = \tau) \in \text{Constraints}(P)$.
- (c- \neq): If $\tau_1 \in P$, $\tau_2 \in \text{Aug}(P)$, then $(\tau_1 \neq \tau_2) \in \text{Constraints}(P)$.
- (c-Atom): If $\tau \in P$, then $\text{Atom}(\tau) \in \text{Constraints}(P)$.

An interpretation $I = (\mathbf{L}, f)$ *satisfies* the constraint $\varphi \in \text{Constraints}(P)$, written $I \models \varphi$, if the appropriate rule given below is satisfied.

- (sat- \leq): $I \models (\tau_1 \leq \tau_2)$ iff $f(\tau_1) \leq f(\tau_2)$.
- (sat- \wedge): $I \models (\bigwedge S = \tau)$ iff $\bigwedge \{f(\sigma) \mid \sigma \in S\} = f(\tau)$.
- (sat- \vee): $I \models (\bigvee S = \tau)$ iff $\bigvee \{f(\sigma) \mid \sigma \in S\} = f(\tau)$.
- (sat- \neq): $I \models (\tau_1 \neq \tau_2)$ iff $f(\tau_1) \neq f(\tau_2)$.
- (sat-Atom): $I \models \text{Atom}(\tau)$ iff $f(\tau)$ is an atom in \mathbf{L} (i.e., $\sigma \leq f(\tau)$ implies $\sigma = f(\tau)$ or $\sigma = \perp$).

An *open specification* is a pair (P, Φ) , in which P is a finite set of clean types and $\Phi \subseteq \text{Constraints}(P)$. A *model* of (P, Φ) is an interpretation $I = (\mathbf{L}, f : \text{Aug}(P) \rightarrow L)$ for which $I \models \varphi$ holds for each $\varphi \in \Phi$. A model (\mathbf{L}, f) of (P, Φ) is *finite* if L is a finite set. The set of all models of (P, Φ) is denoted $\text{Mod}(P, \Phi)$.

It is convenient to partition the constraints into two classes. Elements of the first three categories listed above ($\{\leq, \wedge, \vee\}$) are called *positive constraints*, while elements of the last two ($\{\neq, \text{Atom}\}$) are called *negative constraints*. If $\Phi \subseteq \text{Constraints}(P)$, then Φ^+ denotes the positive constraints over P , and Φ^- the negative constraints. Similarly, $\text{Constraints}^+(P)$ (resp. $\text{Constraints}^-(P)$) denotes the set of positive (resp. negative) constraints contained in $\text{Constraints}(P)$.

It is important to note that other forms of constraints may be realized easily, even though they are not explicitly in this set. For example, a constraint of the form $(\tau_1 = \tau_2)$ is equivalent to the pair of constraints $\{(\tau_1 \leq \tau_2), (\tau_2 \leq \tau_1)\}$. Likewise, the constraint $(\tau_1 < \tau_2)$ may be viewed as an abbreviation for the set $\{(\tau_1 \leq \tau_2), (\tau_1 \neq \tau_2)\}$.

3.3 Canonical Models

In general, an open specification has a multitude of models. This situation arises for two reasons. First, it is always possible to augment a model with all sorts of extraneous types, without violating any constraints. For example, if $P_1 \subseteq P_2$, and Φ is any set of constraints dealing only with symbols in P_1 , then any model of (P_2, Φ) defines a model of (P_1, Φ) . Second, a model may impose a constraint which is not mandated by the specification. For example, for the specification $(\{a, b\}, \emptyset)$, a model in which $(a = b)$ holds is quite permissible, even though it is not required to hold in all models.

The question thus arises as to whether, for a given open specification (P, Φ) , there is a canonical model which (a) does not introduce any extraneous types, and (b) does not force any constraints not shared by all models. The answer is “yes,” provided that Φ does not contain any constraints of the form $\text{Atom}(\tau)$. Such a canonical model is the *initial model*, described as follows.

Let (P, Φ) be an open specification, and let $(\mathbf{L}_1, f_1 : \text{Aug}(P) \rightarrow L_1)$ and $(\mathbf{L}_2, f_2 : \text{Aug}(P) \rightarrow L_2)$ be models of (P, Φ) . A *morphism* $h : M_1 \rightarrow M_2$ is just a bounded lattice homomorphism $h : \mathbf{L}_1 \rightarrow \mathbf{L}_2$ with the property that $f_2 = h \circ f_1$. A model $N = (\mathbf{K}, i)$ for (P, Φ) is *initial* if, for every model $M = (\mathbf{L}, f)$ of (P, Φ) , there is a unique morphism $h : N \rightarrow M$. It is a standard result from category theory that such initial constructions, when they exist, are unique up to isomorphism [16, Chap. 4, Sec. 7]. Thus, we may speak of *the* initial or canonical model. In this paper, the term *canonical model* will be taken to be synonymous with initial model. We prefer the term canonical model because it conveys a sense of its representational power, while initial model conveys a purely algebraic property.

3.4 The Bounded Distributive Lattice of Crowns over P

The definition of canonical model is abstract; however, it is useful to have a basic understanding of its structural and combinatorial aspects. To this end, we provide a concrete construction, beginning with the situation (P, \emptyset) , in which there are no constraints. In that case, the canonical model is represented by the distributive lattice whose elements are \top , together with all expressions built up from elements of P using \wedge and \vee , subject to equivalence via the distributive laws. The idea of how such expressions are represented parallels closely the idea of *disjunctive normal form* (DNF) from propositional logic [10, pp. 48-49]. Just as any formula in propositional logic may be converted to one in DNF (using the distributivity of the corresponding operations), so too may any expression in a distributive lattice be converted to an equivalent one in such a form. Specifically, we work with expressions of the following form,

$$(a_{11} \wedge a_{12} \wedge \dots \wedge a_{1n_1}) \vee (a_{21} \wedge a_{22} \wedge \dots \wedge a_{2n_2}) \vee \dots \vee (a_{m1} \wedge a_{m2} \wedge \dots \wedge a_{mn_m})$$

in which the a_{ij} 's are elements of P .

Such representations of individual elements in a lattice can easily be confused with expressions involving the lattice operations \wedge and \vee . Therefore, an alternate

notation is adopted. If $C_i = \{a_{ij} \mid 1 \leq j \leq n_i\}$, then the *set representation* of the above expression is $\{C_1, C_2, \dots, C_m\}$. Thus, in the set representation, each element other than \top is represented by a set of subsets of elements of P .

There is one further problem; namely an element may have more than one representation. For example $(a \wedge b \wedge c) \vee (a \wedge b)$ and $(a \wedge b)$ are equivalent. To remedy this, we must disallow expressions in which the types in one of the disjuncts is a subset of those in another. This leads to a representation using *co-chains*, or *crowns*.

Let $T = \{C_1, C_2, \dots, C_n\}$ be a set of subsets of P . Call T a *crown* if for no two distinct indices i and j is it the case that $C_i \subseteq C_j$. Let $\text{Crown}(P)$ denote the set of all crowns of P , and let $\text{Crown}_\top(P)$ denote $\text{Crown}(P) \cup \{\top\}$.

It is easy to see that the elements of $\text{Crown}_\top(P)$ form a bounded distributive lattice, under natural operations. Specifically, the empty set \emptyset is the bottom element \perp of the lattice, and further operations are defined as follows.

$$\begin{aligned} (\text{cr-}\vee): \{C_1, C_2, \dots, C_n\} \vee \{D_1, D_2, \dots, D_m\} &= \\ &\text{Crownify}(\{C_1, C_2, \dots, C_n, D_1, D_2, \dots, D_m\}). \\ (\text{cr-}\wedge): \{C_1, C_2, \dots, C_n\} \wedge \{D_1, D_2, \dots, D_m\} &= \\ &\text{Crownify}(\{C_1 \cap D_j \mid 1 \leq i \leq n \text{ and } 1 \leq j \leq m\}). \end{aligned}$$

(The operation *Crownify* converts its argument into a crown by removing all subsumed subsets.) The lattice so constructed shall be denoted $\text{CrownLat}(P)$.

3.5 Theorem

For any set P , $(\text{CrownLat}(P), \iota : P \rightarrow \text{Crown}_\top(P))$, with $\iota : \tau \mapsto \{\{\tau\}\}$, is a canonical model over (P, \emptyset) .

Proof. The proof is a standard free algebra construction [13, Ch. 1, Sec. 5]. \square

3.6 Example

Let $P = \{a, b\}$. Then $\text{Crown}(P) = \{\{\emptyset\}, \{\{a\}\}, \{\{b\}\}, \{\{a, b\}\}, \{\{a\}, \{b\}\}\}$. The corresponding free lattice is leftmost in Fig. 1. (The other two lattices will be considered in 3.10 below.) This example also illustrates why there is a distinct element assigned to be \top , rather than just taking \top to be the largest crown consisting of all maximal subsets of P . The condition that \top is the join of all elements in P is a constraint, and not a condition which must be satisfied in all models.

3.7 Combinatorics of the Crown Construction

The size of the set of crowns grows very rapidly. For $P = \{a, b, c\}$, $\text{Crown}_\top(P)$ has eighteen elements. Specifically, $\text{Crown}(\{a, b, c\}) = \{\{\emptyset\}, \{\{a\}\}, \{\{b\}\}, \{\{c\}\}, \{\{a, b\}\}, \{\{a, c\}\}, \{\{b, c\}\}, \{\{a, b, c\}\}, \{\{a\}, \{b\}\}, \{\{a\}, \{c\}\}, \{\{b\}, \{c\}\}, \{\{a, b\}, \{c\}\}, \{\{a, c\}, \{b\}\}, \{\{b, c\}, \{a\}\}, \{\{a, b\}, \{a, c\}\}, \{\{a, b\}, \{b, c\}\}, \{\{a, c\}, \{b, c\}\}, \{\{a\}, \{b\}, \{c\}\}\}$.

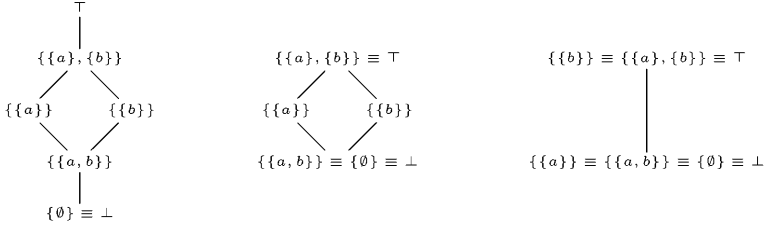


Fig. 1. Three canonical lattices over $P = \{a, b\}$.

In general, the number of elements in $\text{Crown}(P)$ is much greater than 2^n , although less than 2^{2^n} . Thus, *explicit construction of the canonical lattice for (P, Φ) is generally highly impractical*. This is true even when $\Phi \neq \emptyset$, as shown below. Generally, although the canonical lattice will be smaller, it will still be very large.

3.8 Coalescing Constraints and Quotient Lattices

We now turn to the problem of characterizing the canonical lattice over an open specification (P, Φ) in which Φ is not empty. A *congruence* on $\mathbf{L} = (L, \vee, \wedge, \top, \perp)$ is an equivalence relation \equiv on L with the property that whenever $x_1 \equiv x_2$ and $y_1 \equiv y_2$ hold, then $x_1 \vee y_1 = x_2 \vee y_2$ and $x_1 \wedge y_1 = x_2 \wedge y_2$. It is well known that if (and only if) \equiv is a congruence relation, the equivalence classes L/\equiv of L form a lattice \mathbf{L}/\equiv under the induced operations [13, p. 21].

It is straightforward to show that any set of positive constraints on P defines a congruence in a natural way. For $\tau \in P$, define $\text{ECrown}(\tau) = \{\{\tau\}\}$, with $\text{ECrown}(\perp) = \{\emptyset\}$ and $\text{ECrown}(\top) = \top$. Then, for $\Phi \subseteq \text{Constraints}^+(P)$, define \equiv_Φ to be the finest congruence relation on $\text{Crown}_\top(P)$ which includes the following identifications.

- $(\equiv_\Phi - \leq)$: If $(\tau_1 \leq \tau_2) \in \Phi$ with $\tau_1, \tau_2 \in P$, then $\{\{\tau_1\}, \{\tau_2\}\} \equiv_\Phi \{\{\tau_2\}\}$.
 If $(\top \leq \tau) \in \Phi$, then $\{\{\tau\}\} \equiv_\Phi \top$.
 If $(\tau \leq \perp) \in \Phi$, then $\{\{\tau\}\} \equiv_\Phi \{\emptyset\}$.
- $(\equiv_\Phi - \wedge)$: If $(\bigwedge S = \tau) \in \Phi$, then $\{S\} \equiv_\Phi \text{ECrown}(\tau)$.
- $(\equiv_\Phi - \vee)$: If $(\bigvee S = \tau) \in \Phi$, then $\{\{\sigma\} \mid \sigma \in S\} \equiv_\Phi \text{ECrown}(\tau)$.

Define $f_{\equiv_\Phi} : \text{Aug}(P) \rightarrow \text{Crown}_\top(P)/\equiv_\Phi$ by $\tau \mapsto [\{\{\tau\}\}]_{\equiv_\Phi}$, $\perp \mapsto [\{\emptyset\}]_{\equiv_\Phi}$, and $\top \mapsto [\top]_{\equiv_\Phi}$.

We are now in a position to crystallize when and how an open specification has a model. For technical reasons, we first address the case in which there are no constraints of the form $\text{Atom}(\tau)$.

3.9 Theorem — Characterization of Satisfiability

Let (P, Φ) be any open specification with the property that Φ contains no constraints of the form $\text{Atom}(\tau)$. Then the following conditions are equivalent.

- (a) $\text{Mod}(P, \Phi)$ is nonempty.
- (b) (P, Φ) has a finite canonical model, which is given by $(\text{Crown}_\top(P) / \equiv_\Phi, f_{\equiv_\Phi})$.
- (c) \equiv_Φ contains more than one equivalence class and, for each $(\tau_1 \neq \tau_2) \in \Phi^-$, $\{\{\tau_1\}\}$ and $\{\{\tau_2\}\}$ lie in distinct equivalence classes of \equiv_{Φ^+} .

Proof. First of all, assume that Φ contains only positive constraints. If \equiv_Φ contains only one equivalence class, then \perp and \top must collapse to the same element, which violates the definition of a bounded lattice. Thus, no model can exist. If there is more than one equivalence class, then the free algebra exists, as specified, using standard techniques for the construction of free lattices [13, Ch. 1, Sec. 5].

For the general case, note that an inequality constraint of the form $(\tau_1 \neq \tau_2)$ does not alter the canonical model, but it may prevent its existence. More specifically, one first computes the canonical model for the positive constraints, and then tests to see whether the inequality constraints are satisfied in that canonical model. The condition identified in (c) exactly recaptures this situation. \square

3.10 Examples

As in 3.6, let $P = \{a, b\}$, but now let $\Phi = \{(\bigvee\{a, b\} = \top), (\bigwedge\{a, b\} = \perp)\}$. The corresponding canonical lattice is in the middle of Fig. 1. If we add the constraint $(a \neq b)$ to Φ , this does not change the canonical model at all, since this constraint is already satisfied. Adding the constraint $(a \leq \perp)$ results in the rightmost lattice.

3.11 Dealing with Atomic Constraints

The results of 3.9 do not address atomic constraints (i.e., $\text{Atom}(\tau)$), because the situation surrounding such constraints is much more difficult. For example, let $P = \{a, b\}$, and let $\Phi = \{\text{Atom}(a), \text{Atom}(b)\}$. At first glance, it might appear that the leftmost lattice in Fig. 2 is an initial model for (P, Φ) . However, this is not the case. The rightmost lattice in Fig. 2 also satisfies these constraints, yet it is not a homomorphic image of the one on the left, since on the left $[\{\{a\}\}] \wedge [\{\{b\}\}] = \perp$, yet on the right $[\{\{a\}\}]$ and $[\{\{b\}\}]$ are the same element, and so this value is also the meet. This type of behavior is typical of constraints such as $\text{Atom}(\tau)$; initial models often do not exist.

From 3.9, we can deduce that in the absence of atomic constraints, when there is a model, there is a finite model. This is an extremely important result, since infinite hierarchies pose all sorts of conceptual and computational difficulties. Fortunately, this finite result remains valid even in the presence of atomic constraints, as shown below. It result must not be viewed as trivial or frivolous.

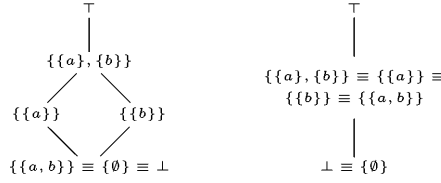


Fig. 2. Two candidates for initial lattices with constraints $\Phi = \{\text{Atom}(a), \text{Atom}(b)\}$.

There exist many classes of lattices (e.g., modular lattices) for which initial models may be infinite. See [13, Ch. 1, Sec. 5, Exer. 12]. From a computational point of view, it is indeed fortunate that the distributive model of a type hierarchy does not share this shortcoming, even in the presence of atomic constraints.

3.12 Theorem — Finiteness of Models

Let (P, Φ) be a finite open specification.

- (a) If (P, Φ) has a model, then it has a finite model.
- (b) If (P, Φ) has a canonical model, then this initial model is finite.

Proof. In both cases, the proof depends upon the observation that in any (not necessarily finite) distributive lattice, the sublattice generated by a finite set is itself finite. This is easily seen from the crown construction (3.4); under distributivity, there is only a finite number of distinct expressions which may be built up from a finite number of elements. Thus, if (P, Φ) has an infinite model, just extract the sublattice generated by $P \cup \{\top, \perp\}$; it is guaranteed to be finite. This also shows that any initial model must be finite, since the part not generated by $P \cup \{\top, \perp\}$ must be extraneous. \square

3.13 Complementation

In many frameworks, including that of CUF [7], complementation is an implicit operation. Thus, every type τ has a complementary type $\bar{\tau}$ which satisfies the conditions $\tau \vee \bar{\tau} = \top$ and $\tau \wedge \bar{\tau} = \perp$. For reasons of space limitation, we have not developed explicit results for this extended framework. However, with minor modifications, all of the theoretical results of this section carry through in the presence of complementation. Particularly, in the crown construction of 3.4, the sets of conjuncts (i.e., the C_i 's) involve symbols of the form a_{ij} and of the form \bar{a}_{ij} , subject to the condition that no set may contain both a and \bar{a} . Needless to say, the size of the canonical model is even larger in the presence of implicit complements. In a general context, further information on the effect of including implicit complementation may be found in [14].

4 Basic Computational Techniques

The results on the size of the canonical model identified in 3.7 suggest that explicit construction of this lattice from an open specification is unrealistic, except in the most limited of circumstances. Fortunately, it is not generally necessary to provide an explicit construction of the whole hierarchy. Often, it is sufficient to know whether a given open specification admits a model; that is, that it is not inconsistent. However, even for that question, we have the following rather negative result, which is proven in [14].

4.1 NP-Completeness

The question of whether a finite open specification (P, Φ) has a model is NP-complete, even when attention is restricted to positive sets of constraints. \square

Despite this negative result, it is important to identify basic solution techniques. In practice, NP-complete problems are dealt with effectively all the time using a variety of strategies. Later, we shall look at some of these approaches, but first some basic results must be established.

4.2 Two-Element Interpretations

The two-element lattice, denoted $\mathbf{2}$, contains $\{\perp, \top\}$ as its only elements. It is trivially distributive.

Let (P, Φ) be an open specification. A *two-element interpretation* for (P, Φ) is an interpretation of the form $(\mathbf{2}, f)$; thus $f : P \rightarrow \{\top, \perp\}$. The set of all two-elements interpretations for (P, Φ) is denoted $\text{Interp}_2(P, \Phi)$. The set of two-element models is denoted $\text{Mod}_2(P, \Phi)$.

4.3 Adequacy of Two-Element Models

A positive open specification (P, Φ) has a model iff it has a two-element model.

Proof. Let $(\mathbf{L}, g : P \rightarrow L)$ be a model of (P, Φ) . In view of the Birkhoff-Stone representation theorem (2.7), we may take \mathbf{L} to be a ring of sets. Let s be any element from this ring which appears in some members of L , but not in all. Then, it is easily verified that $(\mathbf{2}, g_s : P \rightarrow \{\top, \perp\})$ with $g_s : \tau \mapsto \top$ if $s \in g(\tau)$ and $g_s : \tau \mapsto \perp$ otherwise, is a two-element model. The converse is trivial. \square

4.4 Limitations of Two-Element Models

Two-element models are sufficient if one is interested only in positive constraints. However, they are inadequate for negative constraints. For example, let $P = \{\tau_1, \tau_2, \tau_3\}$, and let $\Phi = \{(\tau_1 \neq \tau_2), (\tau_1 \neq \tau_3), (\tau_2 \neq \tau_3)\}$. Then (P, Φ) cannot have a two-element model, because τ_1 , τ_2 , and τ_3 must be distinct elements in the underlying lattice. Fortunately, it is possible to combine two-element models to obtain larger models which satisfy both positive and negative constraints, as we now show.

4.5 Product Interpretations and Representations

A key notion in the construction of models for a general set of constraints is that of the *product model*. Let (P, Φ) be an open specification, and let $\mathcal{I} = \{(\mathbf{L}_j, g_j : \text{Aug}(P) \rightarrow L_j) \mid j \in J\}$ be a finite set of interpretations of (P, Φ) . Define the product interpretation $\prod \mathcal{I}$ to be $(\prod_{j \in J} \mathbf{L}_j, g : \text{Aug}(P) \rightarrow \prod_{j \in J} L_j)$. $\prod_{j \in J} \mathbf{L}_j$ is the product lattice, which is easily verified to be distributive.

Conversely, given an interpretation $I = (\mathbf{L}, g : \text{Aug}(P) \rightarrow L)$, it is possible to construct a product interpretation which satisfies the same constraints. In view of 2.3.3, \mathbf{L} may be taken to be a nonredundant ring of sets. Let S be the basis for such a ring. For each $s \in S$, define a two-element semantics $I_s = (\mathbf{2}, g_s : \text{Aug}(P) \rightarrow \{\perp, \top\})$ with $g_s : \tau \mapsto \top$ if $s \in g(\tau)$ and $\tau \mapsto \perp$ otherwise, for $\tau \in P$. Then $\prod_{s \in S} I_s$ satisfies the same constraints as I .

We are now able to provide the main characterization theorem for existence and structure of models of open specifications.

4.6 Characterization of Models of Arbitrary Open Specifications

An open specification (P, Φ) is satisfiable iff there is a finite nonempty family $\mathcal{I} \subseteq \text{Interp}_2(P, \Phi)$ satisfying the following conditions.

- (a) *Each $\varphi \in \Phi^+$ is satisfied by every $I \in \mathcal{I}$.*
- (b) *Each $\varphi \in \Phi^-$ of the form $(\tau_1 \neq \tau_2)$ is satisfied by least one $I \in \mathcal{I}$.*
- (b) *Each $\varphi \in \Phi^-$ of the form $\text{Atom}(\tau)$ is satisfied by exactly one $I \in \mathcal{I}$.*

□

4.7 Model Characterization via Satisfiability in Propositional Logic

We now turn to the question of computing two-element models for (P, Φ) . The simplest and most direct approach is to reduce the problem to one of satisfiability in a propositional logic. Associate with P a propositional logic whose propositional letters are $\{\tau_\tau \mid \tau \in \text{Aug}(P)\}$. To each $\varphi \in \text{Constraints}(P)$ is associated a propositional formula $\mathfrak{F}(\varphi)$, according to the table below. For a set Φ of constraints, define $\mathfrak{F}(\Phi) = \{\mathfrak{F}(\varphi) \mid \varphi \in \Phi\}$.

Constraint φ	Associated Logical Formula $\mathfrak{F}(\varphi)$
$\tau_1 \leq \tau_2$	$\tau_{\tau_1} \Rightarrow \tau_{\tau_2}$
$\bigvee_{i=1}^n \tau_i = \tau$	$\tau_{\tau_1} \vee \tau_{\tau_2} \vee \dots \vee \tau_{\tau_n} \Leftrightarrow \tau_\tau$
$\bigwedge_{i=1}^n \tau_i = \tau$	$\tau_{\tau_1} \wedge \tau_{\tau_2} \wedge \dots \wedge \tau_{\tau_n} \Leftrightarrow \tau_\tau$
$\tau_1 \neq \tau_2$	$\neg(\tau_{\tau_1} \Leftrightarrow \tau_{\tau_2})$
$\text{Atom}(\tau)$	τ_τ

Each *stable* interpretation I of the propositional logic (i.e., a truth assignment to each proposition, with τ_\top true and τ_\perp false) naturally defines a two-element interpretation $\mathfrak{A}^{-1}(I) = (\mathbf{2}, g_I : \text{Aug}(P) \rightarrow \{\perp, \top\})$ of P via $g_I : \tau \mapsto \top$ if τ_τ is true in I , and $\tau \mapsto \perp$ otherwise. Thus, *the two-element models of (P, Φ) are in natural bijective correspondence with the stable models of $\mathfrak{F}(\Phi)$.*

4.8 Queries

Given an open specification (P, Φ) , we may wish to know whether certain other constraints are implied by this specification. For example, if Φ is the partial specification of a hierarchy, we may wish to know whether Φ forces $\tau_1 = \tau_2$ to hold. A *query* is a question of the form “Does $\Phi \models \varphi$ hold?” with $\Phi \subseteq \text{Constraints}(P)$ and $\varphi \in \text{Constraints}(P)$. This question could be posed formally as the two queries $\Phi \models (\tau_1 \leq \tau_2)$ and $\Phi \models (\tau_2 \leq \tau_1)$. Unfortunately, the complexity of answering queries is co-NP-complete [11, Sec. 7.1].

4.9 Theorem – Complexity of Query Processing

Let P be any finite set. The question of whether $\Phi \models \varphi$ for $\Phi \subseteq \text{Constraints}(P)$ and $\varphi \in \text{Constraints}(P)$ is co-NP-complete, in the size of P .

Proof. First of all, note that $\Phi \models \varphi$ holds iff $\Phi \cup \{\neg\varphi\}$ is unsatisfiable, where $\neg\varphi$ is the negation of the constraint φ , with the obvious semantics. The set $\text{Constraints}(P)$, together with logical negations of its elements, will be called the set of *extended constraints* over P .

Now deciding whether a set of extended constraints is satisfiable is at least as difficult as deciding whether or not a set of ordinary constraints (i.e., a subset of $\text{Constraints}(P)$) is satisfiable. Thus, in view of 4.1, it is NP-hard. On the other hand, it is also in NP, since we may guess at a solution and then test it in linear time. Thus, the problem of deciding the *satisfiability* of $\Phi \cup \{\varphi\}$ is NP-complete. Consequently, the problem of deciding the *unsatisfiability* of such a set is co-NP-complete. \square

5 Efficient Consistency Checking Using Horn Clauses

The results presented in the previous section paint a fairly negative picture of the tractability issues surrounding maintenance of type hierarchies which are openly specified. In particular, it may not be feasible to conduct a complete check of the admissibility of a specification. Under such circumstances, there are two tactics which may be taken. First, one may look for an efficient strategy which solves a limited number of problem instances completely. Second, one may look for an efficient strategy which works on any problem instance, but yields only partial information. In this section, we take the latter approach.

Horn clauses form an important class of sentences for a variety of reasons [20]. Particularly, in propositional logic, they admit very efficient inference mechanisms. While the best known inference algorithms for general propositional logic run in exponential time in the worst case, Horn-clause inference may be performed in linear time [9], [15]. In that which follows, we show how to use a Horn-clause formulation to detect forced equivalences in open specifications (that is, constraint sets Φ for which $\Phi \models (\tau_1 = \tau_2)$). The result is not a mere query mechanism, but a procedure which generates a list of such equivalences. This technique improves substantially upon an earlier one presented in [14, Sec. 2.1], in simplicity, in improved computational complexity, and in extensibility.

5.1 Horn Clauses

In a propositional logic, a *Horn clause* is one in which at most one of the literals is positive. Usually, we will write the Horn clause $\neg p_1 \vee \neg p_2 \vee \dots \vee \neg p_n \vee q$ in rule form, as $p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q$. If there is no positive literal, we will write $p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow \mathbf{F}$, with \mathbf{F} representing the proposition which is always false. Simple clauses consisting of one positive literal (e.g., p) are referred to as *facts*; the atom name itself will represent such clauses. The empty clause will be represented by \mathbf{F} .

For any set Ψ of Horn clauses, define $\mathbf{Facts}(\Psi)$ to be the set of all facts which are semantic consequences of Ψ . $\mathbf{Facts}(\Psi)$ may be computed in time which is linear in the size of the clause set [9], [15].

We will also work with conjunctions of Horn clauses, as though they were clauses themselves. Thus, a formula such as $(p \wedge q \Rightarrow r \wedge s)$ is to be regarded as an abbreviation for the conjunction $(p \wedge q \Rightarrow r) \wedge (p \wedge q \Rightarrow s)$.

5.2 Exclusive-or Constraints

As noted above, inference on sets of Horn clauses is very fast; the best algorithms are linear in the size of the input set. Since the problem of determining satisfiability of an open specification is NP-complete (see 4.1 above), we certainly cannot expect Horn clauses to be a vehicle for the complete description of open specifications. Rather, some additional forms of representation must be employed to recapture completely such specifications.

A propositional formula of the form $p \oplus q$ is called an *exclusive-or constraint*, or *xor-constraint*, for short. The formula $p \oplus q$ is equivalent to $(p \vee q) \wedge (\neg p \vee \neg q)$. Such a constraint expresses the restriction that exactly one of two alternatives must be true. In the work reported in this section, the constraints identified in the table of 4.7 will be re-expressed using a combination of Horn clauses and xor-constraints. Such a representation carries the advantage that the “tractable” part of the representation (the Horn clauses) is completely separated from the “intractable” part (the xor-constraints). Effective computational techniques may then focus on the Horn part, looking for inconsistencies in the specification. While such a technique will not find all inconsistencies, it can find many, as we shall see.

5.3 The Horn Clauses Associated with an Open Specification

Let P be any finite set of clean types. Define two $\mathbf{Aug}(P)$ -indexed sets of propositions, as follows: $\mathbf{Prop}_{\uparrow}(P) = \{\mathbf{p}_{\tau} \mid \tau \in \mathbf{Aug}(P)\}$; $\mathbf{Prop}_{\downarrow}(P) = \{\mathbf{q}_{\tau} \mid \tau \in \mathbf{Aug}(P)\}$. $\mathbf{Prop}_{\uparrow}(P)$ denotes $\mathbf{Prop}_{\uparrow}(P) \cup \mathbf{Prop}_{\downarrow}(P)$. Relative to a two-element interpretation $I = (\mathbf{2}, f)$, think of \mathbf{p}_{τ} as representing the statement $f(\tau) = \top$, and \mathbf{q}_{τ} representing $f(\tau) = \perp$.

Now let Φ be a set of constraints over P . For each $\varphi \in \Phi$, associate two sets of Horn clauses, over $\mathbf{Prop}_{\uparrow}(P)$ and $\mathbf{Prop}_{\downarrow}(P)$, according to the table below.

φ	$\text{Rules}_{\uparrow}(\{\varphi\})$	$\text{Rules}_{\downarrow}(\{\varphi\})$
$(\tau_1 \leq \tau_2)$	$(\mathbf{p}_{\tau_1} \Rightarrow \mathbf{p}_{\tau_2})$	$\mathbf{q}_{\tau_2} \Rightarrow \mathbf{q}_{\tau_1}$
$(\bigvee_{i=1}^n \tau_i = \tau)$	$(\mathbf{p}_{\tau_1} \Rightarrow \mathbf{p}_{\tau}), \dots, (\mathbf{p}_{\tau_n} \Rightarrow \mathbf{p}_{\tau})$	$(\mathbf{q}_{\tau} \Rightarrow \mathbf{q}_{\tau_1}), \dots, (\mathbf{q}_{\tau} \Rightarrow \mathbf{q}_{\tau_n}),$ $(\mathbf{q}_{\tau_1} \wedge \mathbf{q}_{\tau_2} \wedge \dots \wedge \mathbf{q}_{\tau_n} \Rightarrow \mathbf{q}_{\tau})$
$(\bigwedge_{i=1}^n \tau_i = \tau)$	$(\mathbf{p}_{\tau} \Rightarrow \mathbf{p}_{\tau_1}), \dots, (\mathbf{p}_{\tau} \Rightarrow \mathbf{p}_{\tau_n}),$ $(\mathbf{p}_{\tau_1} \wedge \mathbf{p}_{\tau_2} \wedge \dots \wedge \mathbf{p}_{\tau_n} \Rightarrow \mathbf{p}_{\tau})$	$(\mathbf{q}_{\tau_1} \Rightarrow \mathbf{q}_{\tau}), \dots, (\mathbf{q}_{\tau_n} \Rightarrow \mathbf{q}_{\tau})$
$(\tau_1 \neq \tau_2)$	$(\mathbf{p}_{\tau_1} \wedge \mathbf{p}_{\tau_2} \Rightarrow \mathbf{F})$	$(\mathbf{q}_{\tau_1} \wedge \mathbf{q}_{\tau_2} \Rightarrow \mathbf{F})$
$\text{Atom}(\tau)$	(\mathbf{p}_{τ})	$(\mathbf{q}_{\tau} \Rightarrow \mathbf{F})$

For a set $\Phi \subseteq \text{Constraints}(P)$, define

$$\begin{aligned} \text{Rules}_{\uparrow}(\Phi) &= \{(\mathbf{p}_{\top}), (\mathbf{p}_{\perp} \Rightarrow \mathbf{F})\} \cup \left(\bigcup_{\varphi \in \Phi} \text{Rules}_{\uparrow}(\{\varphi\}) \right) \\ \text{Rules}_{\downarrow}(\Phi) &= \{(\mathbf{q}_{\perp}), (\mathbf{q}_{\top} \Rightarrow \mathbf{F})\} \cup \left(\bigcup_{\varphi \in \Phi} \text{Rules}_{\downarrow}(\{\varphi\}) \right). \end{aligned}$$

These sets of clauses are called the \uparrow -rules (resp. \downarrow -rules) for Φ . The notation is suggestive of the semantics of these rules. The \uparrow -rules express closure conditions on the elements of P which must be true in a two-element model $I = (\mathbf{2}, f)$, while the \downarrow -rules express similar conditions on elements which must be false. Consider the generic constraint $(\bigvee_{i=1}^n \tau_i = \tau)$. If it holds, several things are implied. First of all, for each i , $\tau_i \leq \tau$. Thus, for any i , if $f(\tau_i) = \top$, then it must be that $f(\tau) = \top$ also. This condition is recaptured by the rule $(\mathbf{p}_{\tau_i} \Rightarrow \mathbf{p}_{\tau})$. Similarly, if $f(\tau) = \perp$, then $f(\tau_i) = \perp$ must hold as well, and this is recaptured by the rule $(\mathbf{q}_{\tau} \Rightarrow \mathbf{q}_{\tau_i})$. Furthermore, if $f(\tau_i) = \perp$ for each i , then the join condition mandates that $f(\tau) = \perp$ also; this is recaptured by the rule $(\mathbf{q}_{\tau_1} \wedge \mathbf{q}_{\tau_2} \wedge \dots \wedge \mathbf{q}_{\tau_n} \Rightarrow \mathbf{q}_{\tau})$. Note that there is no corresponding \uparrow -rule in this case. The situation for a constraint of the form $(\bigwedge_{i=1}^n \tau_i = \tau)$ is completely analogous.

The rules in $\{(\mathbf{p}_{\top}), (\mathbf{p}_{\perp} \Rightarrow \mathbf{F})\}$ and in $\{(\mathbf{q}_{\perp}), (\mathbf{q}_{\top} \Rightarrow \mathbf{F})\}$ are called *bound rules*, because they assert that $f(\top) = \top$ and $f(\perp) = \perp$, respectively, for any two-element model $(\mathbf{2}, f)$ of (P, Φ) .

In addition to the \uparrow -rules and \downarrow -rules, there are xor-constraints which assert that for any $\tau \in P$, exactly one of $f(\tau) = \perp$ and $f(\tau) = \top$ must hold for any given two-element model $(\mathbf{2}, f)$. The *XOR constraints*, *rule set*, and *total representation* are defined as follows.

$$\begin{aligned} \text{XOR}(P) &= \{\mathbf{p}_{\tau} \oplus \mathbf{q}_{\tau} \mid \tau \in P\} \\ \text{Rules}(\Phi) &= \text{Rules}_{\uparrow}(\Phi) \cup \text{Rules}_{\downarrow}(\Phi) \\ \text{TotalRep}(P, \Phi) &= \text{Rules}(\Phi) \cup \text{XOR}(P) \end{aligned}$$

Finally, given a two-element interpretation $I = (\mathbf{2}, f)$ of (P, Φ) , define the *fact set* of I as $\text{FactSet}(f) = \{\mathbf{p}_x \mid x \in f^{-1}(\top)\} \cup \{\mathbf{q}_x \mid x \in f^{-1}(\perp)\}$. Then, using the semantics which have been outlined above, it is easy to establish the following alternative to 4.7.

5.4 Characterization of Two-Element Models

Let (P, Φ) be an open specification, and let $I = (\mathbf{2}, f) \in \text{Interp}_2(P, \Phi)$. Then $I \in \text{Mod}_2(P, \Phi)$ iff the set $\text{FactSet}(f) \cup \text{TotalRep}(P, \Phi)$ of propositional formulas is satisfiable. \square

5.5 Example

Let $P = \{\kappa_i \mid 1 \leq i \leq 6\}$, and let $\Phi = \{(\bigvee\{\kappa_1, \kappa_2\} = \kappa_5), (\bigvee\{\kappa_2, \kappa_3\} = \kappa_5), (\bigvee\{\kappa_1, \kappa_3\} = \kappa_4), (\bigwedge\{\kappa_2, \kappa_3\} = \kappa_6), (\bigwedge\{\kappa_1, \kappa_6\} = \perp), ((\kappa_5 \leq \kappa_4))\}$. The table below shows the associated rules.

$\text{Rules}_{\uparrow}(\Phi)$	$\text{Rules}_{\downarrow}(\Phi)$
$\mathbf{p}_{\kappa_1} \Rightarrow \mathbf{p}_{\kappa_4} \wedge \mathbf{p}_{\kappa_5}$	$\mathbf{q}_{\kappa_2} \Rightarrow \mathbf{q}_{\kappa_6}$
$\mathbf{p}_{\kappa_2} \Rightarrow \mathbf{p}_{\kappa_5}$	$\mathbf{q}_{\kappa_3} \Rightarrow \mathbf{q}_{\kappa_6}$
$\mathbf{p}_{\kappa_3} \Rightarrow \mathbf{p}_{\kappa_4} \wedge \mathbf{p}_{\kappa_5}$	$\mathbf{q}_{\kappa_4} \Rightarrow \mathbf{q}_{\kappa_1} \wedge \mathbf{q}_{\kappa_3} \wedge \mathbf{q}_{\kappa_5}$
$\mathbf{p}_{\kappa_5} \Rightarrow \mathbf{p}_{\kappa_4}$	$\mathbf{q}_{\kappa_5} \Rightarrow \mathbf{q}_{\kappa_1} \wedge \mathbf{q}_{\kappa_2} \wedge \mathbf{q}_{\kappa_3}$
$\mathbf{p}_{\kappa_6} \Rightarrow \mathbf{p}_{\kappa_2} \wedge \mathbf{p}_{\kappa_3}$	$\mathbf{q}_{\kappa_1} \wedge \mathbf{q}_{\kappa_2} \Rightarrow \mathbf{q}_{\kappa_5}$
$\mathbf{p}_{\kappa_2} \wedge \mathbf{p}_{\kappa_3} \Rightarrow \mathbf{p}_{\kappa_6}$	$\mathbf{q}_{\kappa_2} \wedge \mathbf{q}_{\kappa_3} \Rightarrow \mathbf{q}_{\kappa_5}$
$\mathbf{p}_{\kappa_1} \wedge \mathbf{p}_{\kappa_6} \Rightarrow \mathbf{p}_{\perp}$	$\mathbf{q}_{\kappa_1} \wedge \mathbf{q}_{\kappa_3} \Rightarrow \mathbf{q}_{\kappa_4}$
\mathbf{p}_{\top}	$\mathbf{q}_{\top} \Rightarrow \mathbf{F}$
$\mathbf{p}_{\perp} \Rightarrow \mathbf{F}$	\mathbf{q}_{\perp}

Note that the rules $(\mathbf{p}_{\perp} \Rightarrow \mathbf{p}_{\kappa_1})$, $(\mathbf{p}_{\perp} \Rightarrow \mathbf{p}_{\kappa_6})$, $(\mathbf{q}_{\kappa_1} \Rightarrow \mathbf{q}_{\top})$, and $(\mathbf{q}_{\kappa_6} \Rightarrow \mathbf{q}_{\top})$ are not included in the table, even though they are formally members of the appropriate rule set. Since \mathbf{p}_{\perp} is always false, and \mathbf{q}_{\top} is always true, they are trivial tautologies, and so may be omitted.

Next, define the two-element interpretations $I_j = (\mathbf{2}, f_j)$ for $1 \leq j \leq 7$ according to the following table.

j	$f_j^{-1}(\top)$	$f_j^{-1}(\perp)$
1	$\{\kappa_3, \kappa_4, \kappa_5\}$	$\{\kappa_1, \kappa_2, \kappa_6\}$
2	$\{\kappa_2, \kappa_4, \kappa_5\}$	$\{\kappa_1, \kappa_3, \kappa_6\}$
3	\emptyset	$\{\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6\}$
4	$\{\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6\}$	\emptyset
5	$\{\kappa_1, \kappa_4\}$	$\{\kappa_2, \kappa_3, \kappa_5, \kappa_6\}$
6	$\{\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5\}$	$\{\kappa_6\}$
7	$\{\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5, \kappa_6\}$	\emptyset

I_1 , I_2 , and I_3 , are easily verified to be models of (P, Φ) . Indeed, it is not difficult to see that they are the only two-element models. On the other hand, I_4 fails to be a model, since $(\mathbf{p}_{\kappa_1} \wedge \mathbf{p}_{\kappa_6} \Rightarrow \mathbf{p}_{\perp})$ would then mandate that $\perp \in f_4^{-1}(\top)$, an impossibility. I_5 fails to be a model of (P, Φ) , since the rule $(\mathbf{p}_{\kappa_1} \Rightarrow \mathbf{p}_{\kappa_4} \wedge \mathbf{p}_{\kappa_5})$ mandates that $\kappa_5 \in f_5^{-1}(\top)$, contradicting $\kappa_5 \in f_5^{-1}(\perp)$. Similarly, I_6 is not a model of (P, Φ) , since the rule $(\mathbf{p}_{\kappa_2} \wedge \mathbf{p}_{\kappa_3} \Rightarrow \mathbf{p}_{\kappa_6})$ mandates that $\kappa_6 \in f_6^{-1}(\top)$, contradicting $\kappa_6 \in f_6^{-1}(\perp)$. Finally, I_7 is not a model, since the rule $(\mathbf{p}_{\kappa_1} \wedge \mathbf{p}_{\kappa_6} \Rightarrow \mathbf{q}_{\perp})$ requires that at least one of $\{\kappa_1, \kappa_6\}$ lie in $f_7^{-1}(\perp)$.

Now let $\Phi' = \Phi \cup \{(\kappa_4 \neq \kappa_5)\}$. Then $\text{TotalRep}(P, \Phi') = \text{TotalRep}(P, \Phi) \cup \{(\mathbf{p}_{\kappa_4} \wedge \mathbf{p}_{\kappa_5} \Rightarrow \mathbf{F}), (\mathbf{q}_{\kappa_4} \wedge \mathbf{q}_{\kappa_5} \Rightarrow \mathbf{F})\}$. It is easy to see that (P, Φ) has no model. Indeed, $\{(\mathbf{q}_{\kappa_5} \Rightarrow \mathbf{q}_{\kappa_1} \wedge \mathbf{q}_{\kappa_2} \wedge \mathbf{q}_{\kappa_3}), (\mathbf{q}_{\kappa_1} \wedge \mathbf{q}_{\kappa_3} \Rightarrow \mathbf{q}_{\kappa_4})\} \models (\mathbf{q}_{\kappa_5} \Rightarrow \mathbf{q}_{\kappa_4})$, and since $(\mathbf{q}_{\kappa_4} \Rightarrow \mathbf{q}_{\kappa_5})$ also holds, this means that for any two-element model $(\mathbf{2}, f)$, $\kappa_4 \in f^{-1}(\perp)$ iff $\kappa_5 \in f^{-1}(\perp)$. Thus, $f(\kappa_4) = f(\kappa_5)$. This is impossible, hence there can be no model of (P, Φ) .

5.6 Static Consistency Checking

Let (P, Φ) be an open specification. For any set $X \subseteq \text{Prop}_{\uparrow}(P)$, $\text{Facts}(X \cup \text{Rules}(\Phi))$ may be computed very efficiently — in time proportional to the size of Φ . The idea behind static consistency checking is to compute $\text{Facts}(X \cup \text{Rules}(\Phi))$ for each member X of a judiciously chosen set of subsets of $\text{Prop}_{\uparrow}(P)$. From this computation, many properties of solutions may be detected. One such example is provided by (P, Φ) of 5.5. The condition $(\kappa_4 = \kappa_5)$ holds in every model, as the failure of (P, Φ') to have a model confirms. In 5.5, this failure was shown via a direct proof, which essentially posed $(\kappa_4 = \kappa_5)$ as a query. To check each such condition separately would require a great deal of computational resources. With a static consistency check, on the other hand, we can identify a large number of such conditions at one time. We now develop the machinery to perform such checks systematically. First, observe the following result, which follows immediately from the definition of the fact set.

5.7 Utility of Fact Closure

Let (P, Φ) be an open specification, and let $X \subseteq \text{Prop}_{\uparrow}(P)$. Then $\text{Rules}(\Phi) \models (\bigwedge X \Rightarrow (\bigwedge \text{Facts}(X \cup \text{Rules}(\Phi))))$. In other words, the process of computing $\text{Facts}(X \cup \text{Rules}(\Phi))$ may essentially be viewed as one of applying the rules in $\text{Rules}(\Phi)$ to X . (Note: \bigwedge denotes logical conjunction here, not lattice join.) \square

5.8 Aggregate Complexity for Fact Closure

Let (P, Φ) be an open specification, and let \mathbf{S} be a set of subsets of $\text{Prop}_{\uparrow}(P)$. Then the set of sets $\{\text{Facts}(S \cup \text{Rules}(\Phi)) \mid S \in \mathbf{S}\}$ may be computed in time $\Theta(n + r \cdot s + r \cdot \log(r))$, with n the cardinality of P , s is the number of set in \mathbf{S} , and r is the sum of the lengths of the rules in $\text{Rules}(\Phi)$.

Proof. The proof rests largely upon results found in [9] and [15]. The process is broken into two steps. There is a total of $2(n + 2)$ propositions in $\text{Prop}_{\uparrow}(P) \cup \text{Prop}_{\downarrow}(P)$. Assign each proposition a natural-number tag in $\{0, \dots, 2n + 3\}$. This takes time $\Theta(n)$. Next, sort the propositions in each antecedent set of each clause. This takes time $\Theta(r \cdot \log(r))$.

After this preconditioning, the computation of each $\text{Facts}(S \cup \text{Rules}(\Phi))$ takes just $\Theta(r)$ time, using the techniques in the above-cited references. The total time for all elements of \mathbf{S} is thus $\Theta(s \cdot r)$. Combining these, the running time for the entire algorithm is $\Theta(n + r \cdot \log(r) + s \cdot r)$. \square

5.9 Singleton Associations

Let (P, Φ) be an open specification, and let $p \in \text{Prop}_{\uparrow}(P)$. The *singleton associates* of p , denoted $\text{SingAsc}(p)$, is just $\text{Facts}(\{p\} \cup \text{Rules}(\Phi))$. In words, $\text{SingAsc}(p)$ is the set of all propositions in $\text{Prop}_{\uparrow}(P)$ which can be deduced from p alone, using the rules in $\text{Rules}(\Phi)$.

5.10 Example

Let (P, Φ) be as in 5.5. Here are the singleton associates.

x	$\text{SingAsc}(p_x)$	$\text{SingAsc}(q_x)$
κ_1	$\{p_{\kappa_1}, p_{\kappa_4}, p_{\kappa_5}\}$	$\{q_{\kappa_1}\}$
κ_2	$\{p_{\kappa_2}, p_{\kappa_4}, p_{\kappa_5}\}$	$\{q_{\kappa_2}, q_{\kappa_6}\}$
κ_3	$\{p_{\kappa_3}, p_{\kappa_4}, p_{\kappa_5}\}$	$\{q_{\kappa_3}, q_{\kappa_6}\}$
κ_4	$\{p_{\kappa_4}\}$	$\{q_{\kappa_1}, q_{\kappa_3}, q_{\kappa_4}, q_{\kappa_5}\}$
κ_5	$\{p_{\kappa_4}, p_{\kappa_5}\}$	$\{q_{\kappa_1}, q_{\kappa_2}, q_{\kappa_3}, q_{\kappa_4}, q_{\kappa_5}\}$
κ_6	$\{p_{\kappa_2}, p_{\kappa_3}, p_{\kappa_6}\}$	$\{q_{\kappa_6}\}$
\perp	$\{p_{\perp}\}$	$\{q_{\perp}\}$
\top	$\{p_{\top}\}$	$\{q_{\top}\}$

The information that $(\kappa_4 = \kappa_5)$ is easily recovered from these data. Indeed, note that $q_{\kappa_5} \in \text{SingAsc}(p_{\kappa_4})$ and that $q_{\kappa_4} \in \text{SingAsc}(p_{\kappa_5})$. In light of 5.7, this means that both $(q_{\kappa_4} \Rightarrow q_{\kappa_5})$ and $(q_{\kappa_5} \Rightarrow q_{\kappa_4})$ are logical consequences of $\text{Rules}(\Phi)$. Thus, for any $(2, f) \in \text{Mod}_2(\Phi)$, $f(\kappa_4) = \perp$ iff $f(\kappa_5) = \perp$. Thus, it must be the case that $f(\kappa_4) = f(\kappa_5)$. We now develop a means of discovering such associations systematically.

5.11 The Static Equivalence

Define the relation \preceq_{Φ}^1 on $\text{Prop}_{\uparrow}(\Phi)$ by $p \preceq_{\Phi}^1 q$ iff $q \in \text{SingAsc}(p)$. The transitive closure of \preceq_{Φ}^1 is denoted by $\overline{\preceq_{\Phi}^1}$. The equivalence relation \equiv_{Φ}^1 places into a single equivalence class all elements which lie in the same cycle in $\overline{\preceq_{\Phi}^1}$. Formally, $p \equiv_{\Phi}^1 q$ iff $p \overline{\preceq_{\Phi}^1} q$ and $q \overline{\preceq_{\Phi}^1} p$.

In the above example, only $q_{\kappa_4} \equiv_{\Phi}^1 q_{\kappa_5}$.

5.12 The Complexity of Determining Static Equivalence

Let (P, Φ) be an open specification. Then, with n , r , and s defined as in 5.8, there is an algorithm which computes \equiv_{Φ}^1 from (P, Φ) in worst-case time $\Theta(n^3 + n \cdot r + r \cdot \log(r))$.

Proof. There are $2n+4$ ($= O(n)$) distinct sets of the form $\text{SingAsc}(p)$, two for each element of $\text{Aug}(P)$. Transitive closure has the same computational complexity as matrix multiplication [18, 10.3.6]; thus, the equivalence relation \equiv_{Φ}^1 may be computed in time $\Theta(n^3)$ from $\{\text{Facts}(\{p\} \cup \text{Rules}(\Phi)) \mid p \in \text{Prop}_{\uparrow}(P)\}$. Thus, in view of 5.8, the total complexity is $\Theta(n^3) + \Theta(n + r \cdot s + r \cdot \log(r)) = \Theta(n^3 + n \cdot r + r \cdot \log(r))$. \square

The above bound is a substantial improvement over $\Theta(n^4 \cdot r \cdot \log(r))$ which was reported for the algorithm in [14, Sec. 2.1]. The general idea also lends itself to extension, as outlined below.

5.13 Higher-Level Static Consistency Checking

Although the technique just described detects many element equivalences, it cannot find them all. As a concrete example, consider the open specification (P, Φ) with $P = \{\tau_i \mid 1 \leq i \leq 5\}$, and $\Phi = \{(\tau_i \vee \tau_j = \tau_4) \mid 1 \leq i < j \leq 3\} \cup \{(\tau_i \wedge \tau_j = \tau_5) \mid 1 \leq i < j \leq 3\}$. It is not difficult to see that all five elements of P must be collapsed to the same value in any model. However, this fact is not detected by the static equivalence algorithm of 5.11. It can, however, be detected with a higher-level static consistency check, which works with sets of the form $\text{Facts}(X, \Phi)$, with X a subset of $\text{Prop}_{\uparrow}(P)$ of size at most two. Thus, the technique of static consistency checking may be extended. Indeed, if we work with all sets of the form $\text{Facts}(X, \Phi)$ for X any subset of $\text{Prop}_{\uparrow}(P)$, then consistency checking may be made complete. Unfortunately, the computational complexity which results when all subsets of P are considered yields no advantage over direct satisfaction testing. What may prove promising, though, is to work with all subsets of a small size bound, say two or three. The complexity is still quite manageable, yet many inconsistencies may be detected. This approach is not elaborated further here.

6 Conclusions and Further Directions

Open specification clearly imposes a substantial computational burden. Therefore, a decision to employ it must be measured carefully. As implied by the work in Sec. 2, the first question to ask is whether or not natural semantics are needed for both meet and join. If only natural meet semantics is needed, then open specification, as described in this paper, is not an issue.⁴ However, if one is interested manipulating general classes of representations which involve disjunction, then it may be a necessity, although alternatives to managing limited disjunction have been proposed and implemented [12]. We cannot and do not address the adequacy of such approaches rather we proceed under the assumption that join semantics, and hence distributive hierarchies, are desired.

As illustrated by the construction of 3.7, complete representation of a distributive hierarchy will generally be infeasible. (See also [14, Sec. 0] for some examples.) Therefore, it would seem that open specification is the only alternative. Although we have presented an efficient method for determining certain implied constraints in Sec. 5, such techniques, by themselves, cannot counterbalance the NP-hardness of the underlying problems. Rather, techniques for tackling these underlying problems directly must be developed. Fortunately, NP-hard problems

⁴ It is unclear whether the idea of open specification of meet-only hierarchies is interesting, or nontrivial. We know of no existing work on the topic, and no systems which embody the idea.

are very common, and there is a substantial body of research on computational methods [17]. However, most of these techniques deal with optimization problems, in which the notion of an approximate answer makes sense. On the other hand, as made clear in 4.7, the questions involved in open specification are related closely to satisfiability questions for logical formulas; such problems do not admit useful notions of approximation. Fortunately, there is an active body of research on such problems, as well as a library of tools known *SATLIB – The Satisfiability Library*, which is available on the world-wide web. Our next steps in addressing the problems of open specification must clearly be experimental ones, and will proceed as follows.

1. Direct solution of the associated logic problems, as characterized in 4.7, will be addressed using SATLIB tools such as GSAT [23], a tool which is effective in the solution of large satisfiability problems.
2. A study of techniques related to re-use. One of the features of the satisfiability problems surrounding open specification is that not one, but a whole family of satisfiability problems (one for each two-element model) must be obtained. It is clear from the results of Sec. 4, and 4.7 in particular, that the members of the families of formulas to be solved are closely related; they may differ in only slightly. Therefore, techniques which solve whole families of related formulas in a fashion more efficient than solving each individually must be developed. As far as we know, such techniques are not part of current work on the subject.
3. The alternate characterization of two element models presented in 5.4 suggests that techniques which address re-use in the specific context of Horn and XOR-constraints, may prove useful. As far as we know, SATLIB-style results for such specially conditioned formulas do not exist at this time.

References

1. Ait-Kaci, H. An algebraic semantics approach to the effective resolution of type equations. *Theoret. Comput. Sci.*, 45:293–351, 1986.
2. Ait-Kaci, H., R. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Trans. Programming Languages and Systems*, 11:115–146, 1989.
3. Briscoe, T., V. de Paiva, and A. Copestake, editors. *Inheritance, Defaults, and the Lexicon*. Cambridge University Press, 1993.
4. Carpenter, B. *The Logic of Typed Feature Structures*. Cambridge University Press, 1992.
5. Carpenter, B., and G. Penn. ALE: The Attribute Logic Engine user’s guide, Version 3.1 Beta. Technical report, Bell Laboratories and Universität Tübingen, 1998.
6. Paiva, V. de. Types and constraints in the LKB. In T. Briscoe, V. de Paiva, and A. Copestake, editors, *Inheritance, Defaults, and the Lexicon*, pages 164–189. Cambridge University Press, 1993.
7. Dörre, J., and M. Dorna. CUF – a formalism for linguistic knowledge representation. In Dörre, J., editor, *Computational Aspects of Constraint-Based Linguistic Description, DYANA-2 Deliverable R.1.2.A*, pages 3–22. ESPRIT, 1993.
8. Dörre, J., and A. Eisele. Feature logic with disjunctive unification. In *Proceedings of the COLING 90, Volume 2*, pages 100–105, 1990.

9. Dowling, W. F., and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn clauses. *J. Logic Programming*, 3:267–284, 1984.
10. Enderton, H. B. *A Mathematical Introduction to Logic*. Academic Press, 1972.
11. Garey, M. R., and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, 1979.
12. Gerdemann, D., and P. J. King. The correct and efficient implementation of appropriateness conditions for typed feature structures. In *Proceedings of COLING-94*, pages 956–960, 1994.
13. Grätzer, G. *General Lattice Theory*. Academic Press, 1978.
14. Hegner, S. J. Distributivity in incompletely specified type hierarchies: Theory and computational complexity. In J. Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description II, DYANA-2, ESPRIT Basic Research Project 6852, Deliverable R1.2B*, pages 29–120. DYANA, 1994. Also available as LLI Technical Report No. 4, University of Oslo, Department of Linguistics.
15. Hegner, S. J. Properties of Horn clauses in feature-structure logic. In Rupp, C. J., M. A. Rosner, and R. L. Johnson, editors, *Constraints, Languages and Computation*, pages 111–147. Academic Press, 1994.
16. Herrlich, H., and G. E. Strecker. *Category Theory*. Allyn and Bacon, 1973.
17. Hochbaum, D. S., editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing, 1997.
18. Horowitz, E., S. Sahni, and S. Rajasekaran. *Computer Algorithms*. Computer Science Press, 1998.
19. Levesque, H. J., and R. J. Brachman. Expressiveness and tractability in knowledge representation. *Computational Intelligence*, 3:78–93, 1987.
20. Makowsky, J. A. Why Horn formulas matter in computer science: Initial structures and generic examples. *J. Comput. System Sci.*, 34:266–292, 1987.
21. Meurers, W. D. On implementing an hpsg theory—aspects of the logical architecture, the formalization, and the implementation of head-driven phrase structure grammars. In Hinrichs, E. W., W. D. Meurers, and T. Nakazawa, editors, *Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation*, volume 58 of *Arbeitspapiere des SFB 340*. University of Tübingen, 1994.
22. Pollard, C., and I. A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, 1994.
23. Selman, B., and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of IJCAI-93*, pages 290–295, 1993.
24. Zajac, R. Notes on the Typed Feature System, Version 4, January 1991. Technical report, Universität Stuttgart, Institut für Informatik, Project Polygloss, 1991.

Anaphora and Quantification in Categorical Grammar

Gerhard Jäger

Zentrum für Allgemeine Sprachwissenschaft
Jägerstr. 10/11
D-10117 Berlin
Germany
`jaeger@zas.gwz-berlin.de`

Abstract. The paper proposes a type logical reformulation of Jacobson’s ([9]) treatment of anaphoric dependencies in Categorical Grammar. To this end, the associative Lambek Calculus is extended with a new connective. In the first part, its proof theory is introduced and the logical properties of the resulting calculus are discussed. In the second part, this system is applied to several linguistic phenomena concerning the interaction of pronominal anaphora, VP ellipsis and quantifier scope. Finally, a possible extension to cross-sentential anaphora is considered.

1 Introduction

Anaphora phenomena are a challenge to Categorical Grammar (CG) for several reasons. To start with, the standard treatment of anaphoric pronouns as variables is not really viable in CG due to its essentially variable free design. This is self-evident in Combinatory Categorical Grammar (CCG) since here semantic operations are by definition restricted to combinators (in the sense of Combinatory Logic). Thus in CCG the grammar does not provide variable binding devices. Researchers working in the framework of Type Logical Grammar usually don’t stress the variable free setup of the syntax-semantics interface, but things are not different here than in CCG. This becomes clear if one acknowledges the fact that the type logics used in CG can be seen as fragments of positive Intuitionistic Logic. Thus all type logical proof terms (= admissible semantic operations) can be expressed as combinatory terms (using only Curry’s **S** and **K**).

If one accepts that anaphora does not involve variables, one is faced with another problem. Virtually by definition, anaphora involves a multiple use of semantic resources. To put it the other way round, anaphors are expressions that use a resource without consuming it. In resource conscious logics, re-use of resources is characteristic for systems that contain the structural rule of contraction, like Relevance Logic or Intuitionistic Logic. Such logics lack the finite reading property, i.e one and same valid sequent may have infinitely many proof terms. A simple but telling example is the sequent $p \rightarrow p \Rightarrow p \rightarrow p$. All terms $\lambda x.f^n x$ for finite n are valid relevant or intuitionistic proof terms. Natural language utterances are at most finitely ambiguous though. The finite reading property is thus essential for an adequate grammar logic. So a type logical treatment

of anaphora should be strong enough to admit multiple use of resources, but weak enough to have the finite reading property.

In this paper, we first propose an extension of the Lambek Calculus **L** ([13]) that overcomes this problem. In the sequel, we illustrate its linguistic applications. After discussing the treatment of anaphoric pronouns, we study the interaction of this system with Moortgat’s account of quantification ([15,16]). Next we extend the analysis to VP ellipsis (VPE henceforth) and discuss the interplay of pronominal anaphora and quantification with VPE. Finally we sketch how cross-sentential anaphora can be handled in this setup.

2 The Logic **L**|

2.1 Background: Jacobson’s Proposal

In a series of publications ([6,7,8,9]), Pauline Jacobson has shown how pronominal anaphora can be handled successfully in CCG while maintaining the variable freeness of this framework. The basic intuition of her proposal is the idea that anaphoric expressions denote functions from antecedent meanings to contextually determined meanings. Applied to anaphoric pronouns, this means that they denote the identity function on individuals. Due to the strict category-to-type correspondence in CG, this must be reflected in the syntactic category. To this end, she extends the inventory of type forming connectives of CCG with a third slash $|$. A sign of category $A|B$ is an item that needs an antecedent of category B to behave like a sign of category A . Accordingly, its denotation will be a function from B -denotations to A -denotation. In other words, $|$ creates a functional category like the other two slashes. Under this account, anaphoric pronouns have category $N|N$ and denote the identity function on individuals. The non-local nature of anaphora is taken care of by a generalized version of function composition which makes arbitrarily large portions of syntactic material transparent for anaphoric dependencies. The job of connecting an anaphor to its antecedent—multiplication of a resource in the constructive jargon—is taken over by a modified version of Curry’s **S** that she calls **Z**.

The logic that is introduced in the next subsection can be seen as a translation of Jacobson’s ideas into the type logical architecture. In particular, we will adopt Jacobson’s slash and its intuitive interpretation, including the functional semantics of anaphors. Furthermore all relevant combinators of her system are theorems of our logic. Nonetheless—due to the fact that the Lambek Calculus is completely associative but CCG isn’t—the empirical predictions of the two approaches do not completely coincide. In Sect. 5 we will try to demonstrate that the move from combinatory to type logical design is advantageous from a descriptive point of view.

2.2 Sequent Presentation

The set of formulas \mathcal{F} of the logic **L**| is defined as the closure of some set \mathcal{A} of atomic formulas under the binary operations \backslash , \bullet , $/$ and $|$.

The sequent presentation of $\mathbf{L|}$ extends Lambek's ([13]) syntactic calculus \mathbf{L} with the logical rules for the third slash “ $|$ ”. Sequent rules are augmented with Curry-Howard terms.

$$\frac{}{x : A \Rightarrow x : A} id$$

$$\frac{X \Rightarrow M : A \quad Y, x : A, Z \Rightarrow N : B}{Y, X, Z \Rightarrow N[x \leftarrow M] : B} Cut$$

$$\frac{X, x : A, y : B, Y \Rightarrow M : C}{X, z : A \bullet B, Y \Rightarrow M[x \leftarrow (z)_0, y \leftarrow (z)_1] : C} \bullet L$$

$$\frac{X \Rightarrow M : A \quad Y \Rightarrow N : B}{X, Y \Rightarrow \langle M, N \rangle : A \bullet B} \bullet R$$

$$\frac{X \Rightarrow M : A \quad Y, x : B, Z \Rightarrow N : C}{Y, y : B/A, X, Z \Rightarrow N[x \leftarrow (yM)] : C} /L$$

$$\frac{X, x : A \Rightarrow M : B}{X \Rightarrow \lambda x.M : B/A} /R \quad \text{with } X \text{ non-empty}$$

$$\frac{X \Rightarrow M : A \quad Y, x : B, Z \Rightarrow N : C}{Y, X, y : A \setminus B, Z \Rightarrow N[x \leftarrow (yM)] : C} \setminus L$$

$$\frac{x : A, X \Rightarrow M : B}{X \Rightarrow \lambda x.M : A \setminus B} \setminus R \quad \text{with } X \text{ non-empty}$$

$$\frac{Y \Rightarrow M : B \quad X, x : B, Z, y : A, W \Rightarrow N : C}{X, Y, Z, z : A|B, W \Rightarrow N[x \leftarrow M][y \leftarrow (zM)] : C} |L$$

$$\frac{x : B, y : p, X \Rightarrow \langle M, y, N \rangle : B \bullet p \bullet A \quad x : B, X \Rightarrow \langle M, N \rangle : B \bullet A}{X \Rightarrow \lambda x.N : A|B} |R$$

with

X non-empty

p is atomic and does not occur in A, B, X

$M =_{\alpha\beta\eta} x$

A few comments are in order. Even though the Jacobsonian slash has a functional semantics and thus resembles the standard categorial slashes, it has a

different nature since it cannot be reconstructed as a residuation operation of some product operator. In the presence of cut, rule $|L$ is equivalent to the axioms

- (1) a. $x : A, y : B | A \Rightarrow \langle x, (yx) \rangle : A \bullet B$
 b. $x : A, y : B, z : C | A \Rightarrow \langle x, y, (zx) \rangle : A \bullet B \bullet C$

Together with the rule of proof $|R$ this expresses the intuition that a sign has category $A|B$ if and only if it behaves like a sign of category A in the presence of an antecedent of type B . Anaphor and antecedent may, but need not be adjacent. This motivates the two premises in $|R$. Since the atom p in rule $|R$ does not occur anywhere else, it behaves like a variable over the material between anaphor and antecedent. Furthermore it has to be ensured that neither the antecedent nor the material in between are affected by anaphora resolution. This motivates the constraint of the Curry-Howard terms in the premises of $|R$. So labeling is not just a book keeping device but a genuine restriction here.

$\mathbf{L}|$ has the desired proof theoretic properties. To start with, cut elimination is possible.

Theorem 1 (Cut Elimination)

Cut is admissible in $\mathbf{L}|$.

Proof. The proof relies on the following two lemmas:

Lemma 1 If Π is a correct proof and p an atomic formula that occurs in the conclusion of Π , then $\Pi[p \leftarrow B]$ is a correct proof as well, where $\Pi[p \leftarrow B]$ is the result of replacing all occurrences of p in Π by B .

Proof. By Induction over the complexity of Π . □

Lemma 2 With X^\bullet we refer to the formula that results from replacing all commas in X by \bullet . Then it holds that

$$\vdash X \Rightarrow X^\bullet$$

Proof. Induction over the length of X . □

The cut elimination algorithm follows the one given in [13]. Principal cut for $|$ (see Fig. 1 for the case where the Z in $|L$ is non-empty and Fig. 2 for the case where it is empty) apparently poses a problem since it may replace one cut by three cuts of a higher degree. Nevertheless it can be shown that the algorithm always terminates, for the following reason: We call a proof *special* iff all atoms occurring in it are pairwise different unless a sequent rule requires them to be identical. Clearly every proof can be transformed into a special proof by renaming of atoms. If we restrict our attention to special proofs, the principal cut for $|$ reduces the total number of atoms occurring in the proof (since p completely disappears). This parameter is not increased by any other cut elimination step. This guarantees that cut elimination eventually terminates. The restriction to special proofs is no real restriction since we can always transform a given proof into a special proof via renaming, perform cut elimination and finally reverse the renaming. □

$$\begin{array}{c}
\frac{\Pi_1}{B, p, W \Rightarrow B \bullet p \bullet A} \quad \frac{\Pi_2}{B, W \Rightarrow B \bullet W} \quad \frac{\Pi_3}{Y \Rightarrow B} \quad \frac{\Pi_4}{X, B, Z, A, U \Rightarrow C} \\
\hline
\frac{W \Rightarrow A|B}{X, Y, Z, W, U \Rightarrow C} \text{ } |R \quad \frac{X, Y, Z, A|B, U \Rightarrow C}{X, Y, Z, W, U \Rightarrow C} \text{ } |L \\
\hline
X, Y, Z, W, U \Rightarrow C \quad \text{Cut} \\
\sim \\
\frac{\Pi_3}{Y \Rightarrow B} \quad \frac{\Pi_1[p \leftarrow Z^\bullet]}{B, Z^\bullet, W \Rightarrow B \bullet Z^\bullet \bullet A} \quad \frac{\Pi_4}{X, B, Z, A, U \Rightarrow C} \bullet L \\
\hline
\frac{Y, Z^\bullet, W \Rightarrow B \bullet Z^\bullet \bullet A}{X, Y, Z^\bullet, W, U \Rightarrow C} \text{ } Cut \quad \frac{\vdots}{X, B \bullet Z^\bullet \bullet A, U \Rightarrow C} \bullet L \\
\hline
\frac{Z \Rightarrow Z^\bullet}{X, Y, Z, W, U \Rightarrow C} \text{ } lm \ 2 \quad \frac{X, Y, Z^\bullet, W, U \Rightarrow C}{X, Y, Z, W, U \Rightarrow C} \text{ } Cut \\
\hline
X, Y, Z, W, U \Rightarrow C \quad \text{Cut}
\end{array}$$

Fig. 1. Principal cut for $|$, Z non-empty

$$\begin{array}{c}
\frac{\Pi_1}{B, p, W \Rightarrow B \bullet p \bullet A} \quad \frac{\Pi_2}{B, W \Rightarrow B \bullet W} \quad \frac{\Pi_3}{Y \Rightarrow B} \quad \frac{\Pi_4}{X, B, A, U \Rightarrow C} \\
\hline
\frac{W \Rightarrow A|B}{X, Y, W, U \Rightarrow C} \text{ } |R \quad \frac{X, Y, A|B, U \Rightarrow C}{X, Y, W, U \Rightarrow C} \text{ } |L \\
\hline
X, Y, W, U \Rightarrow C \quad \text{Cut} \\
\sim \\
\frac{\Pi_3}{Y \Rightarrow B} \quad \frac{\Pi_2}{B, W \Rightarrow B \bullet A} \quad \frac{\Pi_4}{X, B, A, U \Rightarrow C} \\
\hline
\frac{Y, W \Rightarrow B \bullet A}{X, Y, W, U \Rightarrow C} \text{ } Cut \quad \frac{X, B \bullet A, U \Rightarrow C}{X, Y, W, U \Rightarrow C} \bullet L \\
\hline
X, Y, W, U \Rightarrow C \quad \text{Cut}
\end{array}$$

Fig. 2. Principal cut for $|$, Z empty

Although $|R$ lacks the subformula property, every sequent rule except cut increases complexity in the sense defined below:

Definition 1

1. $d(p) = 1$ with p atomic
2. $d(A \circ B) = d(A) + d(B) + 1$, where \circ ranges over \bullet, \setminus and $/$
3. $d(A|B) = d(A) + 2d(B) + 5$
4. $d(A_1, \dots, A_n \Rightarrow B) = d(A_1) + \dots + d(A_n) + d(B)$

Theorem 2 The proof search space for $\mathbf{L}|$ is finite.

Proof. Cut free proof search reduces complexity, and at every point in proof search, only finitely many sequent rules are applicable. \square

Corollary 1 (Decidability) $\mathbf{L|}$ is decidable.

Proof. Immediately from Theorem 2. \square

Corollary 2 (Finite Reading Property) $\mathbf{L|}$ has the finite reading property.

Proof. Immediately from Theorem 2. \square

2.3 Natural Deduction

The sequent system is indispensable since it guarantees decidability, but for practical purposes it is rather awkward. A presentation in natural deduction (ND) format is better suited to present concrete derivations. Besides, it has an appealing allusion to the tree format linguists are used to.

We start with a sequent style presentation of the natural deduction system (Fig. 3). Besides the identity rule and the cut rule (which are identical to the corresponding rules in the sequent system and therefore omitted), we have an introduction rule and an elimination rule for each connective.

$$\begin{array}{c}
 \frac{}{x : A, y : B \Rightarrow \langle x, y \rangle : A \bullet B} \bullet I \quad \frac{X \Rightarrow M : A \bullet B \quad Y, x : A, y : B, Z \Rightarrow N : C}{Y, X, Z \Rightarrow N[x \leftarrow (M)_0][y \leftarrow (M)_1] : C} \bullet E \\
 \\
 \frac{x : A, X \Rightarrow M : B}{X \Rightarrow \lambda x. M : A \setminus B} \setminus I \quad \frac{X \Rightarrow M : A \quad Y \Rightarrow N : A \setminus B}{X, Y \Rightarrow (NM) : B} \setminus E \\
 \\
 \frac{X, x : A \Rightarrow M : B}{X \Rightarrow \lambda x. M : B/A} /I \quad \frac{X \Rightarrow M : A/B \quad Y \Rightarrow N : B}{X, Y \Rightarrow (MN) : A} /E \\
 \\
 \frac{X \Rightarrow M : A \quad Y \Rightarrow N : B \quad Z \Rightarrow O : C|A}{X, Y, Z \Rightarrow \langle M, N, (OM) \rangle : A \bullet B \bullet C} |E \\
 \\
 \frac{x : B, y : p, X \Rightarrow \langle N, y, M \rangle : B \bullet p \bullet A \quad X : B, X \Rightarrow \langle N, M \rangle : B \bullet A}{X \Rightarrow \lambda x. M : A|B} |I \\
 \\
 p \text{ not occurring in } A, B, X \\
 M =_{\alpha\beta\eta} x
 \end{array}$$

Fig. 3. Natural Deduction $\mathbf{L|}$

Natural deductions are more conveniently carried out in tree form. The building blocks are given in Fig. 4. Note that a complete deduction always ends in a single conclusion, despite the fact that \bullet elimination and $|$ elimination have

multiple conclusions. For simplicity we combined the \mid -elimination rule with two subsequent applications of the \bullet -elimination rule, thus removing the products in the conclusion of \mid -elimination. The parentheses in the premises of \mid introduction indicate that these premises must be derivable both with and without the material in parentheses. Note that the only structural constraint on anaphora

$$\begin{array}{c}
\frac{M : A \quad N : B}{\langle M, N \rangle : A \bullet B} \bullet I \qquad \frac{M : A \bullet B}{(M)_0 : A \quad (M)_1 : B} \bullet E \\[20pt]
\frac{\frac{\frac{\overline{x : A} \quad i}{\vdots} \quad \vdots}{M : B} \setminus I, i}{\lambda x. M : A \setminus B} \qquad \frac{M : A \quad N : A \setminus B}{(NM) : B} \setminus E \\[20pt]
\frac{\frac{\frac{\vdots \quad \overline{x : A} \quad i}{\vdots} \quad \vdots}{M : B} / I, i}{\lambda x. M : B / A} \qquad \frac{M : A / B \quad N : B}{(MN) : A} / E \\[20pt]
\frac{\frac{\frac{\overline{x : B} \quad i \quad \overline{(y : p)} \quad i}{\vdots} \quad \vdots \quad \vdots}{\langle x, (y,)M \rangle : B \bullet (p \bullet) A} \mid I, i}{\lambda x. M : A \mid B} \qquad \frac{M : B \quad \cdots \quad N : A \mid B}{M : B \quad \cdots \quad (NM) : A} \mid E
\end{array}$$

Fig. 4. Natural deduction in tree format

resolution is the requirement that the antecedent precede the anaphor. No command relations of whatever kind are involved.

For better readability and to stress the similarity to conventional coindexing of constituents, we simplify the notation for $\mid E$ somewhat (see Fig. 5). When

$$[M : B]_i \quad \cdots \quad \frac{[N : A \mid B]_i}{(NM) : A} \mid E$$

Fig. 5. Simplified notation for $\mid E$

$$\frac{\frac{\frac{}{[x : A]_i} 1}{\langle x, y \rangle : A \bullet B} \bullet I \quad \frac{\frac{}{[z : C|A]_i}}{(zx) : C} |E}{\frac{\lambda x. \langle x, y \rangle : A \setminus (A \bullet B) \quad \setminus I, 1}{\langle \lambda x. \langle x, y \rangle, (zx) \rangle : (A \setminus (A \bullet B)) \bullet C} \bullet I}$$

Fig. 6. An illicit Natural Deduction derivation

working with Natural Deduction in tree format, it has to be kept in mind that the domain of rule applications are complete proof trees, not arbitrary subtrees of a proof tree. This is particularly important when $|E$ is involved. Both parts of an anaphoric link belong to one and the same tree. Therefore it is illicit to let another rule operate on a subtree that includes one part of the anaphoric link and excludes the other. (An example of a violation of this constraint is given in Fig. 6. Here the premise $x : A$ is connected with the premise $z : C|A$ by an anaphoric link, i.e an application of $|E$, but the scope of $\setminus I$ includes the former and excludes the latter.) This blocks derivations where the proof term of the conclusion contains free variables that do not correspond to any premise.

3 Pronouns and Quantification

Following Jacobson, we assume that pronouns like *he* have category $N|N$ and denote the identity function on individuals, i.e the associated semantic term is $\lambda x.x$. For a simple example like

(2) John said he walked

where the only potential antecedent of the pronoun is a proper noun, we have the two possible derivations shown in Fig. 7, corresponding to the coreferential and the free reading of the pronoun.

Things become somewhat more involved when we consider possible interaction of anaphora resolution with hypothetical reasoning. Nothing prevents us from using a hypothesis of the appropriate type as antecedent for anaphora resolution. For example, in the VP

(3) said he walked

the pronoun can be linked to the subject argument place of the VP, as Fig. 8 demonstrates. This VP can for instance be combined with a subject relative pronoun to yield the relative clause *who said he walked*. Another type of construction where this kind of derivation is crucial are sloppy readings of VPE that will be discussed below.

Binding to hypothetical antecedents is not restricted to slash introduction rules. Another obvious case in point is the interaction of anaphora with quantification. Here we adopt the type logical treatment of quantification that was

$$\begin{array}{c}
\frac{x : q(A, B, C)}{y : A} i \\
\vdots \\
\vdots \\
\vdots \\
\hline
\alpha : B \\
\hline
x(\lambda y. \alpha) : C \quad qE, i
\end{array}
\qquad
\frac{x : A}{\lambda y. yx : q(A, B, B)} qI$$

Fig. 9. Natural Deduction rules for $q(A, B, C)$

Quantifiers like *everybody* have category $q(N, S, S)$, so in the course of scop-ing the quantifier, a hypothesis of category N is temporarily introduced. This hypothesis can in turn serve as antecedent of *his*, as illustrated in Fig. 10.

$$\begin{array}{c}
\frac{\text{everybody}}{\text{EVERY} : q(N, S, S)} lex \\
\hline
[x : N]_i \quad 1 \\
\hline
\frac{\text{said}}{\text{SAY} : N \setminus S/S} lex \quad \frac{\text{he}}{[\lambda x. x : N|N]_i} lex \quad \frac{\text{walked}}{\text{WALK} : N \setminus S} lex \\
\hline
\frac{\text{SAY}(\text{WALK } x) : N \setminus S}{\text{SAY}(\text{WALK } x)x : S} \setminus E \quad \frac{\text{WALK } x : S}{\text{WALK} : N \setminus S} \setminus E \\
\hline
\frac{\text{SAY}(\text{WALK } x)x : S}{\text{EVERY}(\lambda x. \text{SAY}(\text{WALK } x)x) : S} qE, 1
\end{array}$$

Fig. 10. Derivation of *Everybody said he walked*

If we reverse the order of the quantifier and the pronoun as in (5), the derivation of a bound reading will fail, even though the pronoun is in the scope of the quantifier.

(5) *He_i said everybody_i walked

This configuration—a Strong Crossover violation—is ruled out since the hypothesis that temporarily replaces the quantifier does not precede the pronoun. Thus \setminus -elimination cannot be applied.

As any ND rule, q -elimination can only be applied to complete trees. If the hypothetical N that is used in qE serves as the antecedent of a pronoun, this pronoun must be in the scope of qE . Linguistically speaking, this means that a bound pronoun is always in the scope of its binder. This excludes for instance a wide scope reading of the indefinite object in (6) if the pronoun is bound by the subject.

(6) Every man saw a friend of his

The way the present system excludes such readings is similar to the one proposed in [19], even though the treatment of pronouns in general is different.

Finally it should be stressed that the only constraints on pronoun binding here are the requirements that 1. the quantifier precedes the pronoun, and 2. the pronoun is in the scope of the quantifier. So the derivation of a construction like (7), where the binding quantifier does not c-command the pronoun under the standard conception of constituency, does not differ substantially from the previous example (Fig. 11).

(7) Everybody's mother loves him

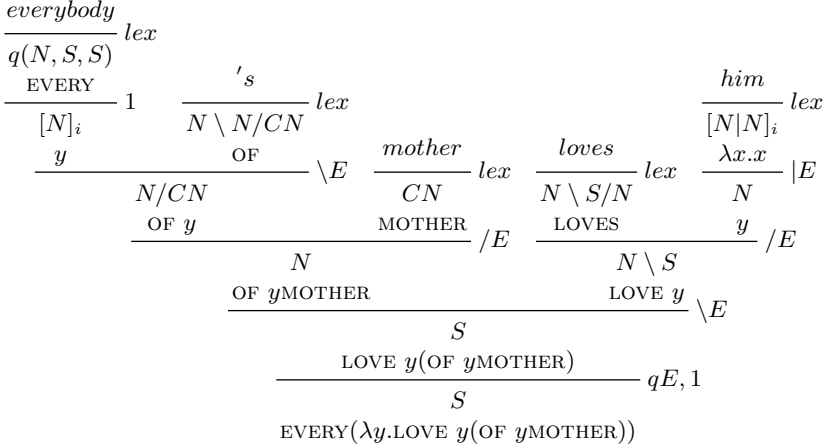


Fig. 11. Everybody's mother loves him

Again, if we change the order of pronoun and quantifier, the derivation will fail since the precedence requirement for $|E$ is not met.

(8) *His mother loves everybody

So the precedence requirement also accounts for Weak Crossover violations.

As mentioned above, binding to hypothetical antecedents is not restricted to quantification. Another obvious case in point is *wh*-movement. There are several proposals for a type logical treatment of this complex around in the literature (see for instance [14,18]). Despite the differences in detail, they share the assumption that a hypothesis is put into the “base position” of *wh*-movement that is later discharged and bound by the operator. So we correctly predict the same patterns concerning the interaction of binding and scope and with respect to Crossover phenomena. Arguably, association with focus involves hypothetical reasoning as well (see [10] for an attempt to spell this idea out in a multi-modal framework). Accordingly, we find bound readings and Crossover effects if the antecedent of a pronoun is focused. (The latter observation was initially made in [1]. Example (10) is from [20].)

- (9) Only JOHN hates his mother
- (10) a. We only expect HIM to be betrayed by the woman he loves
 b. We only expect him to be betrayed by the woman HE loves

The pronoun in sentence (9) can be bound, i.e. the sentence can mean *John is the only x such that x loves x 's mother*. Example (10) illustrates that binding by focus displays Weak Crossover effects. Sentence (10a) has a reading saying that the referent of *him* is the only person x such that we expect x to be betrayed by the woman x loves. No such reading is available in (10b).

4 VP Ellipsis

This treatment of anaphora can straightforwardly be extended to VPE. Ignoring matters of tense and mood, we treat the stranded auxiliary in the second conjunct of constructions like (11) as a proform for VPs.

- (11) John walked, and Bill did too

So *did* will be assigned the category $(N \setminus S)|(N \setminus S)$ and the meaning $\lambda P.P$, i.e. the identity function on properties. The derivation for (11) is given in Fig. 12 (we also ignore the contribution of *too* since it is irrelevant for the semantics of VPE, though not for the pragmatics).

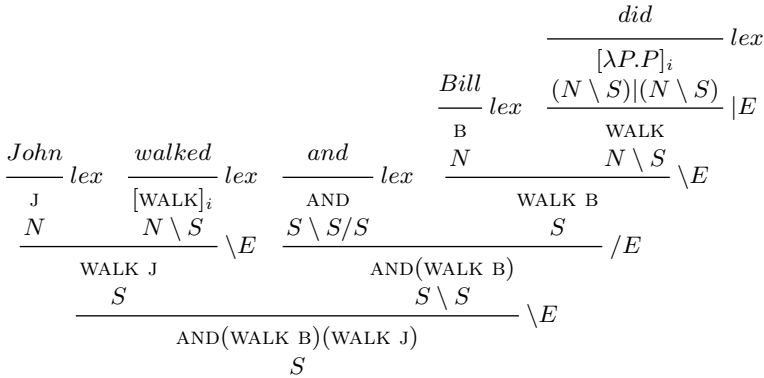


Fig. 12. John walked, and Bill did (too)

What makes VPE an interesting topic is of course its complex interaction with pronominal anaphora and quantification. Due to limitation of space, we cannot give an in-depth investigation of these issues here. Instead we will content ourselves with a discussion of some of the most frequently discussed examples from the literature.

The first non-trivial issue in this connection is the well-known strict/sloppy ambiguity in constructions like (12).

(12) John revised his paper, and Harry did too

The crucial step for the derivation of the sloppy reading is already given for an analogous example in Fig. 8: the pronoun is bound to the subject argument place of the source VP. From this we can continue the derivation completely in parallel to Fig. 12, and we end up with the meaning $\text{AND}(\text{SAY}(\text{WALK B})\text{B})$ ($\text{SAY}(\text{WALK J})\text{J}$). Crucially, here the pronoun was bound by a hypothetical antecedent. Of course it is also licit to bind the pronoun to the actual subject *John* and then doing ellipsis resolution, which results in the strict reading. The derivation of both readings is given in Fig. 13.

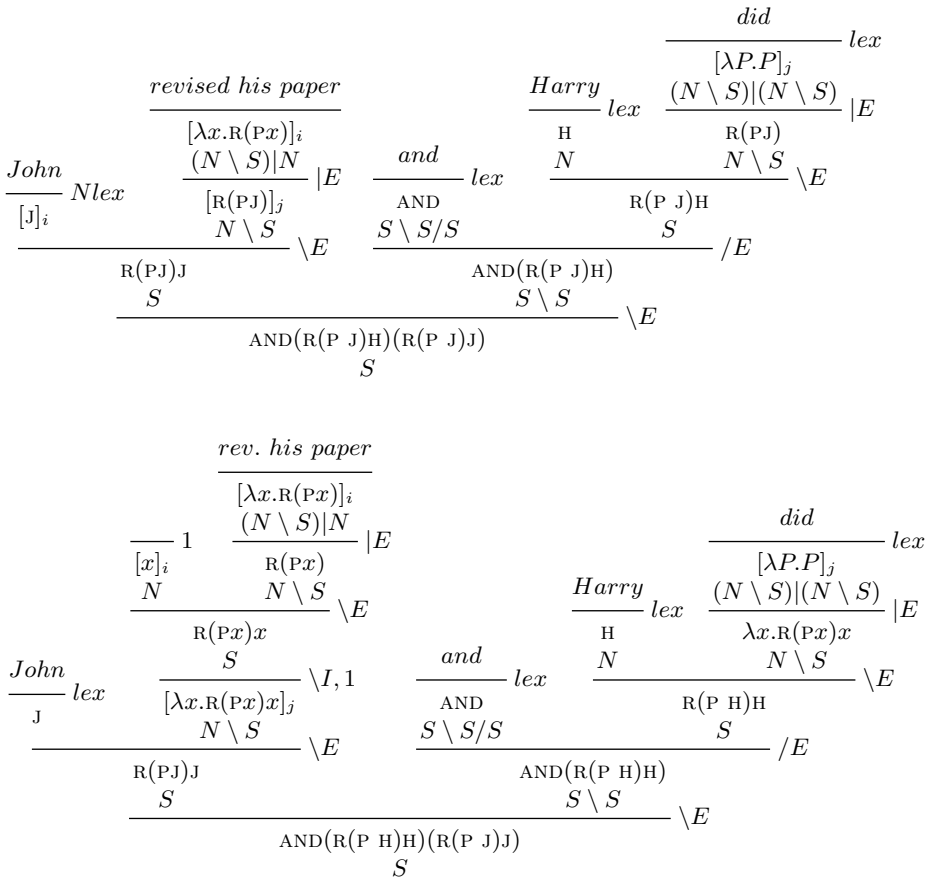


Fig. 13. Derivation of the strict and the sloppy reading of (12)

Next we would like to draw attention to a kind of ambiguity that arises from the interplay of quantification and VPE. Consider the following example.

- (13) a. John met everybody before Bill did
b. John met everybody before Bill met everybody
c. John met everybody before Bill met him

As Ivan Sag observed in [21], constructions like (13a) are ambiguous between a reading synonymous to (13b) and one synonymous to (13c). Under the present approach, reading (13b) arises if the quantifier is scoped before ellipsis resolution takes place. If scoping is postponed until after ellipsis resolution, the antecedent of the ellipsis still contains a hypothetical N , and accordingly the quantifier binds two occurrences of the corresponding variable. Fig. 14 gives the derivations for the source VPs of the two readings.

$$\begin{array}{c}
\frac{\text{everybody}}{q(N, S, S)} lex \\
\frac{\frac{met}{N \setminus S/N} lex \quad \frac{\text{EVERY}}{N} 2}{\frac{\text{MEET}}{y} / E} \\
\frac{\frac{-1}{N} \quad \frac{\text{MEET} y}{N \setminus S}}{S} \setminus E \\
\frac{\text{MEET} y x}{S} qE, 2 \\
\frac{\text{EVERY}(\lambda y. \text{MEET} y x)}{[N \setminus S]_i} \setminus I, 1 \\
\lambda x. \text{EVERY}(\lambda y. \text{MEET} y x)
\end{array}
\qquad
\begin{array}{c}
\frac{\text{everybody}}{q(N, S, S)} lex \\
\frac{\frac{met}{N \setminus S/N} lex \quad \frac{\text{EVERY}}{N} 1}{\frac{\text{MEET}}{x} / E} \\
[N \setminus S]_i \\
\text{MEET} x
\end{array}$$

Fig. 14. Source VPs in (13a)

Again this phenomenon is not restricted to quantification. Whenever the derivation of the source VP involves hypothetical reasoning and it is possible to discharge the hypothesis after ellipsis resolution, multiple binding should be possible. This is in fact the case. For *wh*-movement, this was also observed in [21].

- (14) a. (the man) that Mary met before Bill did
b. How many miles are you prepared to walk if the people want you to

The preferred reading of (14a) is *the man that Mary met before Bill met him*. Example (14b) is similar. Additionally it illustrates that this mechanism is not restricted to hypotheses of type *N*, and that constructions with “binding into ellipsis” need not correspond to a parallel construction without ellipsis and with a pronoun.

Last but not least, we encounter the same pattern in connection with focus. This was first noted [12], where the following example is given:

(15) I only went to TANGLEWOOD because you did

The sentence has a reading saying: The only place x such that I went to x because you went to x is Tanglewood.

Let me close this section with an example from [2] that demonstrates that the ambiguity of bound versus coreferential interpretation of pronouns on the one hand and the strict/sloppy ambiguity on the other hand are independent phenomena:

- (16) a. Every student revised his paper before the teacher did
 b. Every student _{i} revised his _{j} paper before the teacher _{k} revised his _{j} paper
 c. Every student _{i} revised his _{i} paper before the teacher _{j} revised his _{j} paper
 d. Every student _{i} revised his _{i} paper before the teacher _{j} revised his _{i} paper

Sentence (16a) has three readings (paraphrased in (16b-d)). Next to the unproblematic cases where the pronoun is either free and strict (b) or bound and sloppy (c), there is an interpretation where the pronoun is bound but nevertheless strict (d). Gawron and Peters therefore assume a three-way ambiguity of pronoun uses—referential as in (b), role-linking as in (c), and co-parametric as in (d) (cf. [2]).

In the present systems, all three readings fall out immediately, even though the pronoun is unambiguous. If the pronoun is free, the derivation is analogous to Fig. 7. Readings (16c,d) are derived by first plugging in a hypothetical N into the matrix subject position, giving the ellipsis a sloppy or strict construal (as in Fig. 13), and applying qE and thus replacing the hypothetical N by the quantifier.

5 Comparison to Jacobson’s System

Despite the overall similarity between Jacobson’s and the present treatment of anaphora, there are two important differences. As can be seen from rule $|R$, according to our proposal a pronoun is licensed primarily by a preceding antecedent. This antecedent may be hypothetical. In this case, the pronoun may be linked to an argument place of a super-ordinate functor. This option is employed in the derivation of bound and sloppy readings. Jacobson takes this method of binding to be basic. Due to the lack of unrestricted associativity in CCG, the restriction to super-ordinate functors is non-trivial here. This aspect of her system leads to two shortcomings that can be avoided in the type logical setting.¹

First, not all bound pronouns can be treated in this fashion. In (17) (from [2]), there is no constituent that contains the pronoun and takes its antecedent as an argument (even under the flexible notion of constituency that CCG adopts).

(17) The soldiers turned some citizens in [each state] _{i} over to its _{i} governor

¹ The same objections apply to the proposals in [5] and [22] as well.

In the type logical formulation presented here, the discontinuity of the main verb may require an involved treatment, but since the antecedent precedes the pronoun, there is no problem with anaphora resolution.

As for the second point, reconsider the strict reading of (12). The only way to give the pronoun a bound reading under the combinatory approach is to bind it to the subject argument place of the verb *revised*. But this means that the property *to revise John's paper* doesn't occur as the meaning of any constituent. We only get the meaning *to revise one's paper* as the meaning of the source VP. So if we assume an identity-of-meaning approach to ellipsis, we only get the sloppy reading here if the pronoun is construed as bound. The best we can do to derive the strict reading is to consider the pronoun as free and accidentally co-referring with *John*. But as (16) demonstrates, strict readings of bound pronouns are possible. The combinatory treatment of anaphora cannot handle constructions like this one.

6 Cross-Sentential Anaphora

To extend the type logical approach to grammar to the discourse level, we have to introduce a new type, call it D , for discourses. Besides, type assignment has to guarantee at least that every sentence is a discourse, and that appending a sentence to a discourse yields a discourse again. So the following two sequents should be theorems of \mathbf{L} :

- (18) a. $S \Rightarrow D$
 b. $D, S \Rightarrow D$

Clearly these cannot be derivable sequents if S and D are atomic. So we have to replace them by suitable complex types. Two options suggest themselves: both S and D should be identified either with $I \setminus I$ or I/I for some type I .² They both have a dynamic flavor: $I \setminus I$ is akin to the view of Discourse Representation Theory, File Change Semantics or Update Semantics (cf. [4,11,24]), where a sentence defines a function from information states to information states. I/I resembles DMG (Dynamic Montague Grammar, cf. [3]), where a sentence meaning is a function from possible continuations to “static” sentence meanings. I'll adopt the latter, DMG-style option here. The semantic type corresponding to I is t (or $\langle s, t \rangle$ if we incorporate intensionality). A sentence like *John walks* will receive the DMG-style meaning $\lambda p(\text{WALK } J \wedge p)$. Now consider a sample discourse like

- (19) John walked. He talked

If S is uniformly replaced by I/I in the type assignments, the relevant sequent becomes

² To be precise, S should be identified with $\Box^+(I/I)$ or $\Box^+(I \setminus I)$ for some domain modality \Box^+ in the sense of [17] to take the special status of sentences into account. I'll ignore this issue here.

$$(20) \ N, N \setminus (I/I), N|N, N \setminus (I/I) \Rightarrow I/I$$

Following the DMG philosophy again, we assume that verbs like *walk* and *talk* denote dynamic properties, i.e. $\lambda x p.((\{w/T\} \text{ALK } x) \wedge p)$. Fig. 15 shows that the sequent is valid and that the derived meaning is

$$(21) \ \lambda p.((\text{WALK } J) \wedge (\text{TALK } J) \wedge p)$$

The static truth conditions are obtained by applying this to the tautology.

$$\begin{array}{c}
 \frac{\frac{\frac{John}{[N]_i} lex \quad \frac{walked}{N \setminus I/I} lex}{J \quad \lambda x p. \text{WALK } x \wedge p} \setminus E \quad \frac{\frac{\frac{He}{[N|N]_i} lex \quad \frac{talked}{N \setminus I/I} lex}{N \quad J \quad \lambda x p. \text{TALK } x \wedge p} \setminus E \quad \frac{1}{I}}{I/I} \setminus E \\
 \frac{\lambda p. \text{WALK } J \wedge p \quad \lambda p. \text{TALK } J \wedge p}{I} /E \\
 \frac{I}{\text{WALK } J \wedge \text{TALK } J \wedge p} /I, 1 \\
 \frac{I/I}{\lambda p. \text{WALK } J \wedge \text{TALK } J \wedge p}
 \end{array}$$

Fig. 15. John walked. He talked

This treatment can easily be extended to indefinites. Let us assume that an indefinite like *someone* has category $q(N, I, I)$ and meaning $\lambda P. \exists x P x$.

$$(22) \ \text{Someone walked. He talked.}$$

As Fig. 16 shows, a discourse like (22) receives type I/I and the meaning $\lambda p. \exists x ((\text{WALK } x) \wedge (\text{TALK } x) \wedge p)$. The mechanism of cross-sentential/dynamic binding is thus essentially the same as for sentence internal binding. Other quantifiers like *every man* can be prevented from taking discourse scope by the same multi-modal mechanisms that block them from outscoping *and* in coordinate structures (see for instance [18]).

7 Incremental Interpretation

This treatment of cross-sentential anaphora is compatible with the intuitive requirement that discourse interpretation works incrementally. Note that in all compositional theories of dynamic semantics, the meaning of a sentence includes

$$\begin{array}{c}
\frac{\text{someone}}{q(N, I,)} \text{lex} \\
\frac{\lambda P. \exists x P x}{\lambda P. \exists x P x} 1 \\
\frac{[N]_i}{y} \\
\frac{\text{walked}}{N \setminus I/I} \text{lex} \\
\frac{\lambda x p. \text{WALK} x \wedge p}{\lambda x p. \text{WALK} x \wedge p} \setminus E \\
\frac{I/I}{\lambda p. \text{WALK} y \wedge p} \\
\frac{I}{\text{WALK } y \wedge \text{TALK } y \wedge p} qE, 1 \\
\frac{I}{\exists x (\text{WALK } x \wedge \text{TALK } x \wedge p)} /I, 2 \\
\frac{I/I}{\lambda p. \exists x (\text{WALK } x \wedge \text{TALK } x \wedge p)}
\end{array}
\quad
\begin{array}{c}
\frac{He}{[N|N]_i} \text{lex} \\
\frac{\lambda x. x}{\lambda x. x} |E \\
\frac{N}{y} \\
\frac{\text{talked}}{N \setminus I/I} \text{lex} \\
\frac{\lambda x p. \text{TALK} x \wedge p}{\lambda x p. \text{TALK} x \wedge p} \setminus E \\
\frac{I/I}{\lambda P. \text{TALK } y \wedge p} \setminus E \\
\frac{I}{\text{TALK } y \wedge p} /E \\
\frac{I}{\text{WALK } y \wedge \text{TALK } y \wedge p} qE, 1 \\
\frac{I}{\exists x (\text{WALK } x \wedge \text{TALK } x \wedge p)} /I, 2 \\
\frac{I/I}{\lambda p. \exists x (\text{WALK } x \wedge \text{TALK } x \wedge p)}
\end{array}$$

Fig. 16. Someone walked. He talked

information about how many old discourse markers are picked up, and how many novel discourse markers are introduced. Under the type logical perspective, this information has to be encoded in the type of a sentence. So we admit a limited polymorphism for the category of sentences. To take two simple examples, *A man walks* will have (among others) the type $I/(I|N)$ since it licenses a subsequent pronoun. *He walks* will have type $(I/I)|N$, indicating that it contains an old discourse referent that needs an antecedent. Generally, we assume that a sentence containing n (locally unbound) pronouns and introducing m discourse entities will have category $(I/(I(|N)^m))(|N)^n$. Semantically, such a sentence will denote a function from n individuals and an m -place relation to a proposition. Sentence concatenation corresponds to a family of generalized versions of function composition, where each argument place of the second sentence may either be filled by one of the discourse markers introduced by the first sentence, or projected to the discourse as a whole. [23] briefly considers a proposal similar to this, but rejects it due to the apparent proliferations of sentence combining operations. This is no obstacle here though, since all these operations are derivable in $\mathbf{L}_|$.

8 Future Work

Two directions for further research suggest themselves. In its present shape, the system cannot cope with cataphora. This phenomenon is very limited and many cases should arguably be treated as accidental coreference instead as a grammatical dependency (cf. [25] for an insightful discussion), but there are undeniable cases of grammatically determined backward binding.

Furthermore, it is yet unclear how other types of ellipsis should be incorporated. VP ellipsis is simple insofar as it leaves a proform, the stranded auxiliary. Insofar as ellipsis resolution is rooted in the lexicon here. This is not the case with gapping, stripping etc. So apparently an adequate extension of $\mathbf{L}|$ has to include devices to create anaphoric types in syntax. In other words, $\mathbf{L}|$ is still too weak for a treatment of ellipsis in general. Any strengthening of the logic is in risk of losing the finite reading property though.

9 Conclusion

This paper proposed the logic $\mathbf{L}|$ as a type logical reconstruction of Pauline Jacobson's treatment of anaphora in CG. It was shown that $\mathbf{L}|$ is weak enough to have the finite reading property, but strong enough to handle the multiplication of resources that we find in anaphoric dependencies. Paired with Moortgat's type logical approach to quantification, we are able to cope with a substantial amount of phenomena concerning pronominal anaphora, VP ellipsis and quantification. Finally it was sketched how cross sentential anaphora can be handled under this approach.

References

1. Chomsky, N. Conditions on rules in grammar. *Linguistic Analysis*, 2:303–351, 1976.
2. Gawron, J. M., and S. Peters. *Anaphora and Quantification in Situation Semantics*. CSLI, Stanford, 1990.
3. Groenendijk, J., and M. Stokhof. Dynamic Montague Grammar. In Jeroen Groenendijk, Martin Stokhof, and David Ian Beaver, editors, *Quantification and Anaphora I*, DYANA deliverable R2.2a. Amsterdam, 1991.
4. Heim, I. *The Semantics of Definite and Indefinite Noun Phrases*. PhD thesis, University of Massachusetts, Amherst, 1982.
5. Hepple, M. *The Grammar and Processing of Order and Dependency: A Categorical Approach*. PhD thesis, University of Edinburgh, 1990.
6. Jacobson, P. Bach-Peters sentences in a variable-free semantics. In Dekker, P., and M. Stokhof, editors, *Proceedings of the Eighth Amsterdam Colloquium*. University of Amsterdam, 1992.
7. Jacobson, P. Binding connectivity in copular sentences. In Harvey, M., and L. Santelmann, editors, *Proceedings of SALT IV*, pages 161–178. Cornell University, 1994.
8. Jacobson, P. i-within-i effects in a variable-free semantics and a categorial syntax. In Dekker, P., and M. Stokhof, editors, *Proceedings of the Ninth Amsterdam Colloquium*. University of Amsterdam, 1994.
9. Jacobson, P. Towards a variable-free semantics. *Linguistics and Philosophy*, 22(2):117–184, 1999.
10. Jäger, G. Focus without variables: A multi-modal analysis. In Shahin, K. N., S. Blake, and E.-S. Kim, editors, *Proceedings of WCCFL 17*. CSLI Publications, 1999.
11. Kamp, H., and Reyle, U. *From Discourse to Logic. Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer, Dordrecht, 1993.

12. Kratzer, A. The representation of focus. In Arnim von Stechow and Dieter Wunderlich, editors, *Handbook Semantics*. de Gruyter, Berlin, New York, 1991.
13. Lambek, J. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.
14. Moortgat, M. *Categorical Investigations. Logical and Linguistic Aspects of the Lambek Calculus*. Foris, Dordrecht, 1988.
15. Moortgat, M. Generalized quantification and discontinuous type constructors. In Sijtsma, W., and A. von Horck, editors, *Discontinuous Constituency*. De Gruyter, Berlin, 1996.
16. Moortgat, M. In situ binding: A modal analysis. In Dekker, P., and M. Stokhof, editors, *Proceedings of the 10th Amsterdam Colloquium*, pages 539–549. ILLC, University of Amsterdam, 1996.
17. Moortgat, M. Multimodal linguistic inference. *Journal of Logic, Language and Information*, 5(3/4):349–385, 1996.
18. Morrill, G. *Type Logical Grammar*. Kluwer, 1994.
19. Pereira, F. C. N. Categorical semantics and scoping. *Computational Linguistics*, 16(1):1–10, 1990.
20. Rooth, M. *Association with Focus*. PhD thesis, University of Massachusetts, Amherst, 1985.
21. Sag, I. A. *Deletion and Logical Form*. PhD thesis, MIT, 1976.
22. Szabolcsi, A. *Bound Variables in Syntax. (Are there any?)*. Institute of Linguistics, Hungarian Academy of Science, Budapest, 1988.
23. Szabolcsi, A. Reconstruction, anaphora, and pronouns as identity maps. In Dekker, P., M. Stokhof, and Y. Venema, editors, *Proceedings of the 11th Amsterdam Colloquium*. University of Amsterdam, 1997.
24. Veltman, F. Defaults in update semantics. *Journal of Philosophical Logic*, 25:221–261, 1996.
25. Williams, E. Blocking and anaphora. *Linguistic Inquiry*, 28(4):577–628, 1997.

An LTAG Perspective on Categorical Inference^{*}

Aravind K. Joshi¹, Seth Kulick¹, and Natasha Kurtonina²

¹ Department of Computer and Information Science
and Institute for Research in Cognitive Science

² Institute for Research in Cognitive Science

University of Pennsylvania

3401 Walnut Street, Suite 400A

Philadelphia, PA 19119

{joshi,skulick,natashak}@linc.cis.upenn.edu

Abstract. This paper describes a system of categorical inference based on insights from Lexicalized Tree Adjoining Grammar (LTAG). LTAG is a tree-rewriting system and therefore deals with structural, not string, adjacency. When looked at from the logic perspective, the nodes of the trees become types as in a categorical grammar, with corresponding deductive connections between parent and daughter nodes. The resulting system is based on a hybrid logic, with one logic for building Partial Proof Trees, and the other for composing the partial proofs. We reexamine the use of structural modalities in categorical grammar from this perspective, concluding that the use of structural modalities can be considerably simplified, or even eliminated in some cases. The generative power of the hybrid logic system is beyond context-free, as we demonstrate with a derivation of the cross-serial dependencies in Dutch. The system also inherits polynomial parsing from LTAG.

1 Introduction

The purpose of this paper is to describe a perspective on categorical inference based on insights from Lexicalized Tree Adjoining Grammar (LTAG). LTAG is a tree-rewriting system and therefore deals with structural, and not string, adjacency. Being lexicalized, a structure is associated with each lexical item. When looked at from the logic perspective, the nodes of the trees become types as in a categorical grammar, with corresponding deductive connections between parent and daughter nodes.

The resulting categorical system has some advantages as compared with type-logical systems based on the Lambek calculus. First, it inherits polynomial parsing from LTAG (while complexity of parsing in Lambek grammar is not known yet). Second, resource management is localized to the domains of the elementary objects of the grammar. This allows a simplification of the use of structural

^{*} We would like to thank Gerhard Jaeger, Michael Moortgat, Richard Oehrle, Christian Retoré, and an anonymous reviewer for many valuable comments and discussion. This work was partially supported by NSF grant SBR8920230 and ARO grant DAAH0404-94-G-0426.

modalities because their use is localized to these domains. Third, key insights from LTAG are incorporated into a system of categorial inference—namely, the extended domain of locality and consequent factoring of recursion from the domain of dependencies. This means that in some cases the use of a structural modality can be eliminated, while in other cases, their role is translated into constraints on tree composition (stretching).

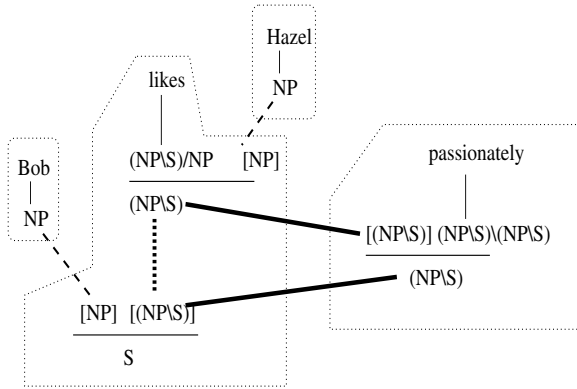


Fig. 1. Sample Partial Proof Tree Derivation

2 Partial Proof Trees

When viewed from the context of categorial grammar, LTAG can be seen as a system of partial proof trees (PPTs) (see [2] for details). The key idea is that instead of associating a type with each lexical item, we associate one or more partial proof trees, and each tree is obtained by unfolding the arguments of the type.

The basic PPTs then serve as the building blocks of the grammar, and complex proof trees are obtained by ‘combining’ these PPTs by two operations: substitution and stretching, illustrated in Figure 1. Substitution, shown by the dashed lines, makes use of the terminal node of a tree, such as substituting the PPT for *Bob* into an *np* node in the tree for *likes*, as shown. The second operation, stretching, provides access to an internal node in a tree, illustrated by the solid lines in the figure. Here the PPT for *passionately* is linked to the “stretching” of the internal *np*\s node in the *likes* tree.

As the figure illustrates, dependencies are represented in the elementary trees by the unfolding process. For example, while the NPs for *likes* are both clearly semantic arguments and so are unfolded, only the verb phrase, but not the noun phrase, is an argument of the adverb *passionately*, and so the tree unfolds only to the *np\s* level. Since we are using the Lambek calculus, function application and conditionalization both play an important role in the construction

of the PPTs. However, crucially, trace assumptions, once introduced, must be discharged within the same PPT in which they originate. Since, as will be discussed, such PPTs are of a small bounded size, this allows for a localization of resource management with corresponding computational advantages. The same advantages hold for the use of structural modalities, such as modally-licensed Permutation, which can be explored only within the scope of an elementary tree and therefore have a restricted capacity to influence the whole structure of generated strings.

The operations of substitution and stretching do not affect this localization. The important point is that constraints that license the basic PPTs are distinct from these operations that combine the PPTs.

3 Hybrid Logic

A logical modelling of the PPTs system makes use of a *hybrid* logic; i.e., two kinds of logic are involved. This is a consequence of LTAG being a tree-rewriting system. We distinguish the logic of constructing basic trees and combining trees. Construction of basic trees is guided by the logic of a CG, while both operations of combining trees (substitution and stretching) are encoded by a single rule: Cut. The logic of constructing the basic trees is based on the usual understanding of structure-sensitive consequence relations between formulas as *types*. The logic of combining trees, however, defines how some set of proofs can be transformed into another proof. Therefore the consequence relation of the logic of combining trees is defined on *proofs*. Before giving details of the system, we first discuss an illustrative example,¹ and then give a scheme of a general definition.

3.1 An Example

- (1) ...who Bob meets today

Consider the example (1) of a relative clause with an adverb, which is a classic example of the use of a Permutation modality in categorial grammar.² We will show that hybridization of the inference allows us to avoid the use of the structural modality.

The PPT system derivation is shown in Figure 2. This derivation takes advantage of the possibilities for conditionalization discussed above. As can be seen, the tree for *meets* assumes an NP assumption which is locally discharged. The sequents in (2) model the derivation of the first part of the *meets* tree. To avoid unnecessary detail, we are not discussing the latter part of the tree with the use of *who*.

- (2) a. $meets : (np \backslash s) / np, np \Rightarrow np \backslash s$
 b. $Bob : np, np \backslash s \Rightarrow s$

¹ With some abuse of notation, and ignoring substitution.

² The need for a permutation modality is discussed in more detail in Section 4.1.

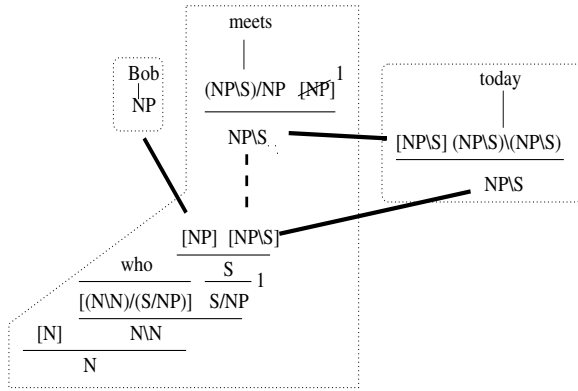


Fig. 2. Sample Partial Proof Tree Derivation with Conditionalization

- (3) a. $\frac{Bob\ meets\ np \Rightarrow s}{Bob\ meets \Rightarrow s/np}$
 d. $who\ Bob\ meets \Rightarrow n \setminus n$
 e. $n\ n \setminus n \Rightarrow n$
- (4) a. $meets\ np\ today \Rightarrow np \setminus s$
 b. $np\ np \setminus s \Rightarrow s$
 c. $\frac{Bob\ meets\ np\ today \Rightarrow s}{Bob\ meets\ today \Rightarrow s/np}$
 d. $who\ Bob\ meets\ today \Rightarrow n \setminus n$
 e. $n\ n \setminus n \Rightarrow n$

The sequent in (3) models the derivation for the *today* tree. It is important that every step of the derivation in (2) is presented to make internal nodes available for stretching.

The sequents in (4) illustrate how the second logic, a result of the hybridization, is used to combine the sequents in (2) and (3). The first step would be cut application with premises (2a) and (3a), resulting in (4a). The second step replaces *meets np* with *meets np today* everywhere in the derivation of (2), resulting in (4bcde).

Note that, crucially, the structure of (2) is not disturbed by this replacement. Therefore, while (4c) has the appearance of a violation of the Lambek calculus, since the hypothetical np is no longer on the right periphery of the expression in the presence of the adverb, this step is justified. The relations between the types in an elementary tree are fixed by the creation of the tree. Since the stretching process maintains the relations between the types, the non-peripheral extraction is legitimate due to the application of the second logic, which does not disturb the type relations in the elementary trees.

3.2 The Hybrid Logic

Figure 3 illustrates the logic used for constructing the basic PPTs. As mentioned, it is based on the usual understanding of structure-sensitive consequence relations between formulas as types.

$$\begin{array}{c}
 \text{Functional application} \\
 A, A \backslash B, \Rightarrow B \\
 B/A, A \Rightarrow B \\
 \text{Conditionalization} \\
 \frac{A, X \Rightarrow B}{X \Rightarrow A \backslash B} \\
 \frac{X, A \Rightarrow B}{X \Rightarrow B/A} \\
 \text{Cut} \\
 \frac{X \Rightarrow A \quad Y[A] \Rightarrow C}{Y[X] \Rightarrow C}
 \end{array}$$

Fig. 3. Logic of Constructing the Basic PPTs

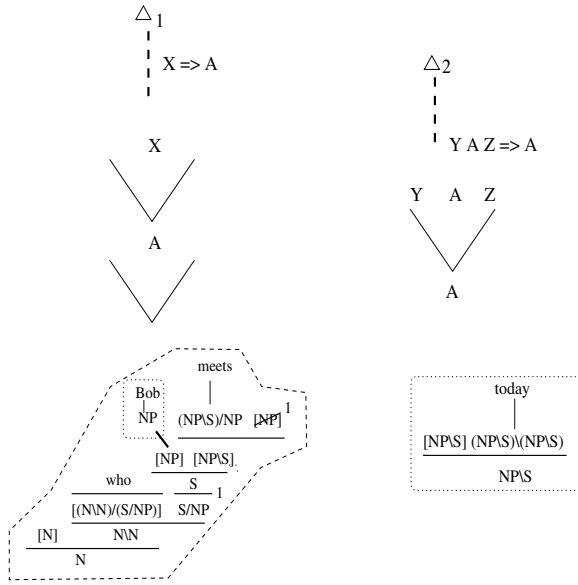
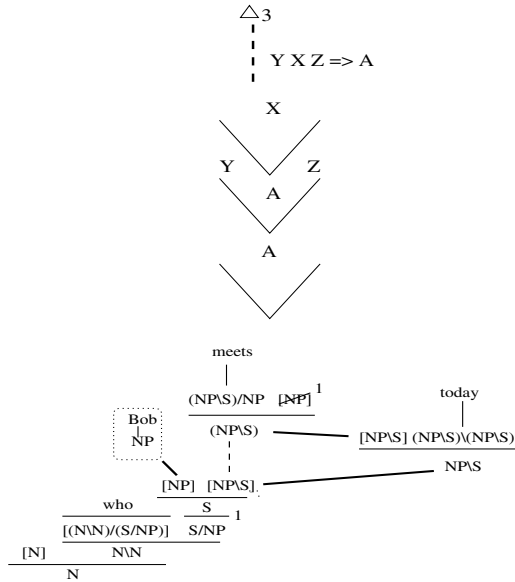
The logic for combining PPTs via stretching is as follows: Let Δ_1 be a proof containing (*) $X \Rightarrow A$ and Δ_2 be a proof containing (**) $Y A Z \Rightarrow A$ as the last sequent. Then a new proof Δ_3 contains all sequents preceding (*) and (**) in Δ_1 and Δ_2 respectively, a new sequent $Y X Z \Rightarrow A$, and all sequents of Δ_1 provided that X is replaced by $Y X Z$, with no change to the structure.

This is illustrated in Figures 4 and 5 for the example discussed in this section. In Δ_1 in Figure 4, X is *meets* : $(np \backslash s)/np \ np$ and A is $np \backslash s$ (and so $X \Rightarrow A$ is sequent (2a)). In Δ_2 in Figure 4, Y is empty, A is $np \backslash s$, and Z is *today* : $(np \backslash s) \backslash (np \backslash s)$ (and so $Y A Z$ is sequent (3a)). Therefore, Δ_3 in Figure 5 contains the new sequent $Y X Z \Rightarrow A$, or *meets* : $(np \backslash s)/np \ np \ today$: $(np \backslash s) \backslash (np \backslash s) \Rightarrow np \backslash s$, and all sequents of Δ_1 with *meets* : $(np \backslash s)/np \ np$ replaced by *meets* : $(np \backslash s)/np \ np \ today$: $(np \backslash s) \backslash (np \backslash s)$, with no change in structure to Δ_1 . This is the set of sequents in (4), with *meets np* replaced by *meets np today*.

This is some relation between this system and that of proposals such as those of [1] and [5]. The main difference is hybridization and the resulting stronger expressivity. Moreover, our system makes essential use of conditionalization.

4 Structural Control and Partial Proof Trees

It is well known that the usage of unary modalities (\Diamond , \Box^\downarrow) is an effective way to provide structural control in categorial inference. Categorial systems with

**Fig. 4.** Logic of Combining the PPTs**Fig. 5.** Logic of Combining the PPTs—Result

structural modalities (see [4], [8], [9] for details) can incorporate not only limited relaxation of the rigid structure to provide more generative capacity, but also impose additional constraints to block undesired derivations. In other words, different unary modalities can license structural relaxation and enforce a stronger structural discrimination.

These modalities play a somewhat different role when used in the context of this hybrid logic, as we now discuss.

4.1 Modality Eliminated

In a multi-modal system, the permutation modality can be used to allow the derivation of an object relative clause with an adverb, as in *... who Bob meets today*. This is because the same type assignments that allow the derivation of a simple object relative clause (5), will not allow the derivation with an adverb, as in (6).

To overcome this difficulty, the type assignment of *who* is modified, as in (7), which together with the use of the permutation modality allows the *np* argument to move to the needed location.

- $$\begin{array}{llll}
 (5) & \text{who} & \text{Bob meets} & \\
 & \vdash r/(s/np) \ np \ (np \backslash s)/np \Rightarrow r & & \\
 (6) & \text{who} & \text{Bob meets} & \text{today} \\
 & \not\vdash r/(s/np) \ np \ (np \backslash s)/np \ s \backslash s \Rightarrow r & & \\
 (7) & \text{who} & \text{Bob meets} & \text{today} \\
 & \vdash r/(s/np^\sharp) \ np \ (np \backslash s)/np \ s \backslash s \Rightarrow r & & \\
 & \vdash & np \ (np \backslash s)/np \ s \backslash s \Rightarrow s/np^\sharp &
 \end{array}$$

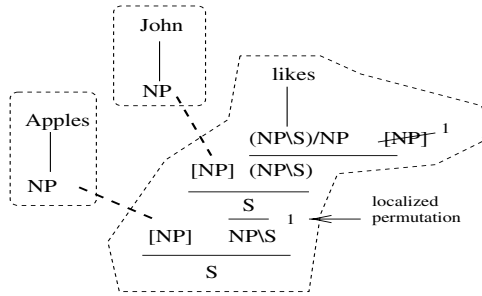
As we have already seen in the example in Figure 2, in the PPT approach, the modality is not needed at all. This is because *today* simply “stretches” into the PPT for *... who Bob meets*. Thus, the use of the permutation modality to allow non-peripheral extraction is handled inherently due to the use of two logics.

In Section 5 we discuss a more extensive case of the elimination of the need for modalities.

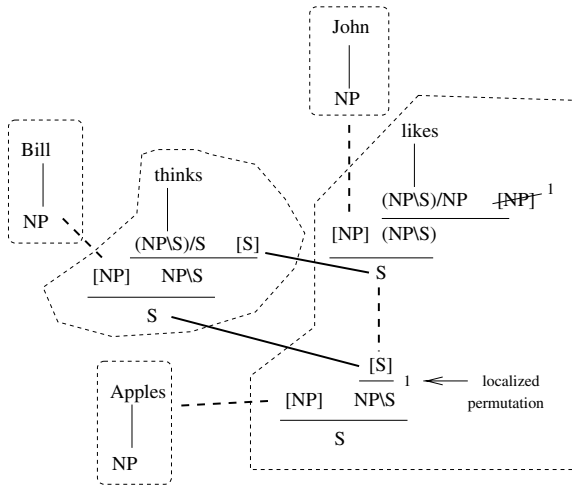
4.2 Modality Required, but Localized

However, in some cases we still retain the need for a modality but its use can be localized with desirable formal consequences. For example, topicalization is handled using the permutation modality. However, exactly because its use is *localized*, since every assumption must be discharged within the *same* elementary tree, it does not cause a collapse of the system. This use of local permutation is illustrated in Figure 6 for the sentence *Apples John likes*.

Unrestricted flexibility of permutation modality may lead to overgeneration. Since permutation is localized in this system, the problem does not arise. This localization is not a stipulation imposed on the system. That is, it is not the case that the logic needs to refer to where the parts of proofs come from. Since we

**Fig. 6.** Localized Permutation

use different logics, one for constructing the basic PPTs and one for combining them, the localization of the first logic to these building blocks (that is, the basic PPTs), is inherent in the system, and is not an arbitrary restriction.

**Fig. 7.** Long-Distance Permutation

Furthermore, no complications are raised by the case of long distance topicalization, as in *Apples Bill thinks John likes*, as illustrated in Figure 7. There is no change in the *likes* tree except that the *s* node is stretched. The localized use of permutation in the *likes* PPT is completely unaffected by this. The use of stretching allows *Bill thinks* to be inserted into the topicalized PPT for *Apples John likes*, resulting in *Apples Bill thinks John likes*.

4.3 Modality as Constraint on Stretching

In multi-modal categorial grammars, constraining modalities are needed to prevent various kinds of overgeneration. Their role can be illustrated by the representation of the relative clause with coordination in (8a), with the type assignments in (8b), and the validity of the latter proven by (8c).

- (8) a. that John wrote and Bob read
 b. $\vdash r/(s/np) (np (np \backslash s)/np (X \backslash \Box^\downarrow X)/X np (np \backslash s)/np)^\diamond \Rightarrow r$
 c. $\vdash np (np \backslash s)/np (X \backslash \Box^\downarrow X)/X np (np \backslash s)/np \Rightarrow$
 (with $X = s/np$) $\Box^\downarrow(s/np)$
- (9) *(the book) that John wrote Moby Dick and Bob read

The modal decorations on the types in (8b) are used to prevent the overgeneration of an island violation such as (9). With no such modality constraint, (9) could be derived with X instantiated to s , since the np assumption could be used to derive an s for *Bob read np*, which would coordinate with *John wrote Moby Dick*. However, with the modal decoration used for *and*, and the closing off of the coordinate structure with the dual structural modality $()^\diamond$, the undesired derivation fails because the hypothetical np finds itself in the scope of the modal operator.

While this is certainly a valid approach, the PPT system's hybrid logic allows for a somewhat different perspective on this problem. First structural bracketing such as $()^\diamond$ can be understood as a command to make a constituent. With PPTs, the constituents do not need to be stipulated—they arise naturally from the unfolding of the trees. Second, the effect of the \Box^\downarrow modality is handled by a restriction on the schema for coordination, where we feel it most naturally belongs. The crucial difference between the two approaches is that the PPTs system does not rely on modal type decorations, which can get successively more complex as the system becomes more sophisticated, although we do not discuss that in full detail here.

To illustrate, Figure 8A shows a partial proof tree for an object relative clause. It is used together with a coordination partial proof tree in Figure 8B to derive (8a). The tree in Figure 8B is inserted into that in Figure 8A by the use of stretching at the s/np node of the latter, deriving (8a).

The violation (9) is ruled out by the most simple of reasons. This sentence would require *and Bob read* to adjoin into a tree for *that John wrote Moby Dick*. The latter tree is simply not a well-formed object relative clause tree since there is no gap.

- (10) *...that John wrote and Bob read Moby Dick

A more interesting case is that of the corresponding violation (10), with the potential derivation shown in Figure 9. Since *that John wrote* is clearly an acceptable relative clause tree, what needs to be prevented is a tree for *and Bob read Moby Dick* stretching in at the s node, therefore coordinating at the s node. However, this is invalid because the s node for the *that John wrote* tree has an

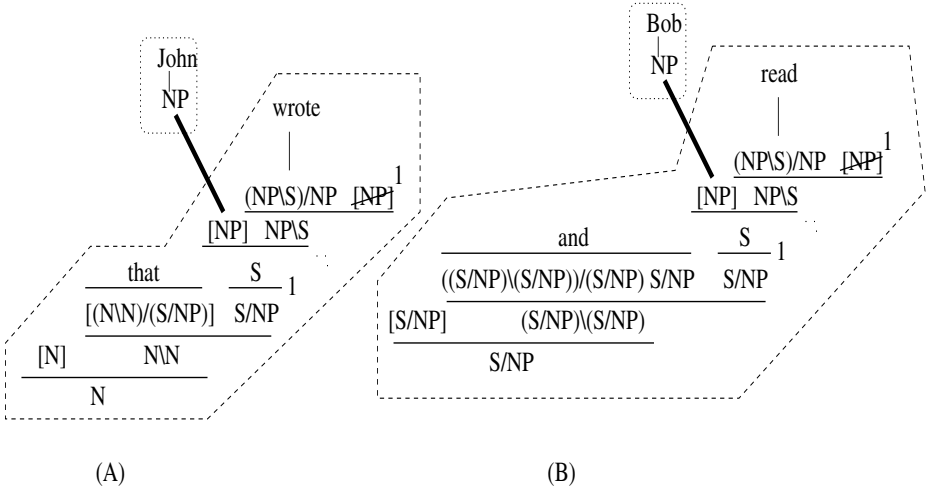


Fig. 8. Object Relative Clause and Coordination Trees

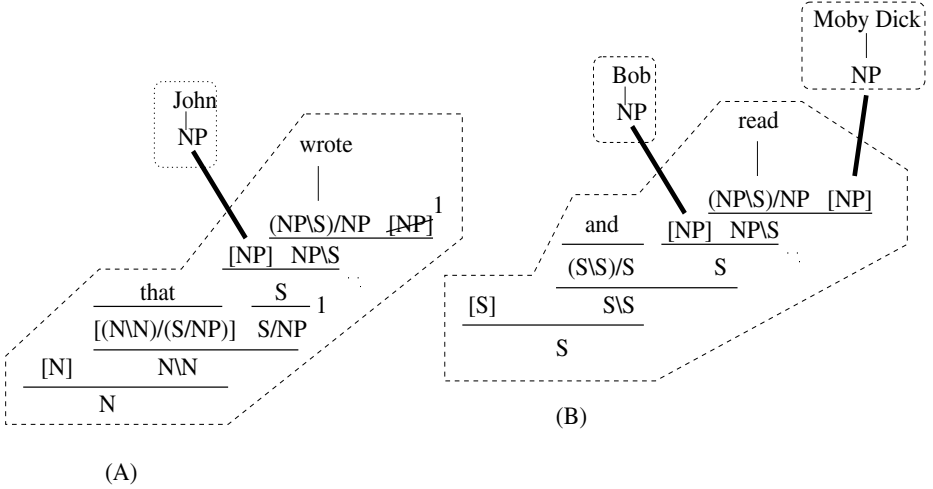


Fig. 9. A Disallowed Derivation

undischarged assumption while the s node for the *and Bob read Moby Dick* does not.

When a tree is inserted for stretching in the case of coordination, not only must the labels match for the stretching, but the labels must also coincide on any undischarged assumptions. So in the case of (10), the s node in the tree for *that John wrote* is more complex than just s , and also contains a list of undischarged assumptions, in this case the object np . In the tree for *and Bob read Moby Dick*, the s node has an empty list of undischarged assumptions. Since the two s nodes are not formally the same, the undesired coordination cannot take place.

More generally, while other approaches need to stipulate structural modalities and bracketing to capture the constituent structure needed for coordination, such structure is available for free in our approach. This is due precisely to the characterization of the PPTs as the unfolding of a lexical item. A good example of the advantages of this approach is given by the string *I think that Harry and Mary will lend you the money*, which is problematic for categorial grammar since both *I think that Harry* and *Mary* can have type $s/(np \backslash s)$, and therefore coordinate with an invalid reading. However, since for us coordination is based on more than just a *type*, but rather on the structural equivalence of the conjuncts, *I think that Harry* and *Mary* can easily be distinguished as unfit for coordination due to their different proof trees.

5 Cross Serial Dependencies in Dutch

We now address the classical problem of generating cross-serial dependencies in Dutch, as in sentence (11), again using the hybrid logic.³

- (11) dat Jan Piet Marie zag laten zwemmen
 that Jan Piet Marie saw make swim
 N1 N2 N3 V1 V2 V3
 ‘that Jan saw Piet make Marie swim’

Figure 10 shows the PPTs used for each clause in the Dutch example. We discuss below a logical view of these trees and how they are put together to derive the cross-serial dependencies. The crucial point to note about these trees is that a hypothetical assumption is used for the verb, which is then discharged later in the tree. This is accomplished by letting the anchor of the tree be a type-raised noun phrase, rather than the verb, and so the verb itself substitutes in.⁴ The main point to note for our approach is that we must allow the verbs to substitute without having their arguments unfolded. For example, if *laten* had its *np* and *s* arguments unfolded as usual, it would form a PPT and not be able to substitute into the $[s \backslash np \backslash s]$ argument in PPT (B) in Figure 10. Thus we conclude that while verbs may unfold, they are not obligated to.⁵

Logic models for these trees are presented in (12), (13), (14), which correspond to the trees (A),(B),(C) in Figure 10. For clarity of presentation we ignore

³ This analysis is based on the TAG analysis given in [3].

⁴ This accomplishes the same result as verb-raising in the TAG analysis of [3], creating the internal S nodes in the the trees as a locus for the required stretching. An alternative approach is to allow an empty verb to head the tree, which takes the real verb as an argument. There are some technical problems with this option, which we cannot discuss here. Note also that the simulation of verb raising in the tree for *Jan zag* is not necessary to derive the cross-serial dependencies, although we have kept it here.

⁵ This suggestion had already been made previously in [2]. An alternative is to allow the verb and the NP argument to be “co-anchors” of the tree. We leave this option aside for now.

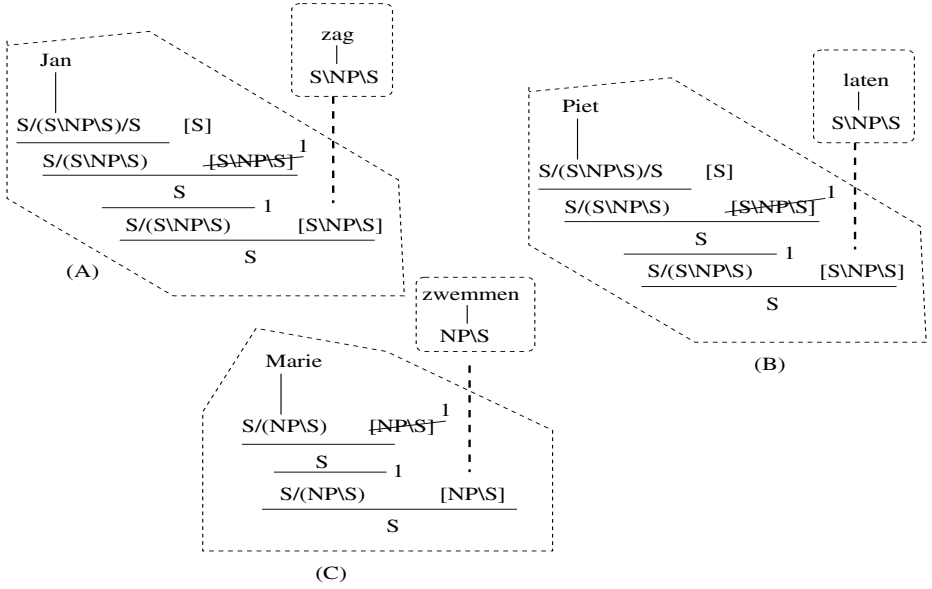


Fig. 10. Trees for Dutch cross-serial dependencies

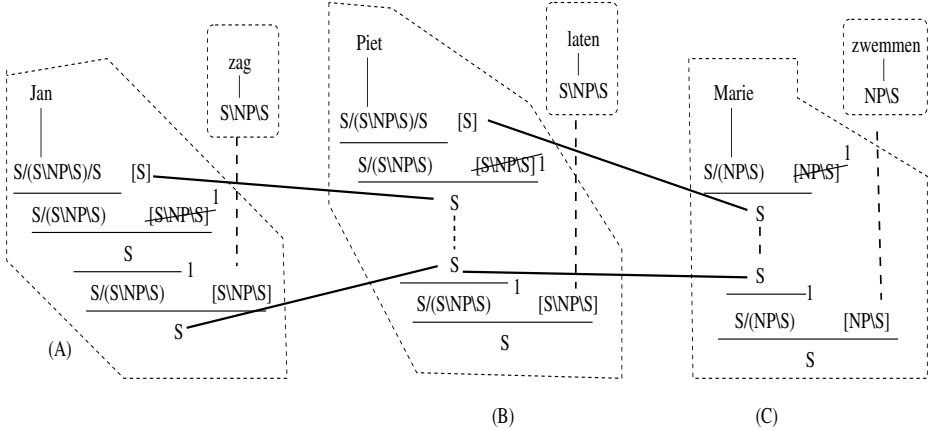


Fig. 11. Deriving the cross-serial dependencies

substitution and use the following abbreviation: α stands for $s\backslash np\backslash s$. Also, since the type of v_1 (*zag*) and v_2 (*laten*) is $s\backslash np\backslash s$, the following sequents are valid: $s/\alpha \ v_1 \Rightarrow s$ and $s/\alpha \ v_2 \Rightarrow s$. The sets of sequents (12), (13), (14) correspond in a simple way to the PPTs (A),(B),(C).

- (12) tree for Jan v_1
- $Jan \ s \Rightarrow s/\alpha$
 - $s/\alpha \ \alpha \Rightarrow s$

- c. $Jan\ s\ \alpha \Rightarrow s$ (cut of a,b)
 - d. $Jan\ s \Rightarrow s/\alpha$ (conditionalization)
 - e. $s/\alpha\ v_1 \Rightarrow s$
 - f. $Jan\ s\ v_1 \Rightarrow s$ (cut of d,e)
- (13) tree for Piet v_2
- a. $Piet\ s \Rightarrow s/\alpha$
 - b. $s/\alpha\ \alpha \Rightarrow s$
 - c. $Piet\ s\ \alpha \Rightarrow s$ (cut of a,b)
 - d. $Piet\ s \Rightarrow s/\alpha$ (conditionalization)
 - e. $s/\alpha\ v_2 \Rightarrow s$
 - f. $Piet\ s\ v_2 \Rightarrow s$ (cut of d,e)
- (14) tree for Marie v_3
- a. $Marie\ np\backslash s \Rightarrow s$
 - b. $Marie \Rightarrow s/(np\backslash s)$ (conditionalization)
 - c. $s/(np\backslash s)\ v_3 \Rightarrow s$
 - d. $Marie\ v_3 \Rightarrow s$ (cut of b,c)

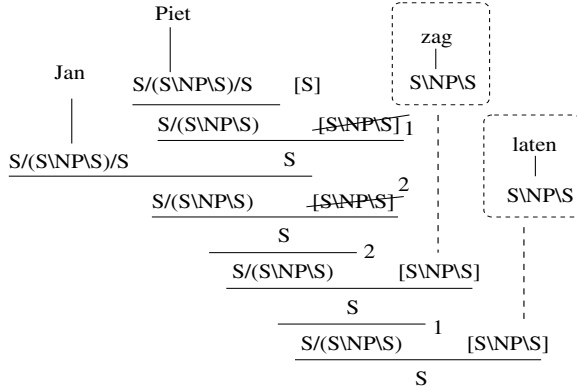


Fig. 12. The result of stretching (A) into (B)

Before detailing the use of the second aspect of the hybrid logic (the use of cut to compose the PPTs together), we illustrate using the figures of the trees how sentence (11) is derived using the PPTs in Figure 10). The basic idea is shown in Figure 11. The internal S nodes of the (B) and (C) (v_2 and v_3) PPTs are stretched and the the (A) and (B) (v_1 and v_2) PPTs are, respectively, inserted. Figure 11 shows the two stretchings (of (A) into (B), and of (B) into (C)) occurring simultaneously. Technically, however, they are two independent compositions and must occur in sequence, although it does not matter which order they occur in. Here we use the order of (A) stretching into (B), and (B) stretching into (C).

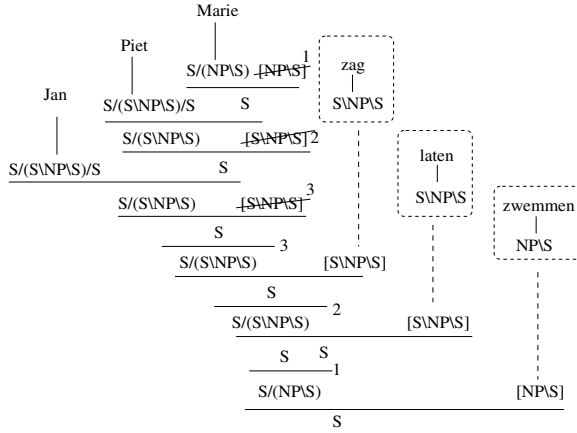


Fig. 13. The result of stretching Figure 12 into (C)

Figure 12 shows the result of (A) stretching into (B). The yield of the result is *Jan Piet zag laten* (N1 N2 V1 V2). Note that we have renumbered the assumption that is assumed and discharged in (A) from 1 to 2 in Figure 12. This is only for expository purposes, to avoid confusion with the assumption discharged in the tree for *Piet laten*, which remains as 1 in the figure. The numbers themselves of course have no relevance whatsoever—the important point is that there are no assumptions or withdrawals taking place as a result of the stretching. The dependencies that are present in (A) and (B) remain in Figure 12, with the dependency between the assumed and discharged $[s\backslash np\backslash s]$ in the *Piet laten* PPT getting “stretched apart” by the insertion of the *Jan zag* tree. Crucially, the yield of the tree in Figure 12 is the desired crossed dependencies for the two clauses.

The final step is the stretching of Figure 12 into the (C) in figure 10. The result is shown in Figure 13. Once again the hypotheticals have been renumbered, with no significance other than appearance. The yield of the tree is sentence (11).

We now describe the precise logical steps involved in the derivation, by applying our technique of combining PPTs. First, the cut rule is applied to (12a) and (13b), which results in $Jan\ s/\alpha\ \alpha \Rightarrow s/\alpha$. Since, according to our strategy, we replace s by $s/\alpha\ \alpha$ in (12), the last step of the tree, corresponding to (12f), is $Jan\ s/\alpha\ \alpha\ v_1 \Rightarrow s$. This sequent replaces (13b), which means that in (13) the context $s/\alpha\ \alpha$ is replaced by the new context $Jan\ s/\alpha\ \alpha\ v_1$, with no structural change. The result is shown in (15).

- (15)
- a. $Piet\ s \Rightarrow s/\alpha$
 - b. $Jan\ s/\alpha\ \alpha\ v_1 \Rightarrow s$
 - c. $Jan\ Piet\ s\ \alpha\ v_1 \Rightarrow s$ (from cut of a,b)
 - d. $Jan\ Piet\ s\ v_1 \Rightarrow s/\alpha$ (conditionalization)
 - e. $s/\alpha\ v_2 \Rightarrow S$
 - f. $Jan\ Piet\ s\ v_1\ v_2 \Rightarrow S$

Now we make a cut between (15f) and (14a). We show in (16) the resulting PPT, again keeping the structure, in this case changing the context from *Mary np\s* to *Jan Piet Mary np\s v₁ v₂*.

- (16)
- a. *Jan Piet Mary np\s v₁ v₂ ⇒ s*
 - b. *Jan Piet Mary v₁ v₂ ⇒ s/(np\s)*
(conditionalization)
 - c. *s/(np\s) v₃ ⇒ s*
 - d. *Jan Piet Mary v₁ v₂ v₃ ⇒ s*
(cut of b,c)

Unlike the multi-modal approach ([6], [7]) to derive cross-serial dependencies in Dutch, the system of categorial inference described here implicitly handles permutation without introducing structural modalities, thanks to the properties of the hybrid logic.

We conclude this section with a note on the generative power of this system. The cross-serial derivation shows that the PPTs system can derive languages that are strongly more powerful than context-free grammar. This construction can be easily extended for a strictly context-sensitive language, such as $a^n b^n c^n$.

6 Conclusion

We have discussed here some key concerns of categorial grammar from the perspective of a hybrid logic system motivated by the insights of Lexicalized Tree Adjoining Grammar. The hybridization of logical inference leads to a localization of resource management that simplifies the use of structural modalities to control such management. The resulting system can derive languages that are beyond context-free. In addition, desirable computational properties are inherited from LTAG, in particular polynomial parsing.

We conclude by mentioning some issues for further work. First, we would like to explore the handling of right node raising in this framework. There are well-known differences between the leftward extraction discussed here and the rightward extraction of the RNR cases. While we cannot discuss it here, the prospects are promising because we can distinguish between the two cases, due to the use of proof trees rather than just types, and because our coordination schema depends on the basic partial proof trees associated with each conjunct. The leftward and rightward cases will have different proof trees and the different properties can therefore be modelled appropriately.

We also plan to refine the coordination schema to handle the coordination of simple NPs and generalized quantifiers, such as *Bob and a boy walked*, by incorporating deductively-connected types in the coordination schema.

References

1. Abrusci, M., Ch. Fouqueré, and J. Vauzeilles. Tree adjoining grammar and non-commutative linear logic. In *Proceedings of Logical Aspects of Computational Linguistics*, 1996.

2. Joshi, A. K., and S. Kulick. Partial proof trees as building blocks for a categorical grammar. *Linguistics and Philosophy*, 20:637–667, 1997.
3. Kroch, A. S., and B. Santorini. The derived constituent structure of the West Germanic verb-raising construction. In Freidin, R., editor, *Principles and Parameters in Comparative Grammar*. MIT Press, Cambridge MA, 1991.
4. Kurtonina, N., and M. Moortgat. Structural control. In Blackburn, P., and M. de Rijke, editors, *Specifying Syntactic Structures*. CSLI, 1997.
5. Lecomte, A., and C. Retoré. Words as modules and modules as partial proof-nets. In Benjamins, J., editor, *Proceedings of ICML 96*, 1996.
6. Moortgat, M., and R. Oehrle. Logical parameters and linguistic variation. Lecture notes on categorical grammar, 1993. Fifth European Summer School in Logic, Language and Information, Lisbon.
7. Moortgat, M., and R. Oehrle. Adjacency, dependency and order. In Dekker, P., and M. Stokhof, editors, *Proceedings Ninth Amsterdam Colloquium*, pages 447–466, ILLC, Amsterdam, 1994.
8. Moortgat, M. Categorical type logics. In Benthem, J. van, and A. ter Meulen, editors, *Handbook of Logic and Language*. North Holland, 1997.
9. Morrill, G. *Type Logical Grammar—Categorical Logic of Signs*. Kluwer, Dordrecht, 1994.

Dominance Constraints: Algorithms and Complexity

Alexander Koller¹, Joachim Niehren², and Ralf Treinen³

¹ Department of Computational Linguistics, Universität des Saarlandes, Saarbrücken, Germany, koller@coli.uni-sb.de

² Programming Systems Lab, Universität des Saarlandes, Saarbrücken, Germany, niehren@ps.uni-sb.de

³ Laboratoire de Recherche en Informatique, Université Paris-Sud, Orsay, France, treinen@lri.fr

Abstract. Dominance constraints for finite tree structures are widely used in several areas of computational linguistics including syntax, semantics, and discourse. In this paper, we investigate algorithmic and complexity questions for dominance constraints and their first-order theory. The main result of this paper is that the satisfiability problem of dominance constraints is NP-complete. We present two NP algorithms for solving dominance constraints, which have been implemented in the concurrent constraint programming language Oz. Despite the intractability result, the more sophisticated of our algorithms performs well in an application to scope underspecification. We also show that the positive existential fragment of the first-order theory of dominance constraints is NP-complete and that the full first-order theory has non-elementary complexity.

Keywords. Dominance constraints, complexity, computational linguistics, underspecification, constraint programming.

1 Introduction

Dominance constraints are a popular tool for describing trees throughout computational linguistics. They allow to express both *immediate dominance* (and labeling) relations and general (reflexive, transitive) *dominance* relations between the nodes of a tree. In syntax, they provide for underspecified tree descriptions employed in deterministic parsing [11] and to combine TAG with unification grammars [20]. In underspecification of the semantics of scope ambiguities, dominance constraints are omnipresent. While they are somewhat implicit in earlier approaches [15,2], they are used explicitly in two recent formalisms [5,13]. An application of dominance constraints in discourse semantics has recently been proposed in [6], and they have been used to model information growth and partiality [12].

Despite their popularity, there have been no results about the computational complexity of solving these constraints, i.e. of finding a tree that satisfies all

given constraints. In this paper, we remedy this situation by proving that the satisfiability problem of dominance constraints is NP-complete. This result holds for all logical fragments between the purely conjunctive fragment and the positive existential fragment. We present two algorithms for solving them; one involves a nondeterministic guessing step, which makes it convenient for a completeness proof, but not for implementation, whereas the other gives priority to deterministic computation steps and enumerates cases only by need. Finally, we show that the first-order theory over dominance constraints with a signature of bounded arity is decidable and has non-elementary complexity. The decidability result is not new – e.g. [16] sketches a proof for a different variant of dominance constraints –, but we work out the details of a transparent proof by encoding into second-order monadic logic for the first time.

Related Work. In [17], it was shown how to solve formulae from the propositional language over (a different variant of) dominance constraints (over a different type of trees). There, tableau-style saturation rules for enumerating models are presented which are quite similar to the ones we use here. This solution procedure terminates, but there are no complexity results. Continuing this line of work, [1] present a sets of first-order axioms over dominance constraints which capture certain classes of trees.

From an implementation perspective, dominance constraints were approached first in [3], which presents an implementation based on *finite set constraints*. A more advanced version of the algorithm presented here and an implementation thereof are given [4]. This implementation is also based on finite set constraint programming but improves that of [3].

Plan of the paper. In Section 2, we start out by defining the syntax and semantics of dominance constraints. In Section 3, we present the solution algorithms for dominance constraints and prove their soundness, completeness, and NP run-times. The algorithms are first defined for the (purely conjunctive) language of dominance constraints and extended to the other propositional connectives later. In Section 4, we complement this result by proving NP-hardness of the problem. In fact, we will not really provide the details of the proof, but give a thorough explanation of the proof idea. In Section 5, we turn to the decidability and complexity of the first-order theory over dominance constraints. Section 6 summarizes and concludes the paper. Some of the proofs are only sketched; for more details, we refer the reader to [8].

2 Syntax and Semantics of Dominance Constraints

In this section, we define the syntax and semantics of dominance constraints. To this end, we first introduce the notion of a *tree structure*, the kind of first-order structure we will interpret dominance constraints over. After that, it will be straightforward to define the actual syntax and semantics. Finally, we look briefly at *constraint graphs*, a graphical syntactic alternative.

2.1 Tree Structures

Throughout, we take \mathbb{N} to be the set of positive integers and \mathbb{N}_0 to be the set of nonnegative integers and assume that Σ is a ranked signature that contains *function symbols* or *tree constructors* f, g, a, b, \dots , which are assigned arities by an arity function $\text{ar} : \Sigma \rightarrow \mathbb{N}_0$. We further assume that Σ contains at least two constructors, one of which is nullary, and the other one of arity at least 2.

Intuitively, we want trees to be essentially the same as ground terms over Σ . Formally, we first define a *tree domain* D to be a nonempty prefix-closed subset of \mathbb{N}^* ; i.e., the elements of D are words of positive integers. These words can be thought of as the paths from the root of a tree to its nodes. We write the concatenation of two words π and π' as juxtaposition $\pi\pi'$.

We define a *constructor tree* τ to be a pair (D_τ, L_τ) of a tree domain D_τ and a *labeling function*

$$L_\tau : D_\tau \rightarrow \Sigma,$$

with the additional property that for every $\pi \in D_\tau$, $\pi k \in D_\tau$ iff $1 \leq k \leq \text{ar}(L_\tau(\pi))$. A *finite constructor tree* is a constructor tree whose domain is finite. Throughout, we will simply say “tree” to mean “finite constructor tree”.

The *tree structure* \mathcal{M}^τ over the tree τ is a first-order model structure with the universe D_τ and whose interpretation function assigns relations over D_τ to a set of fixed predicate symbols. We will use the same symbols for the predicate symbols and their interpretations; as the latter are applied to paths and the former are applied to variables, there is no danger of confusion. The interpretation function is fully determined by τ ; so to specify a tree structure, it is sufficient to specify the underlying tree.

In detail, the interpretation is as follows. If $f \in \Sigma$ has arity n , the *labeling* relation $\pi : f(\pi_1, \dots, \pi_n)$ is true in \mathcal{M}^τ iff $L_\tau(\pi) = f$ and for all $1 \leq i \leq n$, $\pi_i = \pi i$. The *dominance* relation $\pi \triangleleft^* \pi'$ is true iff π is a prefix of π' .

2.2 Syntax and Semantics of Dominance Constraints

With these definitions, it is straightforward to define the syntax and semantics of dominance constraints. Assuming a set of (node) variables X, Y, \dots , a *dominance constraint* φ has the following abstract syntax:

$$\begin{array}{ll} \varphi ::= X : f(X_1, \dots, X_n) & f \in \Sigma, n = \text{ar}(f) \\ \quad \mid X \triangleleft^* Y & \\ \quad \mid \varphi \wedge \varphi'. & \end{array}$$

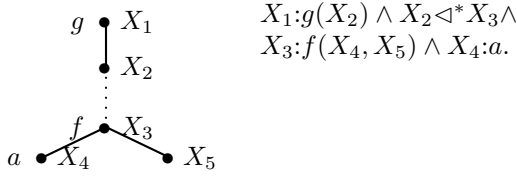
We use the formula $X = Y$ as an abbreviation for $X \triangleleft^* Y \wedge Y \triangleleft^* X$.

We will start by considering only this (purely conjunctive) *constraint language* and successively allow more logical connectives, until we have arrived at the full first-order language in Section 5.

Satisfaction of an atomic constraint is defined with respect to a pair $(\mathcal{M}^\tau, \alpha)$ of a tree structure \mathcal{M}^τ and a variable assignment $\alpha : \text{Var} \rightarrow D_\tau$ that assigns

nodes to the variables. This is extended to satisfaction of arbitrary formulae in the usual Tarskian way.

Because dominance constraints can easily become unreadable, we will use *constraint graphs* as a graphical device to represent them. They are essentially an alternative to the original syntax of the language. Constraint graphs are directed graphs with two kinds of edges: solid and dotted. The nodes of a constraint graph represent variables in a constraint. Labeling constraints are expressed by attaching the constructor to the node and drawing solid edges to the children; dominance constraints are expressed by drawing a dotted edge between the respective nodes. As an example, the graph below is the constraint graph for the constraint to its right.



3 Solving Dominance Constraints

Now we show that the satisfiability problems of all languages over dominance constraints between the (purely conjunctive) constraint language itself and the positive existential fragment are in NP. We first define an algorithm that decides satisfiability for the constraint language and prove the running time, soundness, and completeness. Then we present an algorithm that does the same thing, but lends itself more easily to implementation. Finally, we extend the results to the other propositional connectives.

It turns out that it's actually easier to define satisfiability algorithms for dominance constraints if we additionally allow atomic constraints of the form $\neg X \triangleleft^* Y$. Hence, we are going to work with this extended language of dominance constraints in Sections 3.1 to 3.3.

3.1 The Algorithm

The first algorithm proceeds in three steps. First, we guess nondeterministically for each pair X, Y of variables in φ if X dominates Y or not, and add the corresponding atomic constraint to φ . This is done by the (Choice) rule, where **or** stands for nondeterministic choice.

$$(\text{Choice}) \quad \mathbf{true} \rightarrow X \triangleleft^* Y \mathbf{or} \neg X \triangleleft^* Y$$

In the second step, we saturate φ according to the following deterministic *propagation* rules. We recall that $X = Y$ is an abbreviation for $X \triangleleft^* Y \wedge Y \triangleleft^* X$.

(Refl)	$\mathbf{true} \rightarrow X \triangleleft^* X$	(X occurs in φ)
(Trans)	$X \triangleleft^* Y \wedge Y \triangleleft^* Z \rightarrow X \triangleleft^* Z$	
(Decomp)	$X=Y \wedge X:f(X_1, \dots, X_n) \wedge Y:g(Y_1, \dots, Y_n) \rightarrow \bigwedge_{i=1}^n X_i=Y_i$	
(Disj)	$X:f(\dots, X_i, \dots, X_k, \dots) \rightarrow \neg X_i \triangleleft^* X_k \quad (1 \leq i \neq k \leq n)$	
(Dom)	$X:f(\dots, Y, \dots) \rightarrow X \triangleleft^* Y$	
(Parent)	$X=Y \wedge X':f(\dots, X, \dots) \wedge Y':g(\dots, Y, \dots) \rightarrow X'=Y'$	
(Child)	$X \triangleleft^* Y \wedge X:f(X_1, \dots, X_n) \wedge \bigwedge_{i=1}^n (\neg X_i \triangleleft^* Y) \rightarrow Y \triangleleft^* X$	

In the Parent rule, f and g need not be different.

In the third step, we detect unsatisfiable constraints by applying the following *clash* rules.

(Clash1)	$X:f(\dots) \wedge Y:g(\dots) \wedge X=Y \rightarrow \mathbf{false},$	if $f \neq g$
(Clash2)	$X \triangleleft^* Z \wedge Y \triangleleft^* Z \wedge \neg X \triangleleft^* Y \wedge \neg Y \triangleleft^* X \rightarrow \mathbf{false}$	
(Clash3)	$X \triangleleft^* Y \wedge \neg X \triangleleft^* Y \rightarrow \mathbf{false}$	
(Clash4)	$X:f(X_1, \dots, X_i, \dots, X_n) \wedge X_i \triangleleft^* X \rightarrow \mathbf{false}$	

After the initial guessing step, the algorithm applies all instances of all propagation and clash rules. We call a constraint to which no clash rule can be applied *clash-free*, the result of applying all possible rules to a constraint for as long as the constraint is clash-free its *saturation*, and a constraint which is its own saturation *saturated*. The algorithm outputs that its input is satisfiable if it can find a clash-free saturation (that is, can apply the guessing step in such a way that subsequent propagation and clash rules won't produce **false**); otherwise, it outputs that the input is unsatisfiable.

An example for application of these rules is to prove the unsatisfiability of

$$X:a \wedge X \triangleleft^* Y \wedge \neg Y \triangleleft^* X,$$

where a is a nullary symbol. Application of the (Child) rule adds the new constraint $Y \triangleleft^* X$. But this makes the (Clash3) rule applicable, so the algorithm finds a clash. A really tricky example is to prove the unsatisfiability of

$$Y:f(Z) \wedge X:g(U) \wedge U \triangleleft^* Z \wedge \neg X \triangleleft^* Y.$$

The (Dom) and (Trans) rules will give us $Y \triangleleft^* Z$, $X \triangleleft^* U$, and $X \triangleleft^* Z$. Now for the saturation to be clash-free, (Choice) must have guessed $Y \triangleleft^* X$; for if it had chosen $\neg Y \triangleleft^* X$, we would get a clash with (Clash2). Similarly, (Choice) must have guessed $\neg Z \triangleleft^* X$, for if it had chosen $Z \triangleleft^* X$, we could derive $U \triangleleft^* X$, which produces a clash with (Clash4). But in this case, we can derive $X \triangleleft^* Y$ with the (Child) rule, which causes a clash with (Clash3).

It's easy to see that the algorithm terminates in NP time. As we have guessed the dominance relations between all variables in the first step, the second step can never consistently add a new constraint; either the constraint is already known, or it clashes, by the Clash3 rule. So we will only spend deterministic polynomial time with the application of propagation and clash rules. Note that one major change of the second algorithm below will be to allow the propagation rules to be more productive.

Proposition 1 (Soundness). *A satisfiable dominance constraint has a clash-free saturation.*

Proof. Assume that the constraint φ is satisfiable. Clearly, the guessing step of the algorithm can add a choice of (possibly negated) dominance constraints such that their conjunction φ' with φ is satisfiable as well; we only have to read off whether the denotations of two variables dominate each other in a fixed solution of φ . Now all propagation rules maintain satisfiability, and the preconditions of all clash rules are unsatisfiable. Hence, φ' is saturated and clash-free. \square

3.2 Completeness

As usual, proving completeness is slightly more involved than proving soundness. Here, we proceed in two steps: First we show that a special class of saturated, clash-free constraints is satisfiable; then we show that every saturated, clash-free constraint can be extended by some additional conjuncts to a saturated, clash-free constraint of the restricted class. Together, this shows completeness:

Proposition 2 (Completeness). *A saturated and clash-free constraint is satisfiable.*

Incidentally, the proof also shows how to obtain a model for a clash-free constraint. But first, some terminology. We call $V \subseteq V(\varphi)$ an *equality set* for φ if $Y_1 \triangleleft^* Y_2$ in φ for all $Y_1, Y_2 \in V$. All variables in an equality set must be mapped to the same node in a solution of φ . A variable X is *labeled* in φ if there is an X' such that $\{X, X'\}$ is an equality set for φ and $X': f(X'_1, \dots, X'_n)$ in φ for some term $f(X'_1, \dots, X'_n)$. We call a constraint φ *simple* if all its variables are labeled, and if there is a so-called *root variable* Y for φ such that $Y \triangleleft^* Z$ in φ for all $Z \in V(\varphi)$.

Lemma 1 (Satisfiability of Simple Constraints). *A simple, saturated, and clash-free constraint is satisfiable.*

Proof. It is not difficult to show that for any $Z \in V(\varphi)$, there is a unique sequence of maximal equality sets E_1, \dots, E_n that connect the root of φ to Z via labeling constraints. From this, we can read off the satisfying tree structure and variable assignment in a straightforward way. \square

It remains to show that we can restrict our attention to simple constraints. An *extension* of a constraint φ is a constraint of the form $\varphi \wedge \varphi'$ for some φ' . We will show how to extend a saturated, clash-free constraint to a simple, saturated, and clash free constraint.

We define the set $\text{con}_\varphi(X)$ of variables *connected* to X in φ as follows:

$$\text{con}_\varphi(X) = \left\{ Y \mid \begin{array}{l} X \triangleleft^* Y \text{ in } \varphi, Y \triangleleft^* X \text{ not in } \varphi, \text{ not exists } Z \text{ s.t.} \\ X \triangleleft^* Z, Z \triangleleft^* Y \text{ in } \varphi, Z \triangleleft^* X, Y \triangleleft^* Z \text{ not in } \varphi \end{array} \right\}$$

Note that for this algorithm, $X \triangleleft^* Y$ *not in* φ is the same as $\neg X \triangleleft^* Y$ *in* φ ; it does, however, make a difference for the second algorithm below. Intuitively, a variable is connected to X if it is a “minimal dominance child” of X . So for example, in

$$\varphi_1 := X \triangleleft^* X \wedge X \triangleleft^* Y \wedge \neg Y \triangleleft^* X \wedge X \triangleleft^* Z \wedge \neg Z \triangleleft^* X \wedge Y \triangleleft^* Z \wedge \neg Z \triangleleft^* Y,$$

$\text{con}_{\varphi_1}(X) = \{Y\}$ and $\text{con}_{\varphi_1}(Y) = \{Z\}$.

We call $V \subseteq V(\varphi)$ a *disjointness set* for φ if for any two distinct variables $Y_1, Y_2 \in V$, $Y_1 \triangleleft^* Y_2$ *not in* φ . The idea is that all variables in a disjointness set can safely be placed at disjoint positions of a tree.

Lemma 2. *If φ is saturated and $X \in V(\varphi)$ then for all $Y_1, Y_2 \in \text{con}_{\varphi}(X)$, the set $\{Y_1, Y_2\}$ is either an equality or disjointness set for φ .*

For a constraint φ and a variable X of φ , Lemma 2 implies the existence of maximal disjointness sets $V \subseteq \text{con}_{\varphi}(X)$ for φ . Such a set is constructed by choosing one representative from every maximal equality set contained in $\text{con}_{\varphi}(X)$.

Now we can state and prove the key lemma of the completeness proof.

Lemma 3 (Extension by Labeling). *Let φ be a constraint and X a variable that is unlabeled in φ . Let $\{X_1, \dots, X_n\} \subseteq \text{con}_{\varphi}(X)$ be a disjointness set for φ that is maximal among all disjointness sets that are subsets of $\text{con}_{\varphi}(X)$, and let f be a function symbol of arity n . If φ is saturated and clash-free, then $\varphi \wedge X:f(X_1, \dots, X_n)$ is also saturated and clash-free.*

Proof. Let $\varphi' = \varphi \wedge X:f(X_1, \dots, X_n)$. Since we have not introduced new variables or dominance relations, φ' inherits saturation with respect to the nondeterministic guessing rule, (Refl), and (Trans) and clash-freeness with respect to (Clash2) and (Clash3) from φ' . The rule (Decomp) is not applicable to φ' ; otherwise, X would have been labeled in φ . By the same argument, the (Clash1) rule is not applicable to φ' . The only new way to apply the (Dom) rule is to the new labeling constraint; but the dominances (Dom) can derive are already in φ . The (Clash4) rule is not applicable, either: No $X_i \triangleleft^* X$ can be in φ' because $X_i \in \text{con}_{\varphi}(X)$. The arguments for the remaining rules are more interesting:

(Disj) The only new way in which the (Disj) might apply is as follows:

$$X:f(X_1, \dots, X_n) \rightarrow \neg X_i \triangleleft^* X_j \quad (i \neq j)$$

By assumption, $\{X_1, \dots, X_n\}$ is a disjointness set for φ . Hence $\neg X_i \triangleleft^* X_j$ *in* φ , i.e. φ' is saturated under (Disj).

(Child) The only possible case in which the (Child) rule might newly apply looks as follows:

$$X \triangleleft^* Y \wedge X:f(X_1, \dots, X_n) \wedge \bigwedge_{i=1}^n (\neg X_i \triangleleft^* Y) \rightarrow Y \triangleleft^* X$$

But since we have chosen $\{X_1, \dots, X_n\}$ to be a maximal disjointness set of $\text{con}_\varphi(X)$, it follows from $X \triangleleft^* Y$ in φ that either $Y \triangleleft^* X$ is already in φ , or there is a $1 \leq i \leq n$ such that $X_i \triangleleft^* Y$ in φ (compare Lemma 2). Hence, the (Child) rule is not applicable in any new way either.

(Parent) Finally, the only non-trivial new way in which to apply the (Parent) rule looks as follows.

$$X_i = Z \wedge X : f(\dots, X_i, \dots) \wedge Y : g(\dots, Z, \dots) \rightarrow X = Y$$

We show that this is not possible either: $X_i = Z \wedge Y : g(\dots, Z, \dots)$ may not belong to φ . We distinguish four cases depending on the positive and negative dominance relations between X and Y .

1. $X \triangleleft^* Y$ in φ :
 - a) $Y \triangleleft^* X$ in φ : This implies that X would have been labeled in φ which contradicts the assumption of the lemma.
 - b) $\neg Y \triangleleft^* X$ in φ : Because of (Dom) and (Trans), $Y \triangleleft^* X_i$ in φ . Saturation under (Trans) and (Clash4) implies $\neg X_i \triangleleft^* Y$ in φ . But this contradicts $X_i \in \text{con}_\varphi(X)$, as Y could take the role of Z in the second line of the definition of $\text{con}_\varphi(X)$.
2. $\neg X \triangleleft^* Y$ in φ :
 - a) $Y \triangleleft^* X$ in φ : We show that all immediate children of Y (i.e. Z and all of its siblings in the labeling constraint) do not dominate X ; then we can apply the (Child) rule to conclude $X \triangleleft^* Y$ in φ , which contradicts $\neg X \triangleleft^* Y$ in φ , i.e. the clash-freeness of φ with respect to rule (Clash3).
Clearly, $Z \triangleleft^* X$ not in φ ; for otherwise, (Trans) would imply that $X_i \triangleleft^* X$ in φ , which contradicts the $\text{con}_X(\varphi)$ condition for X_i . So let Z' be a sibling of Z in the labeling constraint above. If $Z' \triangleleft^* X$ in φ , then $Z' \triangleleft^* Z$ in φ (by the Trans rule); but by the (Disj) rule, $\neg Z' \triangleleft^* Z$ in φ , in contradiction to (Clash3).
 - b) $\neg Y \triangleleft^* X$ in φ : Saturation under (Trans) implies $X \triangleleft^* Z$ in φ , and saturation under (Dom) yields $Y \triangleleft^* Z$ in φ . This means that (Clash2) is applicable on φ , in contradiction to the clash-freeness of φ . \square

Corollary 1 (Reduction to Simple Constraints). *Every saturated, clash-free constraint has a simple, saturated, clash-free extension.*

Proof. Let φ be saturated and clash-free. Without loss of generality, φ has a root variable (otherwise, we choose a fresh variable X and consider $\varphi \wedge \bigwedge \{X \triangleleft^* Y \mid Y \in V(\varphi)\}$ instead of φ).

By Lemma 3, we can successively label all variables in φ . The only problem is that the signature might not contain a function symbol for an arity we need; but we can get around this (artificial) problem by encoding these symbols with a nullary symbol and one symbol of arity ≥ 2 , whose existence we have assumed. \square

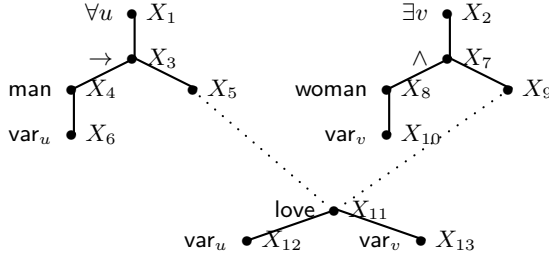


Fig. 1. An underspecified representation of *Every man loves a woman*.

3.3 A More Practical Solution Algorithm

The algorithm we have just presented is convenient from a theoretical perspective, but from a practical perspective, it's totally useless. Let's consider an example from scope underspecification for illustration. The constraint graph in Fig. 1 represents a dominance constraint describing the readings of the ambiguous sentence (1).

(1) *Every man loves a woman.*

This constraint has 14 variables; so the algorithm will consider 2^{196} or about 10^{59} alternatives in the guessing step, which of course is way too much to search through deterministically.

A more practically feasible satisfiability algorithm is inspired by *constraint programming* [9]. We replace the nondeterministic guessing rule (Choice) by two *distribution* rules. The application strategy is to apply propagation and clash rules for as long as possible; only when no such rule is applicable, a single application of a distribution rule takes place.

$$\left. \begin{array}{llll} \text{(Ref)} & \text{(Trans)} & \text{(Decomp)} & \text{(Child)} \\ \text{(Disj)} & \text{(Dom)} & \text{(Parent)} & \end{array} \right\} \text{ Propagation rules from Sect. 3.1}$$

$$\left. \begin{array}{ll} \text{(Clash1)} & \text{(Clash2)} \\ \text{(Clash3)} & \text{(Clash4)} \end{array} \right\} \text{ Clash rules from Sect. 3.1}$$

$$\begin{array}{ll} \text{(Distr1)} & X \triangleleft^* Z \wedge Y \triangleleft^* Z \rightarrow X \triangleleft^* Y \text{ or } Y \triangleleft^* X \\ \text{(Distr2)} & X \triangleleft^* Y \wedge X : f(X_1, \dots, X_n) \rightarrow Y \triangleleft^* X \text{ or } \bigvee_{i=1}^n X_i \triangleleft^* Y \end{array}$$

Proving soundness of the modified algorithm is a trivial extension of the original soundness proof. The completeness proof has exactly the same structure as the one above, but the details of the proofs of Lemmas 1 and 3 have to be changed.

On the example (1), the new algorithm will alternate between propagation steps and single applications of the (Distr1) rule. The first application of this rule will be to X_5 and X_9 . The algorithm terminates after a total of three applications of (Distr1); this means that the search space has a size of at most eight.

As it stands, the second algorithm can be implemented with reasonable efficiency, but it's still not perfect for the application to scope underspecification, mainly because it is based on the wrong set of atomic constraints (\prec^* and $\neg\prec^*$); using \prec^* , inequality \neq , and disjointness \perp (two nodes are disjoint if they don't dominate each other) would be better. A more advanced version of the second algorithm which takes this into account and runs even more efficiently is defined in [4].

Alternatively, an implementation of a dominance constraint solver can be based on *finite set constraints*. This was first suggested in [3] and is worked out e.g. in [10]. [4] compares these two fundamental alternatives to implementing dominance constraint solvers in terms of runtimes and search spaces. An implementation of the set constraint based solver can be found on the WWW at <http://www.coli.uni-sb.de/cl/projects/chorus/demo.html>.

3.4 Larger Logical Languages

The algorithms we have defined so far solve dominance constraints that are pure conjunctions of labeling, dominance, and negated dominance constraints. We now extend them to allow in addition disjunctions and, later, negations in formulae over these atomic constraints. Positive occurrences of existential quantifiers can always be added, as they make no difference for satisfiability. The proofs for this section are simple and will be omitted.

In a nondeterministic setting, it is easy to deal with *disjunctions*; all we have to do is to go through the formula recursively and guess for each disjunction which disjunct can be satisfied. In this way, we produce a conjunction that we can feed into the original algorithm. It is easily shown that a formula φ of disjunctions and conjunctions over dominance constraints is satisfiable iff there is a choice of disjuncts that has a clash-free saturation.

The only difficulty in handling *negations* is to deal with negated labeling constraints, as a formula containing negations can clearly be reduced to an equivalent one where the only negations are single negations of atomic formulae, and we already know what to do with negated dominance constraints. We can get rid of negated labeling constraints as well by replacing them with satisfiability equivalent formulae that do not contain negated labeling constraints. If the signature is finite, we can replace a constraint $\neg X:f(X_1, \dots, X_n)$ by

$$\left(\bigvee_{g \neq f \in \Sigma} X:g(X'_1, \dots, X'_{\text{ar}(g)}) \right) \vee \left(X:f(X''_1, \dots, X''_n) \wedge \bigvee_{i=1}^n \neg X''_i = X_i \right),$$

where the X'_i and X''_i are fresh variables. Now all we need to show is that a negated labeling constraint φ is satisfied by a pair (\mathcal{M}, α) iff its encoding φ' is satisfied by (\mathcal{M}, α') , where α' agrees with α on α 's domain.

This construction does not work for infinite signatures since the first disjunction would become infinite. Except in pathological cases, however, negated labeling constraints can be eliminated in this case as well, at the price of additional case distinctions.

Taking all the results from this section together, we have shown:

Proposition 3. *The satisfiability problem of the positive existential fragment over dominance constraints (and, of course, all smaller languages) is in NP.*

4 NP-Hardness of Dominance Constraints

As we have just seen, the satisfiability problems of all languages over dominance constraints which are sublanguages of the positive existential fragment are in NP. In this section, we complement this result by showing that even for purely conjunctive dominance constraints, this problem is NP-hard. To this end, we reduce the 3SAT problem to the satisfiability problem of dominance constraints. We only sketch the proof, as the main construction is quite intuitive, and further details provide no further insight. Together with the result from the previous section, we obtain the following result:

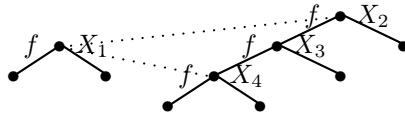
Theorem 1. *The satisfiability problems of all logical languages over dominance constraints between the (purely conjunctive) constraint language and the positive existential fragment are NP-complete.*

3SAT, a classical NP-hard problem, is the satisfiability problem of propositional formulae in conjunctive normal form where every conjunct is a disjunction of exactly three literals. This special type of conjunctive normal form is called 3-CNF.

$$\begin{array}{ll} \text{formulae} & \psi = C_1 \wedge \dots \wedge C_m \\ \text{clauses} & C_i = L_{i1} \vee L_{i2} \vee L_{i3} \\ \text{literals} & L_{ij} = X_k \text{ or } \neg X_k. \end{array}$$

We assume that the variables that occur in ψ are X_1, \dots, X_n .

The reduction is by encoding formulae in 3-CNF as satisfaction equivalent dominance constraints. The central problem that we must overcome is to model clauses without using disjunctions. We do this by using *dominance triangles*, subconstraints whose graphs look like this:



If $(\mathcal{M}^\tau, \alpha)$ is a solution of such a constraint, then α must map exactly one of the variables X_2, X_3, X_4 to the same node as X_1 because $\alpha(X_2)$ must be a prefix of $\alpha(X_1)$, which in turn must be a prefix of $\alpha(X_4)$. We can exploit this effect to model three-way disjunction – just what we need to encode a clause.

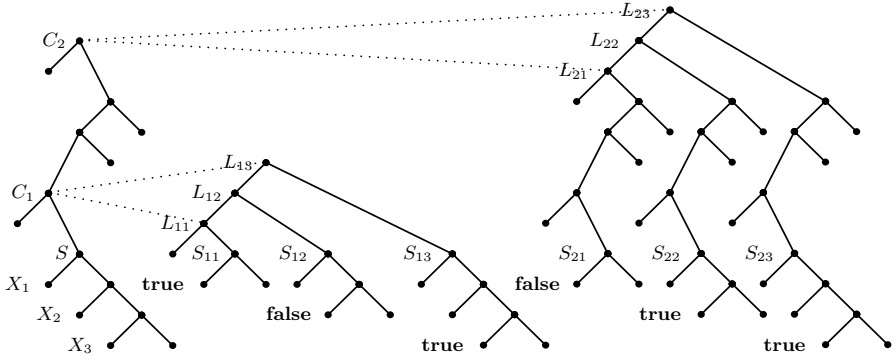


Fig. 2. An encoding of $(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee X_3)$ as a dominance constraint.

4.1 An Example of the Reduction

As an example of a 3-CNF formula, consider the following formula ψ :

$$(2) \quad (X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee X_3)$$

The constraint graph in Fig. 2 represents the dominance constraint φ which is the encoding of ψ . We are drawing the constraint graph in a somewhat simplified manner by leaving away all labels of inner nodes and most variable names; all inner nodes should be read as being labeled with a fixed binary constructor f . The signature we use is $\{f:2, \mathbf{true}:0, \mathbf{false}:0\}$, but the proof remains valid if the signature contains more labels.

We claim that φ is satisfiable iff ψ is satisfiable. To understand this, let us take a closer look at the various parts of the diagram.

The lower left part of the graph (below the node S) holds a variable assignment: For each of the variables X_k that occur in ψ , there is one node. In a solution, each of these nodes must be labeled with either **true** or **false**, but not both.

We can view ψ as a constraint on admissibility of variable assignments by calling a variable assignment admissible if it satisfies ψ . Each clause imposes such a restriction on the variable assignments; within a clause, we have a choice between three different options for satisfying the constraint.

The dominance constraint expresses the very same thing.

Because it is part of a dominance triangle, C_1 must be identified with one of the L_{1j} in any solution. But once we have identified C_1 with one of the three L_{1j} , we have decided which of the clause C_1 's literals we want to satisfy: The right daughter of the chosen L_{1j} node is identified with S , some entries in the variable assignment subtree may be skipped, and then a value restriction is imposed on

one of the variables X_k . In the example, L_{11} forces the label of X_1 to be **true**; L_{12} forces the label of X_2 to be **false**; etc. We have imposed a constraint on the variable assignment that is obviously equivalent to that imposed by the first clause in ψ .

The second clause is represented in a similar way: The dominance triangle between L_{21} , C_2 , and L_{23} allows a choice which literal of this clause we want to satisfy. Whichever literal we pick, its right daughter first skips the entry for C_1 (identifying S with one of the S_{2j}), and then it selects a variable entry and imposes a value constraint.

An important detail of this encoding is the presence of more nodes than just the C_i in the main branch of the graph (for example, there are two additional nodes between C_1 and C_2 in the constraint graph). These nodes are “rubbish dumps” which can be used to store unneeded material in such a way that it won’t interfere with anything else. Suppose we identified C_1 and L_{12} in a solution of φ . Then L_{11} will be identified with the left daughter of C_1 , and L_{13} will be identified with the mother of C_1 . Clearly, we do not want any other part of the constraint to say anything about the right child of C_1 ’s mother because otherwise, we might run into unnecessarily unsatisfiable dominance constraints. This means that above each C_i node, we need two additional nodes to drop material from the identification process. We do not need any additional nodes below the C_i because the unnecessary material is then a left child of the selected literal node and can safely be stored below C_i ’s left daughter.

4.2 The Reduction in the General Case

Now that we have made the intuition clear, we define the encoding in a more systematic way.

We build the constraint graph that corresponds to φ from the “building blocks” in Fig. 3. Larger building blocks can include several copies of smaller building blocks. For most of the building blocks, we have specified with arrows an upper and a lower attachment site where it can be composed with other blocks by identifying the two attachment sites; we write such compositions as trees whose labels are the two building blocks. Furthermore, we take a block with a superscript s (such as Skip with superscript $i - 1$ in X_i) to mean s -fold composition of building blocks. So we want the X_i block to consist of $i - 1$ occurrences of Skip blocks and two additional nodes that are immediate children of the lowest attachment site in the sequence of Skips, the left of which is labeled with **true**. It is easy to see that the constraint graph from the previous section was built according to this scheme. The overall structure consists of m entries for the clauses, below which n Skip blocks hold a variable assignment. Within each C_i block, there is a dominance triangle that allows the selection of a literal, together with a sufficient number of SkipC blocks to skip lower clauses. Finally, the encoding of a literal selects a propositional variable and imposes a value restriction.

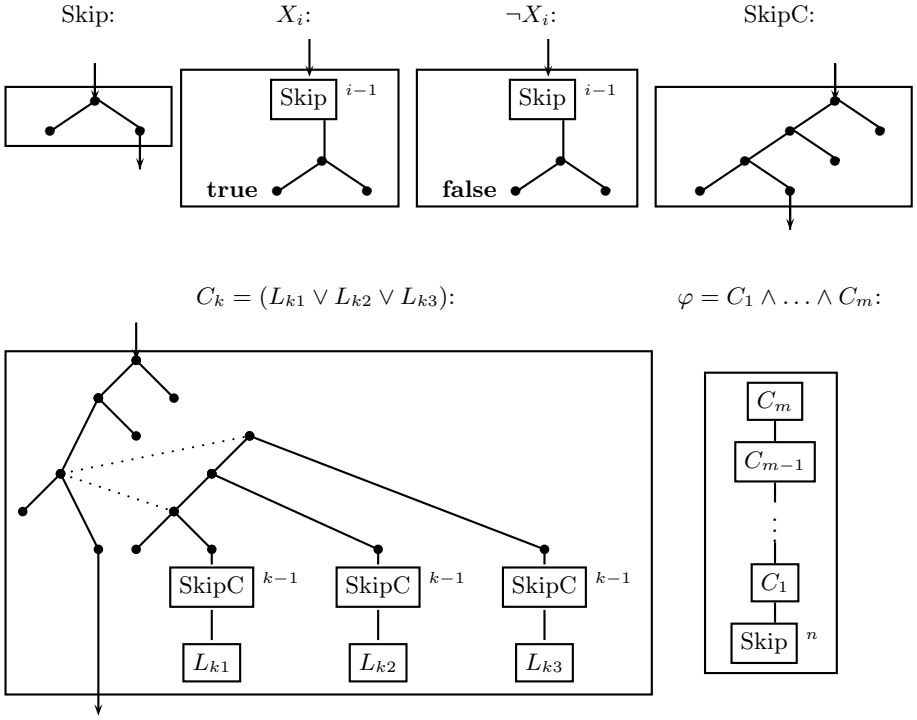


Fig. 3. Building blocks for the encoding of 3SAT as a dominance constraint.

The intuitive explanation from the beginning of this section should make clear that this encoding is correct. For a formal proof, we can encode valuations satisfying a 3-CNF formula ψ as tree structures and variable assignments satisfying the encoding of ψ , and vice versa. The gory details of this construction can be found in [8].

5 The First-Order Theory of Dominance Constraints

In the sections so far, we have focused on propositional languages over dominance constraints. Now we allow all propositional and first-order connectives (over the same set of atomic constraints) and consider the validity problem of first-order formulae Φ over dominance constraints. We first show a direct proof of the decidability of this problem for the case of bounded arity by reduction to second-order monadic logic. Afterwards, we show that the problem has non-elementary complexity by reducing the equivalence problem of regular languages with complement to it. Both results hold true for validity both over finite and over arbitrary trees.

5.1 Reduction to Second-Order Monadic Logic

In this section we assume a (finite or infinite) signature with a bound n on the arities of the function symbols. We show how to reduce the theory of dominance constraints to $S(n+1)S$, the second-order monadic theory of $n+1$ successor functions. The reduction carries over verbatim to $WS(n+1)S$, so finiteness of the trees doesn't make a difference.

The language of $S(n+1)S$ contains first-order variables x, y, z, \dots and second-order monadic variables (i.e., variables denoting sets) X, Y, Z, \dots , a binary relation symbol \in , a constant ϵ , and for every $0 \leq i \leq n$, a unary function symbol i . The universe of the corresponding structure is the set $\{0, \dots, n\}^*$ of words over the alphabet $\{0, \dots, n\}$, where ϵ denotes the empty word, and a function symbol i is interpreted as $i(w) = wi$. The formula $x \in X$ is true iff the denotation of x is contained in the denotation of X . The *theory* $S(n+1)S$ is the set of all closed formulae valid in this structure. The theory $WS(n+1)S$ is defined by restricting the denotation of monadic second-order variables to *finite* sets. The decidability of these theories has been established in [19] for the case of $WS(n+1)S$ and [14] for the case of $S(n+1)S$. A function application $i(x)$ is usually written as $x.i$.

We encode tree structures as the denotations of set variables X . Both are sets of words, and we can easily express closure under prefix and left brother; the only challenge is how to encode labels and arities. Below, we will first write down an $S(n+1)S$ formula that characterizes (encodings of) tree structures. Once this is done, we can easily encode dominance and labeling constraints.

We assume that the function symbols of a given arity k are numbered $1, 2, \dots$ if there are infinitely many symbols of arity k and $1, \dots, a_k$ if there are only finitely many. If there are infinitely many function symbols of arity k , we write $a_k = \infty$.

We encode a tree τ as the following finite subset of $\{0, \dots, n\}^*$:

$$T_\tau = \{\pi 0^i \mid \pi \in D_\tau, i \leq \tau(\pi)\}$$

In words, we represent τ as a set T_τ of words in $S(n+1)S$ by first requiring that all words in the tree domain of τ are also in T_τ . Then we add 0-successors to signify the label: The label of the node π of τ is represented by the arity of the node π together with the length of the string of 0's attached to π in T_τ . That is, instead of requiring that the label should determine the arity of a node, we only have to make sure that the label which is indicated by the arity and the zeroes really exists (i.e. there are at most a_k zeroes).

We encode a closed first-order formula Φ over dominance constraints as a closed monadic second-order formula containing a second-order variable X which denotes the encoding T_τ of a tree model τ . To this end, we first axiomatize the general structure of an T_τ :

$$tree(X) = \forall x \bigwedge_{i=0, \dots, n} (x.i \in X \rightarrow x \in X) \quad (3)$$

$$\wedge \forall x \bigwedge_{i=2, \dots, n} (x.i \in X \rightarrow x.(i-1) \in X) \quad (4)$$

$$\wedge \forall x ((x \in X \wedge \neg \exists y x = y.0) \rightarrow x.0 \in X) \quad (5)$$

$$\wedge \forall x \left(\bigwedge_{i=1, \dots, n} x.0.i \notin X \right) \quad (6)$$

$$\wedge \forall x (x.1 \notin X \rightarrow x.0^{a_0+1} \notin X) \quad (7)$$

$$\wedge \forall x \bigwedge_{\substack{i=1, \dots, n \\ a_i < \infty}} (x.i \in X \wedge x.(i+1) \notin X \rightarrow x.0^{a_i+1} \notin X) \quad (8)$$

(In this formula, the expression $x.(n+1) \notin X$ should be read as **true**.)

The formula first says that X is a tree, that is, prefixed-closed (formula 3) and closed under left brother (formula 4). By formula 5, every “proper” tree node, that is, a word not ending on 0, has to be labeled. Formula 6, together with 3, ensures that if $x0w \in X$, then $w \in \{0\}^*$. Finally, the consistency of the label of a node with its number of children (in the sense discussed above) is expressed by formula 7 for the case of nullary function symbols, and by formula 8 for non-nullary function symbols.

To complete our translation, we need another auxiliary predicate, $treenode(x)$, which expresses that x is a “proper tree node” (i.e. not part of the encoding of a label). More exactly, under the assumption that $tree(X)$ holds, we will get that $treenode(x)$ holds exactly when $x \in X$ and $x \in \{1, \dots, n\}^*$:

$$treenode(x) \quad := \quad x \in X \wedge \neg \exists y x = y.0$$

The prefix predicate is expressed in $S(n+1)S$ as usual:

$$prefix(x, y) \quad := \quad \forall X \left((y \in X \wedge \bigwedge_{i=0, \dots, n} (\forall z. z.i \in X \rightarrow z \in X)) \rightarrow x \in X \right)$$

Finally, we encode an arbitrary first-order formula Φ over the dominance constraints as an $S(n+1)S$ -formula Φ_X by

1. relativizing all quantifiers by the predicate $treenode(\cdot)$;
2. replacing $x \triangleleft^* y$ by $prefix(x, y)$;
3. replacing $x:f(x_1, \dots, x_n)$ by $label_f(x, x_1, \dots, x_n)$,

where we define

$$label_f(x, x_1, \dots, x_n) = \bigwedge_{i=1, \dots, n} x_i = x.i \wedge x.0^f \in X \wedge x.0^{f+1} \notin X \wedge x.(n+1) \notin X$$

Theorem 2. *A first-order formula Φ over dominance constraints is valid in the class of all (resp. finite) tree structures iff the following formula is valid in $S(n+1)S$ (resp. $WS(n+1)S$):*

$$\forall X(\text{tree}(X) \rightarrow \Phi_X)$$

Using the signature consisting of a constant a and a binary f , the following formula is valid over finite trees but not valid over infinite trees:

$$\forall X, Y, Z(X:f(Y, Z) \rightarrow \exists V(V:a \wedge X \triangleleft^* V))$$

Correspondingly, the following $S3S$ -formula is valid in $WS3S$ but not in $S3S$:

$$\begin{aligned} \forall X(\text{tree}(X) \rightarrow \forall x, y, z(\text{treenode}(x) \wedge \text{treenode}(y) \wedge \text{treenode}(z) \wedge \\ \text{label}_f(x, y, z) \rightarrow \exists v(\text{treenode}(v) \wedge \text{label}_a(v) \wedge \text{prefix}(x, v)))) \end{aligned}$$

Corollary 2. *The theory of first-order formulae over dominance constraints with a signature of bounded arity is decidable.*

5.2 The First-Order Theory is Non-elementary

We recall that a problem has *non-elementary complexity* if there is no algorithm for it running in time bounded by $\exp_k(n)$ for any k , where $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$ (see, for instance, [7] for a survey).

Theorem 3. *The first-order theory of dominance constraints has non-elementary complexity, both in the case of the finite tree models and in the case of arbitrary tree models.*

Recall that we have assumed the signature to contain at least a constant and a binary function symbol. We show this theorem by a reduction of the following classical problem:

Theorem 4 (Stockmeyer and Meyer, [18]). *The problem whether two regular expressions formed with $1, 2$, concatenation, union and complement (interpreted with respect to $\{1, 2\}^*$) denote the same set is non-elementary.*

We define our syntax of regular expressions as

$$R ::= 1 \mid 2 \mid R \cup R \mid RR \mid \neg R$$

The language defined by the regular expression R is called \mathcal{L}_R .

Given two variables X and Y , we translate a regular expression R of this class into a formula $\varphi_{[R]}(X, Y)$ with free variables $\{X, Y\}$. Roughly, $\varphi_{[R]}(X, Y)$ expresses that in any of its solutions, X dominates Y , and the path between the two nodes is in \mathcal{L}_R .

$$\begin{aligned}
\varphi_{[1]}(X, Y) &= \exists Z (X : f(Y, Z)) \\
\varphi_{[2]}(X, Y) &= \exists Z (X : f(Z, Y)) \\
\varphi_{[R_1 \cup R_2]}(X, Y) &= \varphi_{[R_1]}(X, Y) \vee \varphi_{[R_2]}(X, Y) \\
\varphi_{[R_1 R_2]}(X, Y) &= \exists Z (\varphi_{[R_1]}(X, Z) \wedge \varphi_{[R_2]}(Z, Y)) \\
\varphi_{[\neg R]}(X, Y) &= X \triangleleft^* Y \wedge \neg \varphi_{[R]}(X, Y)
\end{aligned}$$

The following formula says that a tree is (an approximation of) the full binary tree. More precisely, if τ is a tree satisfying Bin , all of its inner nodes must be labeled with f and all of its leaves must be labeled with a .

$$Bin = \forall X (X : a \vee \exists Y \exists Z \ X : f(Y, Z))$$

Proposition 4. *For any tree structure \mathcal{M} and variable assignment α ,*

$$M, \alpha \models Bin \wedge \varphi_{[R]}(X, Y) \text{ iff exists } v \in \mathcal{L}_R : \alpha(Y) = \alpha(X) \cdot v.$$

Proof. This is proven by induction on R . The only non-trivial case is the complement. Let (\mathcal{M}, α) satisfy Bin . Then

$$\begin{aligned}
M, \alpha &\models \varphi_{[\neg R]}(X, Y) \\
&\Leftrightarrow M, \alpha \models X \triangleleft^* Y \wedge \neg \varphi_{[R]}(X, Y) && \text{by definition} \\
&\Leftrightarrow \alpha(X) \triangleleft^* \alpha(Y) \text{ and } \neg \exists v \in \mathcal{L}_R \ \alpha(Y) = \alpha(X) \cdot v && \text{by induction} \\
&\Leftrightarrow \exists v \in \mathcal{L}_{[\neg R]} \alpha(Y) = \alpha(X) \cdot v && (*)
\end{aligned}$$

The step $(*)$ is justified by the fact that in a solution of Bin , $\alpha(X) \triangleleft^* \alpha(Y)$ iff there is a word $v \in \{1, 2\}^*$ with $\alpha(Y) = \alpha(X) \cdot v$, and that such a v is *unique*. \square

We can finally reduce the equivalence problem of regular expressions to the theory of dominance constraints:

Lemma 4. *Let R_1 and R_2 be regular expressions, \mathcal{I} the class of infinite tree models and \mathcal{F} the class of finite tree models. Then we have that*

$$\mathcal{L}_{R_1} = \mathcal{L}_{R_2} \tag{9}$$

$$\Leftrightarrow \mathcal{I} \models Bin \rightarrow (\forall X \forall Y (\varphi_{[R_1]}(X, Y) \leftrightarrow \varphi_{[R_2]}(X, Y)) \tag{10}$$

$$\Leftrightarrow \mathcal{F} \models Bin \rightarrow (\forall X \forall Y (\varphi_{[R_1]}(X, Y) \leftrightarrow \varphi_{[R_2]}(X, Y)) \tag{11}$$

Proof. $(9) \Rightarrow (10)$: If $\mathcal{L}_{R_1} = \mathcal{L}_{R_2}$ then, by Proposition 4, the formulae $\varphi_{[R_1]}$ and $\varphi_{[R_2]}$ are equivalent in any model of Bin .

$(10) \Rightarrow (11)$: Immediate since $\mathcal{F} \subseteq \mathcal{I}$.

$(\neg 9) \Rightarrow (\neg 11)$: Let $v \in \mathcal{L}_{R_1} - \mathcal{L}_{R_2}$, and let τ be the binary tree of depth $length(v)$; that is, all nodes of depth $< length(v)$ are labeled with f , and all nodes of depth $length(v)$ are labeled with a . \mathcal{M}^τ , together with the variable assignment $\{X \mapsto \epsilon, Y \mapsto v\}$, satisfies $\varphi_{[R_1]}$ but not $\varphi_{[R_2]}$, as a consequence of Proposition 4. \square

6 Conclusion

In this paper, we have analyzed the complexity of various logical languages over dominance constraints. We have shown that the satisfiability problems of all languages between the purely conjunctive constraint language and the positive existential fragment are NP-complete. We have presented two algorithms for solving dominance constraints, the second of which has been implemented and works well for real-world examples. Finally, we have presented a new proof of the decidability of the first-order theory of dominance constraints with signatures of bounded arity by encoding it into $(W)SnS$, and we have shown that its complexity is non-elementary.

From a practical perspective, the most important of the logical languages over dominance constraints we have considered here is the purely conjunctive constraint language. Despite the NP-hardness result we have derived for this language, there are implementations that decide satisfiability very efficiently for constraints from scope underspecification. These implementations are either advanced variants of the second algorithm presented here or based on finite set constraints; either way, they follow the “propagate and distribute” strategy advocated by constraint programming.

The observation that the general intractability does not affect the behaviour of actual implementations suggests that the linguistically relevant dominance constraints all belong to a subclass with an easier satisfiability problem, and that our algorithm exploits this automatically. The NP-hardness proof in this paper can be invalidated by only considering *normal* constraints, which contain an *inequality constraint* between any two variables for which they contain a labeling constraint; but inequality constraints can be used to build another NP-hardness proof. So an exact characterization of this subclass is an open problem.

Acknowledgments. We would like to thank Manuel Bodirsky for implementing dominance constraints in Oz and Denys Duchier for having suggesting to do so via finite set constraints. It is a pleasure to thank all members (student or not) of the CHORUS project. The research reported here was supported by the SFB 378 at the Universität des Saarlandes and the Esprit Working Group CCL II (EP 22457).

References

1. Backofen, R., J. Rogers, and K. Vijay-Shanker. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language, and Information*, 4:5–39, 1995.
2. Bos, J. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143, 1996.
3. Duchier, D., and C. Gardent. A constraint-based treatment of descriptions. In *Proceedings of IWCS-3*, Tilburg, 1999.

4. Duchier, D., and J. Niehren. Solving dominance constraints with finite set constraint programming. Technical report, Universität des Saarlandes, Programming Systems Lab, 1999.
<http://www.ps.uni-sb.de/Papers/abstracts/DomCP99.html>.
5. Egg, M., J. Niehren, P. Ruhrberg, and F. Xu. Constraints over Lambda-Structures in Semantic Underspecification. In *Proceedings COLING/ACL'98*, Montreal, 1998.
6. Gardent, C., and B. Webber. Describing discourse semantics. In *Proceedings of the 4th TAG+ Workshop*, Philadelphia, 1998. University of Pennsylvania.
7. Johnson, D. S. A catalog of complexity classes. In Leeuwen, J. van, editor, *Handbook of Theoretical Computer Science, vol A: Algorithms and Complexity*, chapter 2, pages 67–161. Elsevier, 1990.
8. Koller, A. *Constraint languages for semantic underspecification*. Diplom thesis, Universität des Saarlandes, Saarbrücken, Germany, 1999.
<http://www.coli.uni-sb.de/~koller/papers/da.html>.
9. Koller, A., and J. Niehren. Constraint programming in computational linguistics. Submitted. <http://www.coli.uni-sb.de/~koller/cpcl.html>, 1999.
10. Koller, A., and J. Niehren. Scope underspecification and processing. Lecture Notes, ESSLLI '99, Utrecht, 1999.
<http://www.coli.uni-sb.de/~koller/papers/esslli99.html>.
11. Marcus, M. P., D. Hindle, and M. M. Fleck. D-theory: Talking about talking about trees. In *Proceedings of the 21st ACL*, pages 129–136, 1983.
12. Meyer-Viol, W., and R. Kempson. Sequential construction of logical forms. In *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics*, Grenoble, France, 1998.
13. Muskens, R. Underspecified semantics. Technical Report 95, Universität des Saarlandes, Saarbrücken, 1998. To appear.
14. Rabin, M. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.
15. Reyle, U. Dealing with ambiguities by underspecification: construction, representation, and deduction. *Journal of Semantics*, 10:123–179, 1993.
16. Rogers, J. *Studies in the Logic of Trees with Applications to Grammar Formalisms*. PhD thesis, University of Delaware, 1994.
17. Rogers, J., and K. Vijay-Shanker. Reasoning with descriptions of trees. In *Proceedings of the 30th ACL*, pages 72–80, University of Delaware, 1992.
18. Stockmeyer, L. J., and A. R. Meyer. Word problems requiring exponential time. In *5th Annual ACM Symposium on the Theory of Computing*, pages 1–9, 1973.
19. Thatcher, J. W., and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2:57–68, 1968.
20. Vijay-Shanker, K. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18:481–518, 1992.

Strict Compositionality and Literal Movement Grammars

Marcus Kracht

II. Mathematisches Institut
Freie Universität Berlin
Arnimallee 3
D – 14195 Berlin
Germany
`kracht@math.fu-berlin.de`

Abstract. The principle of compositionality, as standardly defined, regards grammars as compositional that are not compositional in an intuitive sense of the word. There is, for example, no notion of a part of a string or structure involved in the formal definition. We shall therefore propose here a stricter version of compositionality. It consists in a conjunction of principles which assure among other that complex signs are in a literal sense made from simpler signs, with the meaning and syntactic type being computed in tandem. We shall argue that given this strict principle, quite powerful string handling mechanisms must be assumed. Linear Context Free Rewrite Systems (see [13]) are not enough to generate human languages, but most likely Literal Movement Grammars will do.

A grammar is *compositional* if the meaning of a (complex) expression is determined from the meaning of its (immediate) parts together with their mode of combination. A language is *compositional* if there is a compositional grammar that generates it. (See [6] for a discussion.) Recently, Zadrozny [15] has presented a proof that any meaning assignment function can go with a compositional grammar. This proof has been dismissed by Kazmi and Pelletier [7] and Westerståhl [14] on the grounds that Zadrozny is computing the meaning through a detour. He introduces new functions, one per word, and reduces the requirement of compositionality to the solution of an equation, which always exists if one assumes the existence of non-well founded sets. In fact, Zadrozny himself finds this solution formalistic and proposes restrictions under which certain pathological examples are excluded. So, there is an agreement that one should not count the proof by Zadrozny as showing us anything about the problem of compositionality of language — if the latter is understood in an intuitive sense. But in what sense can or must compositionality be understood if it is to have any significance? And what is the source of the complaints that people like Kazmi, Pelletier, Westerståhl (and others) raise? In this paper we will try to elaborate a formal setup of language as a semiotic system that allows to discuss this problem in a meaningful way. We shall propose a definition of compositionality, called strict compositionality, which is restrictive and therefore non vacuous.

Before we enter the discussion, we need to provide some basic definitions. Let F be a set (possibly empty) and Ω a function from F to the set ω of natural numbers. Throughout this paper, F will always be assumed to be finite. The pair $\langle F, \Omega \rangle$ (often denoted just by Ω) is called a *signature*. A (partial) Ω -algebra is a pair $\mathfrak{A} = \langle A, I \rangle$, where A is a non-empty set (called the *carrier set* of \mathfrak{A}) and I is a function with codomain F , which assigns to each $f \in F$ a (partial) $\Omega(f)$ -ary function on A . A *homomorphism* from a \mathfrak{A} to some (partial) Ω -algebra $\mathfrak{B} = \langle B, J \rangle$ is a function $h : A \rightarrow B$ such that for all $f \in F$ and all $\mathbf{a} \in A^{\Omega(f)}$:

$$h(I(f)(\mathbf{a})) = J(f)(h(\mathbf{a})) .$$

(Here, $h(\mathbf{a}) = \langle h(a_0), \dots, h(a_{\Omega(f)-1}) \rangle$.) This means that the left hand side is defined iff the right hand side is defined, and if both sides are defined they are equal. If $B \subseteq A$ and h is the identity map we speak of \mathfrak{B} as a *subalgebra* of \mathfrak{A} . In that case one can show that $J(f) = I(f) \upharpoonright B^{\Omega(f)}$. Let $X \subseteq A$ be an arbitrary subset of A . We denote by $[X]$ the least subset of A that contains X and is closed under all partial functions $I(f)$. If X has cardinality n and $[X] = A$, we say that \mathfrak{A} is *n-generated*. In particular, \mathfrak{A} is 0-generated if $A = [\emptyset]$.

Fix a set $V := \{x_i : i \in \omega\}$. An Ω -term is defined inductively as follows. Every $x_i \in V$ is a term; if $f \in F$ and $t_i, i < \Omega(f)$, are terms, so is $f(t_0, t_1, \dots, t_{\Omega(f)-1})$. Now, given an algebra $\langle A, I \rangle$, the set of polynomials is the set of Ω_A -terms, where Ω_A is obtained from Ω by adding a nullary function symbol \underline{a} for each $a \in A$, which is interpreted by a in A .

We assume that a language is a set of *signs*. In accordance with the syntactic literature, a sign is a triple, consisting of an exponent (this is what you can actually see of the sign), a type, and a meaning.

Definition 1 Let E, T and M be (nonempty) sets. A sign over (E, T, M) is a member σ of the Cartesian product $E \times T \times M$. If $\sigma = (e, t, m)$, we call e the exponent of σ , t the type of σ and m the meaning of σ . A language over (E, T, M) is a set of (E, T, M) -signs.

We denote the first projection from a sign by ε , the second projection by τ and the third projection by μ . Given a language L , then $\varepsilon[L] := \{\varepsilon(\sigma) : \sigma \in L\}$ is the set of exponents of L . When there is no risk of confusion we shall also speak of $\varepsilon[L]$ as a *language*. This covers the traditional usage of a subset of A^* being a language. Here are some examples of signs.

$$\begin{array}{ll} \text{'a'} & : \quad \langle \mathbf{a}, np/n, \lambda\mathcal{P}.\lambda\mathcal{Q}.(\exists x)(\mathcal{P}(x) \wedge \mathcal{Q}(x)) \rangle \\ \text{'man'} & : \quad \langle \mathbf{man}, n, \lambda x.\mathbf{man}'(x) \rangle \\ \text{'walks'} & : \quad \langle \mathbf{walks}, np/t, \lambda x.\mathbf{walk}'(x) \rangle \end{array}$$

Notice that the name of the sign (eg 'man') can be anything, even a number. What we can actually see of the sign is its exponent, ie **man**. (We write in typewriter font the exponent of a sign. The symbols in typewriter font are therefore true letters in print, while any other symbol is only proxy for some letter or string. Typewriter fonts are our device for quoting a string in running text without having to use quotation marks. Notice that 'man' is used to refer to the complete sign rather than its exponent.)

Definition 2 A (E, T, M) -grammar G consists of a finite signature Ω and functions I_E, I_T and I_M such that $\mathfrak{E} := \langle E, I_E \rangle$, $\mathfrak{T} := \langle T, I_T \rangle$ and $\mathfrak{M} := \langle M, I_M \rangle$ are partial Ω -algebras. The language of G , $L(G)$, is defined by $L(G) := [\emptyset]$. We call F the set of modes of G . A mode is proper if it has nonzero arity. The lexicon of G is the set $\{\langle I_E(f), I_T(f), I_M(f) \rangle : \Omega(f) = 0\}$.

This definition needs some exegesis. We think of E, T and M as independent sets, given in advance. At present, there are no conditions on these sets. A language is any subset of $E \times T \times M$. A grammar is a system by which the signs of L are built from some finite set of signs (the lexicon) by means of some finite set of functions (called proper modes). Therefore, modes correspond to grammatical rules. For example, if f is a binary mode, one thinks of f as the following grammatical rule

$$f(\sigma_1, \sigma_2) \rightarrow \sigma_1 \ \sigma_2$$

Notice that terminal symbols of a grammar are generally not considered to be rules (they are part of the lexicon), but there is no harm in thinking of them as rules of the form $\sigma \rightarrow \cdot$ (think of the arrow as the $:=$ in **Prolog**). Notice that our definition of grammar is modular: not any set of functions from signs to signs qualifies. Rather, we assume that the modes operate independently on the exponents, the types and the meanings of the signs. Therefore, in order to define a grammar, one needs only to specify the interpretation of the modes in each of the three sets E, T and M independently. This defines the algebras $\mathfrak{E}, \mathfrak{T}$ and \mathfrak{M} . The rest is completely determined. One forms the product $\mathfrak{E} \times \mathfrak{T} \times \mathfrak{M}$. This algebra contains a unique minimal subalgebra. Its carrier set is precisely the set of all signs that can be produced from the lexicon by the set of proper modes. If L has a grammar, it is also a partial Ω -algebra \mathfrak{L} in some canonical way. We should note that if L is a (E, T, M) -language, it is also a (E', T', M') -language for any $E' \supseteq E, T' \supseteq T$ and $M' \supseteq M$. However, there is a different sense of extension of a language that will play a role in the discussion.

Definition 3 Let L be a (E, T, M) -language and L' a (E', T', M') -language. L' extends L if $L' \cap E \times T \times M = L$.

If L' extends properly L , one may think of L' as having *hidden signs*. Such a sign is of the form $\langle e, t, m \rangle$ where either $e \notin E$ or $t \notin T$ or $m \notin M$. It may be difficult to argue for a sign with exponent $e \in E$ to be hidden. Anyhow, we shall argue below that one should not postulate hidden signs. But for the moment, we leave this possibility open. The following definition embodies the notion of compositionality as employed in the literature.

Definition 4 Let L be a (E, T, M) -language. L is naturally compositional if there is a (E, T, M) -grammar G such that $L(G) = L$. L is compositional if there is a L' which extends L and is naturally compositional.

Notice that the notion of extension allows the introduction of new exponents or new types or new meanings if necessary. However, no new (E, T, M) -signs may be introduced. To see why this is the standard approach, notice that the

problem is stated as follows. A language is defined a subset L of $E \times M$. The pair $\langle e, m \rangle$ corresponds to a sentence e with meaning m . With this datum, a grammar is written, introducing new types (for intermediate constituents) and assigning arbitrary meanings to signs containing them. However, in natural languages the natural parts of a sentence — the constituents — do have their own meaning, and so we cannot choose that meaning arbitrarily. Moreover, the meaning of the constituent contributes to the meaning of the sentence. Furthermore, a context-free grammar is often defined as a string generating device of the following form. The symbol S is the start symbol. Starting with S , one may replace step by step each nonterminal by the right side of an appropriate rule. In our view, however, this approach suffers from a confusion between the type of a sign and its exponent. The symbol NP is not part of the string-algebra, which contains only sequences of terminal strings. Hence, NP is a type. In the course of a derivation we actually derive triples $\langle x, NP, m \rangle$, where x is a string of type NP and meaning m .

Nevertheless, if the set of types is finite, \mathfrak{L} may in fact be thought of as a many sorted algebra over sound-meaning pairs. However, the introduction of the types has the advantage to eliminate having to type the signs, and it allows to incorporate grammars with explicit type constructors, such as categorial grammars. If \cdot is a mode, we write \cdot_e in place of $I_E(\cdot)$, \cdot_t in place of $I_T(\cdot)$ and \cdot_m in place of $I_M(\cdot)$. By our definitions, \cdot is defined on a tuple of signs exactly when all three functions \cdot_e , \cdot_t and \cdot_m are defined on the corresponding projections. This means that the partiality is introduced by the algebras of exponents, signs and meanings. One fundamental assumption is made.

Feasibility. If \cdot is a mode, then the corresponding functions \cdot_e , \cdot_t and \cdot_m are computable.

Moreover, in Montague Grammar the functions are actually polynomials in the basic functions of the algebras of strings, types and meanings, respectively. We shall assume the same here. This means that they can be expressed in λ -notation, but we shall suppress λ -notation whenever possible.

Polynomial Feasibility. If \cdot is a mode, then the corresponding functions \cdot_e , \cdot_t and \cdot_m are polynomial functions. Moreover, each of the functions is computable in polynomial time.

Notice that it is not clear that a polynomial function is computable in polynomial time. If, say, \mathfrak{M} consists in the domain ω of natural numbers and a unary function $f : \omega \rightarrow \omega$ which is not computable, then f is a polynomial function but is not polynomially computable. The results rely only partly on this restriction of polynomial time computability.

In the simplest case, the grammar has only one mode, \bullet , called the *merge*. It is given by the functions \bullet_e , \bullet_t and \bullet_m , which are defined by

$$\begin{aligned} \varepsilon(\sigma_1 \bullet \sigma_2) &= \varepsilon(\sigma_1) \bullet_e \varepsilon(\sigma_2) \\ \tau(\sigma_1 \bullet \sigma_2) &= \tau(\sigma_2) \bullet_t \tau(\sigma_1) \\ \mu(\sigma_1 \bullet \sigma_2) &= \mu(\sigma_1) \bullet_m \mu(\sigma_2) \end{aligned}$$

Let \bullet_e be concatenation of strings with a blank (\square) inserted, \bullet_t be slash-cancellation and \bullet_m be function application, and we get Montague–Grammar. To give an example, we may compose the signs ‘a’ and ‘man’. This gives the sign $\sigma := \text{‘a’} \bullet \text{‘man’}$. Now,

$$\begin{aligned}\varepsilon(\sigma) &= \varepsilon(\text{‘a’}) \bullet_e \varepsilon(\text{‘man’}) \\ &= \mathbf{a} \wedge \square \wedge \mathbf{man} \\ &= \mathbf{a\ man}\end{aligned}$$

Likewise, $\tau(\sigma) = np$ and $\mu(\sigma) = \lambda Q.(\exists x)(\mathbf{man}'(x) \wedge Q(x))$ are established. Hence we get

$$\sigma = \langle \mathbf{a\ man}, np, \lambda Q.(\exists x)(\mathbf{man}'(x) \wedge Q(x)) \rangle$$

A word on notation. We shall assume that the exponents of signs are strings or sequences thereof. These strings are represented in exactly the same way as they are written. So, each word is actually already a sequence (of letters) and concatenation of words is done by putting a blank in between the two. We use \square to denote the blank. Plain concatenation is denoted by \wedge , or, if no confusion arises, it is denoted simply by concatenation. Word concatenation is denoted by \cdot . We have $x \cdot y = x \wedge \square \wedge y$.

Suppose that G generates L . Since $L = [\emptyset]$, any sign of the language can be represented by a constant term. This term is called its *representing term*. Terms uniquely designate derivations. So, if for a given exponent x there are m terms representing a sign with exponent x then x is said to be *m-fold structurally ambiguous*. The reader should bear in mind that the term denotes the signs only with the grammar being given.

The definitions allow that the exponents of signs may be anything we please. Yet, if by exponent we mean the visible (or audible) exponent of the sign, there is little sense in assuming that the exponents of signs are anything but strings. This is, with a minor adaptation, what we shall assume here. It has been shown that given any recursively enumerable language and any computable function assigning meanings to strings, there is a type assignment and a compositional grammar generating this language (see [6] for details). However, there is a sense in which some of these grammars may fail to be compositional in an intuitive sense. We shall therefore say that a grammar G is a *strict grammar of L* if it satisfies the following two requirements. (Two other principles will still follow, but we assume them to follow from the next two, if not in the literal sense, then at least in spirit.)

Naturalness. $L(G) = L$.

Analyticity. There is a finite set A such that $E \subseteq A^*$. For each mode f and each sequence σ on which f is defined there is a string polynomial $p(\mathbf{x})$, in which each \mathbf{x}_i , $i < \Omega(f)$, occurs at least once, such that

$$f_e(\varepsilon(\sigma)) = p(\varepsilon(\sigma))$$

(Here, $\varepsilon(\sigma) = \langle \varepsilon(\sigma_i) : i < \Omega(f) \rangle$. Notice that $f_e(\varepsilon(\sigma)) = \varepsilon(f(\sigma))$, by the assumption that f is defined on σ .)

Definition 5 *A language L is strictly compositional if it has a strict compositional grammar.*

These principles need a certain amount of motivation. The principle of *Naturalness* is stronger than requiring that $L(G)$ extends L' , which is commonly used in the definitions of compositionality. If G is natural for L , L is actually naturally compositional, as defined earlier. So, notice that we require that all signs that the grammar generates are part of the language. In particular, the signs generated have exponents that are units of the language, and the grammar assigns those types and meanings to them that they have in that language. What signs there are is of course an empirical question. It is to a large extent also a question of methodology or personal persuasion. Not everyone will accept that λ -terms count as meanings for the words of natural language. However, even if the existence and nature of signs is unclear the question of what they are is not without meaning at all. Just like Augustinus observed with respect to time, there is a perennial problem with respect to meaning. It seems that we know perfectly well what meaning is, but when asked to give a definition, we fail. The present discussion is therefore not targeted at the question of what signs there are; rather, granted that we know that, how can a grammar for them look like? To emphasize our point once more: if the question whether languages are compositional is to have any nontrivial meaning at all, it is because we exclude the introduction of new signs.

The *Principle of Analyticity* means the following. If a sign σ is directly derived from σ_i , $i < n$, then the exponents of the σ_i are disjoint parts of the exponent of σ . We have phrased this using the language of string polynomials. It should be stressed that the polynomial may depend on the mode as well as the sequence of signs. However, if it is independent of the choice of the signs, we call the grammar *uniformly analytic*.

Definition 6 *A grammar is uniformly analytic if for every mode f there exists a string polynomial p , in which each variable x_i , $i < \Omega(f)$, occurs at least once, such that for all sequences σ of signs on which f is defined the equation*

$$f_e(\varepsilon(\sigma)) = p(\varepsilon(\sigma))$$

holds.

However, the interpretation of the mode in A^* , f_e , need not be identical to p , it may just be a subset of p (since f_e may be a strictly partial function). In fact, *Uniform Analyticity* comes down to the existence of a polynomial p of the appropriate kind such that $f_e = p \upharpoonright \text{dom}(f_e)$.

EXAMPLE 1. To illustrate our point we shall discuss the example of Kazmi and Pelletier in [7]. Let $A = \{a, b, c\}$, be the alphabet and let the set of exponents be $\{a, b, c, ca, cb\}$. Now we introduce a meaning assignment, which has the property that $\mu(a) = \mu(b)$ but $\mu(ca) \neq \mu(cb)$. Now, suppose first that there is only one type of expression. Then it is easily seen that there exists no strictly compositional grammar for that language. This is independent of whether we assume *Polynomial Feasibility*. For there simply is no function \cdot_μ whatsoever that satisfies

$$\begin{aligned}\mu(\mathbf{ca}) &= \mu(\mathbf{c}) \cdot_{\mu} \mu(\mathbf{a}) \\ \mu(\mathbf{cb}) &= \mu(\mathbf{c}) \cdot_{\mu} \mu(\mathbf{b})\end{aligned}$$

So, restricting the class of functions to exclude this example — as Zadrozny proposes — is not necessary at all under the assumption of strict compositionality. We shall briefly outline two alternatives to the present example to shed light on the kind of restrictions that are operative here. First, we could have said that \mathbf{a} , \mathbf{b} , \mathbf{c} , \mathbf{ca} and \mathbf{cb} are the exponents of primitive signs and that there is no proper mode, just five 0-ary modes. This is unintended in the example above, since we write \mathbf{ca} to imply that the sign with exponent \mathbf{ca} is composed from the signs with exponents \mathbf{c} and \mathbf{a} , respectively. Notice however that many languages have words that can be segmented as strings into ‘unnatural’ parts, in which case the impossibility to have a compositional account of the meaning of that word does simply not arise since the word as a sign is not conceived of as consisting of these parts. To give an example, the word ‘selfish’ is not the result of composing ‘sell’ and ‘fish’, although that (almost) sounds the same. Also the word ‘caterpillar’ is not a compound built from ‘(to) cater’ and ‘pillar’. Likewise, idioms must be considered as units from a semantic point of view. Evidently, they can often also be read literally (in which case they are decomposed), but this is not at issue here. Thus, the notion of a basic sign (ie a 0-ary mode) is different from the notion of a sign having a nonsegmentable exponent. A second variation on that theme is to allow the same string to have several types. We could, for example, allow any string to denote itself when of type, say, s . In that case we have two signs with the same exponent, for example

$$\langle \mathbf{a}, t, \mu(\mathbf{a}) \rangle, \langle \mathbf{a}, s, \mathbf{a} \rangle$$

In this situation, a compositional grammar in the strict sense once again exists. Just assume two binary modes \circ and \bullet :

$$\begin{aligned}\langle x, s, m \rangle \circ \langle y, s, n \rangle &= \langle xy, s, m \wedge n \rangle \\ \langle x, s, m \rangle \bullet \langle y, s, n \rangle &= \langle xy, s, \mu(m \wedge n) \rangle\end{aligned}$$

So, we can either compose two strings qua strings, in which case we do concatenation on the semantic side. Or we can compose strings qua meaningful entities — in which case we shall step from the string to the corresponding meaning. (Actually, \bullet alone would have sufficed here. One can also introduce a unary mode that transports a string to its meaning.) In this way, once again a compositional grammar exists for any language whatsoever. Or so it seems. What should be observed, however, is that we are not giving a grammar for the same language, since a language is a set of signs, and we have expanded the set of signs to include strings of type s . But how about human languages? We claim that in human languages it is not possible to have a string denote its usual meaning in addition to denoting itself. In the latter case we call the string quoted. There exist devices that allow to quote a string qua string, so actually we *do* have strings as meanings in our language. (Just anything can be the meaning of an expressions.) But the string x can never denote itself (ie x). Rather it must occur in a context that quotes it. For example, we distinguish in writing between **man** — the exponent of a sign whose meaning is, say, $\lambda x.\mathbf{man}'(x)$ — and ‘**man**’ (using object language

quotation marks here), the exponent of a sign whose meaning is **man**. So, it is never x itself that denotes x but, for example, the string x enclosed in quotes. Of course, it is only natural for a semiotic system not to allow for the possibility that its signs are all self-denoting, which would mean that we probably didn't need the signs in the first place. And precisely this saves us from vacuity.

The *Principle of Analyticity* is needed to be able to talk meaningfully about parts of a sign. Hence, we can talk about constituents and of a constituent analysis. As the principle stands, it is not unproblematic. First, we shall have to talk of occurrences of strings. The *Principle of Analyticity* is to be understood to require that $\varepsilon(\sigma)$ is made from the $\varepsilon(\sigma_i)$, and that each $\varepsilon(\sigma_i)$ occurs at least once. So, there is no deletion of any material whatsoever. Assuming for the moment that the exponents are strings, the function \cdot_e shall simply be a semigroup polynomial in n -variables such that each variable occurs at least once. Examples are

$$p(x, y) := xxy, \quad q(x, y) := yxy\mathbf{a}$$

Here, **a** is a certain constant (the letter **a**). The *Principle of Analyticity* does allow for empty categories. However, we shall apply Occam's Razor and assume

Nonemptiness. No sign has empty exponent.

Now follow some more examples of grammars.

EXAMPLE 2. Numbers are written as sequences of digits, where a digit is a member of $\{0, 1, \dots, 9\}$. The sequence $x_{k-1}x_{k-2}\dots x_0$ represents the number $\sum_{i=0}^{k-1} \mu(x_i) \cdot 10^i$, where $\mu(x_i)$ is the number assigned to each digit. There are therefore two types: *digits* (D) and *sequences* (S). The following is a strict grammar. These are the nullary modes:

$$\begin{aligned} '0' & : \langle 0, D, 0 \rangle \\ '1' & : \langle 1, D, 1 \rangle \\ & \dots \\ '9' & : \langle 9, D, 9 \rangle \end{aligned}$$

There is a unary mode $*$ and a binary mode \circ . $*_e(x) := x$, $*_m(\mu) := \mu$, and $*_t(D) := S$. $*_t(S)$ is undefined. For the binary mode \circ we have $x \circ_e y := xy$, $S \circ_t D := S$. \circ_e is undefined otherwise. $m \circ_m n := 10m + n$. So, the term $*'7'$ corresponds to the sign $\langle 7, S, 7 \rangle$, the term $'2' \circ ('3' \circ '7')$ to the sign $\langle 237, S, 237 \rangle$.

Since the algebra of types is finite, one can actually present the algebra of signs in form of the following context-free grammar.

$$\begin{aligned} \langle 0, D, 0 \rangle \mid \dots \mid \langle 9, D, 9 \rangle & \rightarrow \cdot \\ \langle xy, S, 10 \cdot m + n \rangle & \rightarrow \langle x, S, m \rangle \quad \langle y, D, n \rangle \\ \langle x, S, n \rangle & \rightarrow \langle x, D, n \rangle \end{aligned}$$

(Here, \mid denotes an alternative; the first line abbreviates therefore a total of ten rules.) Terminal symbols are denoted here by nullary rules. This grammar is left-regular.

EXAMPLE 3. As above. However, assign to a sequence x as meaning the pair $\nu(x) := \langle \ell(x), \mu(x) \rangle$.

$$\begin{array}{ll}
 \langle 0, D, \langle 1, 0 \rangle \rangle \mid \dots \mid \langle 9, D, \langle 1, 9 \rangle \rangle & \rightarrow \cdot \\
 \langle yx, T, \langle \ell_1 + \ell_2, 10 \cdot m_1 + m_2 \rangle \rangle & \rightarrow \langle x, T, \langle \ell_1, m_1 \rangle \rangle \quad \langle y, D, \langle \ell_2, m_2 \rangle \rangle \\
 \langle x, T, \langle \ell, n \rangle \rangle & \rightarrow \langle x, D, \langle \ell, n \rangle \rangle \\
 \langle x, S, n \rangle & \rightarrow \langle x, T, \langle \ell, n \rangle \rangle
 \end{array}$$

This is a compositional grammar. However, it is not strict. The problem is that the meaning of a string x simply *is* the number it represents, we are not allowed to add any more to it. It can be shown that there is no right-regular strict grammar that generates the number sequences. For a proof note that the sequences 7, 07, 007 etc all have the same meaning, namely 7. However, the result of prefixing 1 is different in all cases. We get 17, 107, 1007 etc, which all represent different numbers. However, since we have only finitely many types in a right-regular grammar, some of the sequences 7, 07, 007 etc must have equal type, and therefore the grammar generates two different sequences starting with 1, which are assigned the same number. This is, however, incorrect.

It is tempting to conclude that a strict grammar for the number sequences must be left regular. This is more or less correct, but there are infinitely many grammars that can generate the number sequences compositionally even in the strict sense and each is different in the constituent analysis that it introduces. Nevertheless, the constituent analysis cannot be right regular. Strict grammars must satisfy the condition that not both a string x and $0x$ are generated having identical type. So, even if the constituent analysis is not uniquely defined by the principles laid out above, there nevertheless are certain things that can be said about possible constituents.

The present definitions do not say anything about the size of the algebra of types. It may be finite or infinite. However, notice that if a grammar has infinitely many signs then the same exponent can in principle have infinitely many meanings depending on which type it has. Moreover, strings can be infinitely ambiguous, simply because they can be derived from some the same string using some unary modes. Since we do not want to exclude the number of types to be infinite, we shall rather require that unary modes must also change the exponent. Although this does not follow from *Analyticity* in a literal sense, it is nevertheless a principle of the same sort, since it requires that every step leaves a visible trace on the exponent.

Productivity. If σ is composed from τ by applying a unary mode, then $\varepsilon(\tau)$ is shorter than $\varepsilon(\sigma)$.

This means that there can be no unary rules that only change the category (type) and the meaning; rather, a unary mode must introduce material into the string. The grammar in Example 2 does not comply with this restriction. There is an easy fix for that. Just assume the following additional 0-ary modes:

$$\begin{array}{ll}
 \text{'0\#'} & : \langle 0, S, 0 \rangle \\
 \text{'1\#'} & : \langle 1, S, 1 \rangle \\
 & \dots \\
 \text{'9\#'} & : \langle 9, S, 9 \rangle
 \end{array}$$

Now eliminate the last rule and add instead the rule

$$\langle xy, S, 10m + n \rangle \rightarrow \langle x, D, m \rangle \langle y, D, n \rangle$$

This looks like a bad trick but is in fact quite logical. The digit 3 for example, is actually the exponent of two signs, that of the *digit* 3 and that of the *sequence* consisting of 3. This grammar reflects the dual nature of sequences of length 1. Sequences of length > 1 can of course not be digits.

There is plenty of evidence that in language there are empty signs and also non productive modes. However, their use must obviously be highly restricted otherwise the determination of the meaning from the sound can become infeasible. So, when one looks closely at the matter it often enough appears that the use of empty signs and non productive modes can be eliminated in much the same way as it can be done in context free grammars.

EXAMPLE 4. This example concerns the English number names, in a slightly simplified form. This example goes back to Arnold Zwicky, for a discussion of the formal complexity of English and Chinese number names see [11]. Each language has a largest primitive name for a number. This number varies from language to language. Let us assume that it is *million* for English. Numbers of the form 10^{6k} are represented by the k -fold iteration of the word *million*. The number 2000003000005 is therefore represented by the string

two million million three million five

We shall leave out the words *thousand*, *hundred* as well as *ten*, *twenty* etc and assume that our alphabet consists only of

$$\{\text{zero, one, two, } \dots, \text{nine, million}\}$$

Legitimate expressions have the following form:

$$x_0 \cdot \text{million}^{k_0} \cdot x_1 \cdot \text{million}^{k_2} \cdot \dots \cdot x_{m-1} \cdot \text{million}^{k_{m-1}}$$

where $k_0 > k_1 > \dots > k_{m-1}$ and $x_j \neq \text{million}$ for all $j < m$. This sequence represents the number

$$\sum_{j=0}^{m-1} x_j \cdot 10^{6k_j}$$

This language is not generable by Linear Context Free Rewrite System (otherwise known as LCFRSs); in fact, it is not even semilinear. However, it is recognizable in polynomial time. We shall propose two quite similar grammars. Each of the two have the following 0-ary modes:

‘0’ : $\langle \text{zero}, D, 0 \rangle$	‘0#’ : $\langle \text{zero}, S, 0 \rangle$
‘1’ : $\langle \text{one}, D, 1 \rangle$	‘1#’ : $\langle \text{one}, S, 1 \rangle$
\dots	\dots
‘9’ : $\langle \text{nine}, D, 9 \rangle$	‘9#’ : $\langle \text{nine}, S, 9 \rangle$
‘m’ : $\langle \text{million}, M, 10^6 \rangle$	‘0†’ : $\langle \text{zero}, SE, 0 \rangle$
	‘1†’ : $\langle \text{one}, SE, 1 \rangle$
	\dots
	‘9†’ : $\langle \text{nine}, SE, 9 \rangle$

Here, D is the type of a digit, S that of a number expression, and SE that of a *simple expression*, where a simple expression is of the form $x \cdot \text{million}^k$, x a digit. Now, there is a binary mode(s) \bullet and \bullet^S , operate as follows:

$$\begin{aligned}\langle x, SE, m \rangle \bullet \langle y, M, n \rangle &:= \langle x \cdot y, SE, m \cdot n \rangle \\ \langle x, SE, m \rangle \bullet^S \langle y, M, n \rangle &:= \langle x \cdot y, S, m \cdot n \rangle\end{aligned}$$

Finally, the mode \circ is defined as

$$\langle x, S, m \rangle \circ \langle y, SE, n \rangle := \langle x \cdot y, S, m + n \rangle$$

However, as the definitions stand the grammar generates all sequences of simple expressions, not necessarily in decreasing order. To implement the latter restriction, we have two choices: (a) we define \circ_m to be $m \circ_m n := m + n$ if $m > n$, and $m \circ_m n$ undefined otherwise, (b) we define $x \circ_e y := x \cdot y$ if x ends in a larger block of **million** than y , and $x \circ_e y$ is undefined otherwise. The proposals differ slightly. If we disallow expressions of the form **zero million**^k then they are actually equivalent, otherwise option (a) gives incorrect results. The trouble with both proposals is that we need to introduce strange partial functions. However, notice that the definition of a grammar did not tell us what the basic functions of the partial algebras are. So, rather than taking a simple merge as the (only) basic operation, we may also take merge functions as basic which require properties of strings to be checked. This requires careful formulation. We shall have to require that these functions be computable in polynomial time, otherwise Theorem 8 below is actually incorrect. In this case, if we are operating on the string algebra, testing whether one string is a substring of the other certainly takes only polynomial time. We come to our main definition:

Definition 7 *A grammar is strictly compositional (or strict) if it is analytic, polynomially feasible, has no nonempty signs and is productive. A language is strictly compositional if it has a strictly compositional grammar.*

The present restrictions can be shown to be nontrivial. Before we engage in the proof we have to fix a last detail. We shall assume that the algebra operates not necessarily on strings but on sequences of strings. Moreover, these sequences shall have bounded length, say k . It is beyond the scope of this paper to motivate exactly why we depart from the ideal model that the exponents are strings. There are basically two arguments: (a) if we allow only strings then there does not seem to be a feasible algorithm to generate those languages that are not context-free, (b) strings are not continuous but are naturally segmented (eg by pauses). Of course, it is always possible to introduce a boundary marker into the string which would function the same way. We have felt it technically more clean to allow sequences of strings. The relation between the sequences of strings and the strings themselves is fixed by the following requirement.

Vectorization. The string associated with an exponent $\langle x_i : i < n \rangle$ is its concatenation $x_0 x_1 \dots x_{n-1}$.

Hence, we shall assume that the exponents of the grammar depart only mildly from strings, namely, they are allowed to be strings with some gaps. We call a grammar *vectorized* if it uses sequences of strings rather than strings.

Theorem 8 *Let \mathfrak{A} be a strictly compositional vectorized grammar for L . Then the following holds:*

1. *Given a string x it can be decided in polynomial time whether it belongs to the language.*
2. *Given a string x , there are at most exponentially many derivations for x .*

Proof. The algorithm is an adaptation of the chart method. First, notice that any derivation has length at most $|x|$, where $|x|$ denotes the length of x . A representing term for x has therefore at most $|x|$ mode symbols. It is now easy to see that x is at most exponentially ambiguous. Now for the first claim. Let $n := |x|$. By analyticity, the sequence $\langle x_i : i < k \rangle$ must consist of disjoint substrings of x . (This is easily established by induction.) There exist less than n^2 substrings, and hence less than n^{2k} k -sequences of substrings. In the first step, try to match a sequence against the exponent of a 0-ary mode. This takes polynomial time. This gives the list L_0 . Now, given L_i , let L_{i+1} be the result of applying all possible modes to the members of L_i and matching the result against x . x corresponds to some exponent of a sign in L iff there is a $\sigma \in L_i$ for some $i < n$ such that the exponent (or its product) is x . Now, computing L_{i+1} from L_i takes time polynomial in n . For there are at most n^{2k} members, and there are finitely many modes. Each application of a single mode takes polynomial time (by polynomial feasibility). We have to compute L_i only for $i < n$. This concludes the proof of the first claim. Q. E. D.

(The polynomial bounds computed here are rather bad. They suffice for the argument, however.) So, strictness *is* restrictive. The question therefore arises: which languages are strict and which ones are not? In particular: are natural languages at all strictly compositional in the sense of the definition? We believe that the answer to the second question is positive. Notice that the rather tight constraints on putting together signs do not allow to dissociate the meaning composition and the string handling. For example, devices such as a Cooper-storage are inadmissible since they dissociate the syntactic structure from the semantic structure by introducing new meanings, something which is strictly prohibited. It may well be that Categorical Grammar takes us out of that problem by associating enough types with a string to cover its potentials in a context. If we are not so happy with having infinitely many types, however, the only way to surround this is to postulate stronger string handling mechanisms. One promising proposal that has been made recently are the so-called Literal Movement Grammars (LMG) as have been introduced by Annius Groenink in [5]). An LMG has rules of the following form

$$A(\gamma) \rightarrow B_0(\delta_0) B_1(\delta_1) \dots B_{n-1}(\delta_{n-1})$$

where A and the B_i are nonterminals, here called *types*, and γ and δ_i sequences of polynomials over the alphabet, possibly using variables. Since there are only finitely many rules, the length of these sequences is bounded. We may therefore assume that they all have the same length, adding empty strings if necessary. The advantage of LMGs is that they add explicit rules for handling the strings. By adding a third dimension, the meaning, we can turn an LMG into a grammar of signs. We call an *interpreted LMG* a grammar that has rules of the form

$$A(\gamma, q(\mu_0, \dots, \mu_{n-1})) \rightarrow B_0(\delta_0, \mu_0) B_1(\delta_1, \mu_1) \dots B_{n-1}(\delta_{n-1}, \mu_{n-1})$$

where μ_i are variables for meanings and q is a polynomial function $\mathfrak{M}^n \rightarrow \mathfrak{M}$.

An easy example of an LMG is the following

$$S(xx) \rightarrow S(x); \quad S(\mathbf{a}) \rightarrow .$$

This grammar generates the language $\{\mathbf{a}^{2^n} : n \in \omega\}$. It is shown in [5] that any recursively enumerable language can be generated by an LMG. Therefore, these grammars are very powerful. However, certain subclasses can be identified. According to [5], an LMG is *bottom up nonerasing* if each variable on the right hand side occurs at least once on the left hand side. An LMG is *bottom up linear* if each variable on the right hand side occurs at most once on the left hand side. An LMG is *noncombinatorial* if each term on the right hand side consists of a single variable. Finally, an LMG is *simple* if it is *bottom up linear*, *bottom up nonerasing* and *noncombinatorial*. Now the following holds

Theorem 9 (Groenink) *Let $L \subseteq A^*$ be a language. L is PTIME-recognizable iff it can be generated by a simple LMG.*

Now, this does not mean of course that a PTIME-recognizable sign grammar can be generated by a simple interpreted LMG. But we shall present evidence below that natural languages can be generated by (more or less) simple LMGs. Notice that in the definition of *bottom up nonerasingness* one thing must be added: first, one should talk of constants, not only variables which occur on the right hand side. However, for noncombinatorial grammars this is obviously unnecessary.

Simple LMGs are not necessarily analytic, but if they are, they are uniformly analytic. For example, a clause of the form

$$A(x) \rightarrow B(x) C(x)$$

can occur in a simple LMG, but if we read the rules as modes this is unacceptable. The condition of analyticity requires instead that each variable occurs to the left hand side at least as often as it occurs on the right hand side. Furthermore, [5] disallows a variable to occur more than once to the left, but shows that this condition can be circumvented. We shall therefore say that an LMG is *analytic* if it is (a) noncombinatorial and (b) every variable occurs on the left hand side of a production at least as often as it occurs on the right hand side. Languages generated by analytic LMGs are also generated by simple LMGs but the converse need not hold. Notice that our first example of an LMG is actually analytic. However, it is not simple. Yet, it is easily transformed into a simple LMG (see also [5]):

$$\begin{array}{llll} S(xy) & \rightarrow & S(x) S(y) E(x, y); & S(\mathbf{a}) \rightarrow . \\ E(\mathbf{a}, \mathbf{a}) & \rightarrow & . & ; E(x_1 y_1, x_2 y_2) \rightarrow E(x_1, x_2) E(y_1, y_2) \end{array}$$

We notice that in Example 4, assuming that we work with pairs of strings rather than strings, under a suitable reformulation we only need to assume the existence of a binary string function of the following form:

$$p(x, y) := \begin{cases} x & \text{if } x = y \\ \uparrow & \text{else} \end{cases}$$

(Here, \uparrow means that the function is undefined.)

We shall illustrate the potential of this proposal by giving an analysis of two benchmark examples. One is the cross-serial dependencies. The other are the languages with iterated cases. A particular example of the latter kind was analyzed in [8], where it was shown that the language generated in this example was not semilinear, hence not generable by a LCFRS (or a Multi-Component TAG, for that matter). (For a reference on LCFRSs see [13].)

EXAMPLE 5. The following LMG generates the cross-serial dependencies of Dutch.

$$\begin{array}{ll} VR(\mathbf{zien}) & \rightarrow \cdot \\ VR(\mathbf{laten}) & \rightarrow \cdot \\ NP(\mathbf{Jan}) & \rightarrow \cdot \\ NP(\mathbf{Piet}) & \rightarrow \cdot \\ V(\mathbf{zwemmen}) & \rightarrow \cdot \\ VC(x_1 \cdot y_1, x_2 \cdot y_2) & \rightarrow NP(x_1) VR(y_1) VC(x_2, y_2) \\ VC(x, y) & \rightarrow V(y) NP(x) \end{array}$$

It is straightforward to transform this LMG into a sign grammar. We basically need in addition to the nullary modes, a binary mode \bullet and a ternary mode \circ . The binary mode is defined as follows.

$$\langle \langle x_1, x_2 \rangle, NP, f \rangle \bullet \langle \langle y_1, y_2 \rangle, V, g \rangle := \langle \langle x_1 \cdot x_2, y_1 \cdot y_2 \rangle, VC, f(g) \rangle$$

\circ operates as follows.

$$\begin{aligned} \circ(\langle \langle x_1, x_2 \rangle, NP, f \rangle, \langle \langle y_1, y_2 \rangle, V, g \rangle, \langle \langle z_1, z_2 \rangle, VC, \mathcal{P} \rangle) \\ := \langle \langle z_1 \cdot x_1 \cdot x_2, z_2 \cdot y_1 \cdot y_2 \rangle, VC, g(f)(\mathcal{P}) \rangle \end{aligned}$$

The semantics of a raising verb, here **zien** (English ‘to see’) is as follows:

$$\lambda x. \lambda \mathcal{P}. \lambda y. \mathbf{see}'(y, \mathcal{P}(x))$$

This generates the cross-serial dependencies. If we want to give an analysis of the German verb cluster or of the analogous English construction, we only have to modify the string polynomial \circ_e , nothing else needs to be changed. The analysis of cross-serial dependencies is similar to that of Calcagno [2], where a larger fragment is analyzed using head-wrapping, which was introduced in Pollard [10]. Calcagno also discusses the relationship with a proposal by Moortgat [9], who uses string equations. String equations are one way to try to avoid the use of vectorization. It is therefore worthwhile to see why it does not work. Suppose we have the following rule

$$A(u \cdot x, v \cdot y) \rightarrow B(u, v) C(x, y)$$

Then this must be replaced in Moortgat’s system by the following rule:

$$A(p) \rightarrow B(q) C(r) \quad : \quad p = u \cdot x \cdot v \cdot y, q = u \cdot v, r = x \cdot y.$$

This means that if an A is analyzed as a B plus a C then the strings associated with A , B and C should satisfy the string equations shown to the right. The trouble with string equations, as Calcagno rightly points out, is that they come down to an a posteriori analysis of a single string into several components, which may or may not reflect the actual composition of the string. String equations are simply not analytic (in our sense of the word). Head-grammars and LCFRSs on the other hand mark explicit places for inserting strings, thus reflecting the actual construction of the string rather than just any other. Notice that LMGs allow both for string equations and for vectorization, but string equations are not allowed in an analytic or a simple LMG.

EXAMPLE 6. We finally discuss the stacked genitives of Old Georgian (see [8]). Old Georgian displays a phenomenon called *Suffixaufnahme* or *Double Case*. Suffixaufnahme is found in many Australian languages, and it is said to be iterable beyond limitation (at least in some languages with respect to certain constructions, to be exact, see [4], [3] for examples). Old Georgian provides a generic case of Suffixaufnahme that is iterable (see Boeder [1]).

gov-el-i igi sisxl-i saxl-isa-j m-is
all-NOM Art-NOM blood-NOM house-GEN-NOM Art-GEN
Saul-is-isa-j
Saul-GEN-GEN-NOM
All the blood of the house of Saul

We will give a compositional LMG for this construction. However, we will simplify the matter. Full NPs shall have no article, the nominative is always -j and the genitive does not appear in its short form -is. The grammar manipulates triples $\langle x, y, z \rangle$, where x is the first noun, y its stack of case suffixes and z the remaining NP. First we write the plain LMG. (Recall here that we distinguish plain concatenation (\frown) from word concatenation (\cdot)).

- | | | |
|---|---------------|---|
| (1) $NPc(x, y, \varepsilon)$ | \rightarrow | (1) $NP(x, \varepsilon, \varepsilon)$ $Case(\varepsilon, y, \varepsilon)$ |
| (2) $Case(\varepsilon, \text{isa}, \varepsilon)$ | \rightarrow | . |
| (3) $Case(\varepsilon, j, \varepsilon)$ | \rightarrow | . |
| (4) $NPs(x, y, z)$ | \rightarrow | $NPc(x, y, z)$ |
| (5) $NPs(x, y \frown \text{isa}, \varepsilon)$ | \rightarrow | $NPs(x, y, \varepsilon)$ |
| (6) $NPs(x, y \frown j, \varepsilon)$ | \rightarrow | $NPs(x, y, \varepsilon)$ |
| (7) $NPs(x_1, y_2, x_2 \frown \text{isa} \frown y_2 \cdot z_2)$ | \rightarrow | $NPs(x_1, y_2, \varepsilon) NPs(x_2, \text{isa} \frown y_2, z_2)$ |

Notice that the suffix sequence in the last rule occurs twice on the left and twice on the right hand side. This grammar is analytic (modulo massaging away unproductive rules). The semantic functions are as follows. For (4), (5) and (6) the corresponding function is the identity. For (1) and (7) it is application of the right argument to the left; (3) has the interpretation $\lambda\mathcal{P}.\mathcal{P}$ (\mathcal{P} of type $\langle e, t \rangle$), but this is done for the purpose of exposition only. (2) has the semantics

$$\lambda\mathcal{P}.\lambda\mathcal{Q}.\text{belong}'(y, x) \wedge \mathcal{P}(x) \wedge \mathcal{Q}(y)$$

For the nouns we take $\lambda x.\text{blood}'(x)$, $\lambda x.\text{house}'(x)$ and $\lambda x.x \doteq \text{saul}'$ as interpretations. They have type NP . Our target example is

sisxlj saxlisaj Saulisaisaj

Here are the translations of the nouns with stacked genitives:

sisxlj	:	$\lambda x.\text{blood}'(x)$
saxlisaj	:	$\lambda Q.\text{belong}'(y, x) \wedge \text{house}'(x) \wedge Q(y)$
Saulisaisaj	:	$\lambda Q.\text{belong}'(y, x) \wedge x \doteq \text{saul}' \wedge Q(y)$

Notice that only the inner layer of case marking is semantically operative. The outer layers do not change the meaning. Now we compose these words. By the rules of the grammar, only the last two words can be analyzed as a constituent:

$$\lambda Q.\text{belong}'(y, x) \wedge x \doteq \text{saul}' \wedge \text{belong}'(y, z) \wedge \text{house}'(x) \wedge Q(z))$$

This can now be composed with the first word; this gives the following translation for our example.

$$\text{belong}'(y, x) \wedge x \doteq \text{saul}' \wedge \text{belong}'(z, y) \wedge \text{house}'(y) \wedge \text{blood}'(z)$$

For this to work properly, substitution must be defined correctly. That we have free variables here is just an artifact of the simplicity of the fragment and could of course be avoided.

The last example also demonstrates how languages with stacked case marking can be treated. However, we shall note here that LMGs cannot handle such languages if they have completely free word order. It has been confirmed by Alan Dench (p. c.) that those languages which have the most extensive iterated case marking system do not allow for free word order beyond the clause boundary. Given that free word order within a clause can in principle be accounted for compositionally using LMGs — as we believe — this gives evidence that LMGs have enough string handling capacity. To show this is however beyond the scope of this paper. We shall only note that languages with stacked cases cannot simply be treated using a function that compares strings of cases, since the exponents of cases may actually be different. This means that the string handling component must in these examples rely on rather delicate functions, which are however computable in linear time. [12] has argued that German allows scrambling across any number of clause boundaries. If that is so, German could also not be handled by an interpreted LMG compositionally (in the strict sense). There is however every reason to believe that the arguments cannot follow in any order whatsoever. Rather, free word order is only present when the arguments are sufficiently distinguishable either morphologically (by their case endings) or semantically (animate vs inanimate). Otherwise, we claim, word order is fixed. If we are right, also German is not so exceptional after all.

References

1. Boeder, W. Suffixaufnahme in Kartvelian. In Plank, F., editor, *Double Case. Agreement by Suffixaufnahme*, pages 151 – 215. Oxford University Press, 1995.
2. Calcagno, M. A Sign-Based Extension to the Lambek Calculus for Discontinuous Constituents. In *Bulletin of the IGPL*, 3:555 – 578, 1995.

3. Dench, A. Suffixaufnahme and Apparent Ellipsis in Martuthunira. In Plank, F., editor, *Double Case. Agreement by Suffixaufnahme*, pages 380 – 395. Oxford University Press, 1995.
4. Evans, N. D. *A Grammar of Kayardild. With Historical-Comparative Notes on Tangkic*. Mouton de Gruyter, Berlin, 1995.
5. Groenink, A. *Surface without Structure. Word order and tractability issues in natural language processing*. Ph.D. thesis, Utrecht University, 1997.
6. Janssen, T. Compositionality. In van Benthem, J. and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 417 – 473. Elsevier, Amsterdam, 1997.
7. Kazmi, A. and F. J. Pelletier. Is Compositionality Formally Vacuous? In *Linguistics and Philosophy*, 21:629 – 633, 1998.
8. Michaelis, J. and M. Kracht. Semilinearity as a syntactic invariant. In Retoré, C., editor, *Logical Aspects of Computational Linguistics (LACL '96)*, number 1328 in Lecture Notes in Computer Science, pages 329 – 345. Springer, Heidelberg, 1997.
9. Moortgat, M. Generalized quantifiers and discontinuous type constructors. In Sijtsma, W. and A. van Horck, editors, *Discontinuous Constituency*. Mouton de Gruyter, Berlin, 1993.
10. Pollard, C. J. *Generalized Phrase Structure Grammar, Head Grammars and Natural Language*. Ph.D. thesis, Stanford University, 1984.
11. Radzinski, D. Chinese Number Names, Tree Adjoining Languages, and Mild Context-Sensitivity. In *Computational Linguistics*, 17:277 – 299, 1991.
12. Rambow, O. *Formal and Computational Aspects of Natural Language Syntax*. Ph.D. thesis, University of Pennsylvania, 1994.
13. Seki, H., T. Matsumura, M. Fujii, and T. Kasami. On multiple context-free grammars. In *Theoretical Computer Science*, 88:191 – 229, 1991.
14. Westerståhl, D. On Mathematical Proofs of the Vacuity of Compositionality. In *Linguistics and Philosophy*, 21:635 – 643, 1998.
15. Zadrozny, W. From Compositional Semantics to Systematic Semantics. In *Linguistics and Philosophy*, 17:329 – 342, 1994.

Categorial Minimalism

Alain Lecomte

LORIA, 615 rue du Jardin Botanique, VILLERS-LES-NANCY

`Alain.Lecomte@upmf-grenoble.fr`,

`http://www-bsh.m.upmf-grenoble.fr/~alecomte/`

Université Pierre Mendès France,

Bâtiment SHM,

F-38040 Grenoble Cedex, France

Abstract. In this paper, we try to put a bridge in between Categorial Grammar and Minimalist Grammars as they result from works which follow the Chomskyan enterprise (cf Stabler ([9]), Cornell ([3])). We show that weak and strong features can be replaced by special modalities which are used to control resource management.

1 Introduction

There is a growing interest in comparing the Minimalist Program ([2]) and the Categorial Grammar ([3], [6]). It has become particularly stronger since Stabler's formalization of minimalist ideas under the label of "Minimalist Grammars" ([9]), and since other works like those of T. Cornell ([3]), D. Heylen ([4]) and W. Vermaat (to appear). We are trying in this paper to develop a viewpoint based on a categorial type logic as proposed by M. Moortgat, N. Kurtonina, and D. Oehrle ([8]), on these minimalist grammars. It can be distinguished from other attempts by many respects. In contrast with a conception we have proposed elsewhere ([6]) which uses proof-nets as a representational device for expressing derivations, we are here using a purely "derivationalist" approach. In contrast with Cornell's approach ([3]), which is also strictly derivationalist, we don't start with an enumeration (in the Chomskyan sense) of resources trying to build by their means a correct sentence, but like in orthodox categorial grammars, we start from a sequent as a goal to demonstrate, the antecedent of which consists of a parenthesized phonological form.

Therefore, in some sense, we are developing a *reverse* conception with regards to the generative aspect of minimalist grammars. But of course, that belongs to the nature of categorial grammars themselves, as opposed to generative ones. Another aspect of our proposal concerns the treatment of the numerous functional categories Chomsky introduces in his representation of a sentence, which include AGRSP, AGROP, NEGP, TP, CP etc. Like it has been pointed out by some researchers ([1]), these nodes have not always a status equivalent to usual categories like NP (or DP), S, PP or AP. In fact, functional nodes often appear to provide mere targets for *moves*¹. It is the reason why we assume that such

¹ Bouchard ([1]) speaks of *the proliferation of categories that function strictly as escape hatches, or landing sites, for moved elements (such as AGR)*

nodes are replaced by *composition modes*, in the sense of [8] and [5]. Moreover, if moves are always explained by the need for checking features, we shall assume that such features are expressed by *modalities* (cf [4]) and are checked under their appropriate composition mode.

This brings us another distinction with regards to [3], where features are dealt with exactly like categories. We can summarize by saying that every deduction in the calculus is oriented towards a goal which is a sequent of the form $\Gamma \vdash F$ where Γ is a structured multiset of lexical resources and where F is a formula ². At the beginning, lexical resources are mere words that are replaced by more or less complex formulae in the course of the deduction, by means of a [lex]-rule. The structure itself is not known in advance and must be guessed. Because we want to stay close to the categorial tradition, which always starts from a set of types associated with words in order to reduce the sequence they form to a base-category, we are led to describe the reverse transformations w.r.t. the *move* operations of the minimalist grammars.

More explicitly, we try to study how constituents are moved from their overt position to the position from where they originate. These positions are simply those where they can be cancelled by the usual [/L] and [\L]-rules.

Of course, when reading the deductions in the top-down direction, we find back the usual directions for *moves*, that is a leftward orientation, towards higher positions in the P-marker.

The Multimodal calculus we present here is not a translation of Stabler's Minimalist Grammars in multimodal terms, even if it is supposed to have a similar generative power. Our main objective is to remain in the spirit of the Minimalist Program by avoiding as many spurious devices (like empty elements, empty nodes or empty types) as possible.

2 Resource Control

2.1 Modes and Modalities

Let us simply recall that on a set of grammatical resources, we can define several binary products and their residuals, and several unary operators and their residuals. For a non commutative binary product, the residuation law simply expresses that:

$$A \bullet B \rightarrow C \Leftrightarrow A \rightarrow C / \bullet B \Leftrightarrow B \rightarrow A \backslash \bullet C$$

For each unary operator \diamond , we can also canonically define a dual one \square such that:

$$\diamond A \rightarrow B \Leftrightarrow A \rightarrow \square B$$

We shall use a calculus with three products: \circ , \bullet and $*$. \bullet is neither commutative nor associative, its adjuncts will be the usual / and \. \circ is not strictly speaking commutative nor associative either, but interaction postulates between the two products will give us access to commutativity and associativity. Its adjuncts will be denoted $/^\circ$ and \backslash° , they will occur in the lifted types associated with some

² or a possible set of formulae, like we shall see later on.

lexical entries. The product $*$ will have also residuals, denoted by $/^*$ and \backslash^* , mainly used for semantic purposes.

The key point will be that a deduction has to switch from an occurrence of \bullet to an occurrence of \circ each time some move (ie some tree restructuring) has to be performed, and this will be done each time a *strong* modality gives access to \circ . So, only strong modalities will be responsible for overt moves. Moreover, the occurrence of \circ responsible for a move is lowered at each application of the corresponding structural postulate and replaced by an occurrence of \bullet at the upper position. We shall see later on the use of $*$.

In the Došen-style axiomatic presentation of resource-conscious logics, we shall assume the following postulates:

For strong modalities:

$$\diamond_c^S(A \bullet B) \rightarrow (\diamond_c^S A) \circ B \quad [\text{K1S}]$$

$$\diamond_c^S(A \circ B) \rightarrow (\diamond_c^S A) \circ B \quad [\text{K1}]$$

$$\diamond_c^S(A \circ B) \rightarrow A \circ \diamond_c^S B \quad [\text{K2}]$$

For weak modalities:

$$\diamond_c(A \bullet B) \rightarrow (\diamond_c A) \bullet B \quad [\text{K1W}]$$

$$\diamond_c(A \circ B) \rightarrow (\diamond_c A) \circ B \quad [\text{K1W}]$$

$$\diamond_c(A \bullet B) \rightarrow A \bullet \diamond_c B \quad [\text{K2W}]$$

$$\diamond_c(A \circ B) \rightarrow A \circ \diamond_c B \quad [\text{K2W}]$$

Communication between products:

$$A \circ B \rightarrow A \bullet B \quad [\text{Incl}]$$

$$A \circ (B \bullet C) \rightarrow (A \circ B) \bullet C \quad [\text{MA}]$$

$$A \circ (B \bullet C) \rightarrow B \bullet (A \circ C) \quad [\text{MC}]$$

$$A \circ B \rightarrow B \bullet A \quad [\text{Comm}]$$

if A and B are product-free

Comments:

- A **strong** composition mode \diamond_c^S gives license to a constituent to move. That means that when such a composition mode meets the \bullet -product, it is changed into the \circ -product. Because a strong mode *attracts* the corresponding feature to a "specifier" position in the structure, we assume that this feature is in the highest left position in the tree under the root affected by this composition mode. That amounts to distribute the mode only over the first conjunct.
- Of course, a strong mode can affect a \circ -product: that will happen every time another strong feature has already occurred and the constituent affected by the dual modal has not (yet) moved. In this case, the second strong mode is distributed either over the first conjunct (in case the constituent which has not moved is also provided with the modality corresponding to this new mode) or over the second one (in order to be transmitted later on to its first conjunct).

- A **weak** mode does not give any license to move and therefore if it meets a $\{\bullet, \circ\}$ -product, the sort of product is kept. The second case occurs when a \diamond_c has to "jump" over a strong feature which has not yet moved.
- The [Incl]-rule makes it possible to go back to \bullet at any moment.
- The [MA]-rule ensures that associativity can be performed only by exchanging the two products: the main one must switch from \circ to \bullet and the subordinate one from \bullet to \circ . In transformational terms, that will correspond to the lowering of A and its adjunction to B on its left.
- The [MC]-rule expresses a kind of mixed commutativity and in transformational terms to the lowering of A and its permutation with B.
- The [Comm]-rule is commutativity under the condition of changing \circ into \bullet and restricted to product-free formulae. This rule is intended to be used at the end of a sequence of permutation steps of the [MC]-kind.

At the beginning of the analysis, we have a tree-structured multiset of lexical resources: we assume the comma ", " be the structural counterpart of the \bullet -product. We also assume the following inclusion between strong and weak modalities:

$$\diamond_c^S A \rightarrow \diamond_c A$$

This allows the following cancellation to be performed:

$$\diamond_c^S \square_c A \rightarrow A$$

This means that a strong modality \diamond_c^S can cancel a weak modality \square_c , but not the other way round.

Apart from this *postulates package*, we recall here for memory, the usual rules we use for introducing each binary connective depending on a mode i , and each unary connective relative to a modality (*feature*) α in the sequent calculus format.

Rules:

$$\frac{\Theta \vdash B \quad \Gamma[A] \vdash C}{\Gamma[(A/_i B, \Theta)^i] \vdash C} [L/_i]$$

$$\frac{\Theta \vdash B \quad \Gamma[A] \vdash C}{\Gamma[(\Theta, B \backslash_i A)^i] \vdash C} [L \backslash_i]$$

$$\frac{(\Gamma, B)^i \vdash A}{\Gamma \vdash A/_i B} [R/_i]$$

$$\frac{(B, \Gamma)^i \vdash A}{\Gamma \vdash B \backslash_i A} [R \backslash_i]$$

$$\frac{\Gamma[(A, B)^i] \vdash C}{\Gamma[A \bullet_i B] \vdash C} [L \bullet_i]$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{(\Gamma, \Delta)^i \vdash A \bullet_i B} [R \bullet_i]$$

$$\frac{\Theta \vdash A \quad \Gamma[A] \vdash B}{\Gamma[\Theta] \vdash B} [cut]$$

$$A \vdash A[axiom]$$

$$\frac{\Gamma[A] \vdash B}{\Gamma[(\Box_{\alpha} A)^{\diamond}_{\alpha}] \vdash B} [L\Box_{\alpha}]$$

$$\frac{(\Gamma)^{\diamond}_{\alpha} \vdash B}{\Gamma \vdash \Box_{\alpha} A} [R\Box_{\alpha}]$$

$$\frac{\Gamma[(A)^{\diamond}_{\alpha}] \vdash B}{\Gamma[\Box_{\alpha} A] \vdash B} [L\Box_{\alpha}]$$

$$\frac{\Gamma \vdash A}{(\Gamma)^{\diamond}_{\alpha} \vdash \Box_{\alpha} A} [R\Box_{\alpha}]$$

Every postulate $F \rightarrow G$ may be set in the sequent format and then becomes a rule:

$$\frac{\Gamma \vdash C}{\Delta \vdash C}$$

where Δ is the structural counterpart of F and Γ the structural counterpart of G .

2.2 Lexicon and Goals

Lexical entries. They are made of categorial types using / and \ (and later on /*, *, /° and \°). and modalities representing their features. Examples:

$$aime ::= \Box_{agrV} \Box_{infl} ((np \backslash s) / np) \quad Paul ::= \Box_{agrN(m,s,3)} \Box_k np$$

The interpretation for $\Box_k np$ is: a np waiting for a case.

Goals. The sequents we want to demonstrate have antecedents made of parenthesized strings of words and a consequent consisting of a unique modalized formula, like for instance: $\Box_{nom}^S \Box_{infl}^S s$, this expressing the fact that we want to reduce the structured resources in the antecedent to a category s which needs an inflection and the assignment of a nominative case.

Of course, there can be situations with no accusative case, and others with more cases. In fact, for any arbitrary sentence, there are a priori several modalized types to which it can be reduced, but among them only one which is the true type, this depending on the type of the main verb, the presence or absence of negation and so on. But these possibilities are very limited, they form a *set*, like for instance (for an affirmative sentence):

$$\{\Box_{nom}^S \Box_{acc} \Box_{dat} \Box_{infl}^S s, \Box_{nom}^S \Box_{acc} \Box_{infl}^S s, \Box_{nom}^S \Box_{infl}^S s\}$$

From the parsing viewpoint, we can consider that a first phase consists in selecting the correct final type in the consequent simply by scanning the modalities which occur in the lexical types, in the antecedent.

Moreover, given a set of types \mathcal{S} , we shall write $\Gamma \vdash \mathcal{S}$ if and only if there exists some t such that $t \in \mathcal{S}$ and $\Gamma \vdash t$.

Because the $[\Box R]$ -rule is such that, in order to prove

$$(x_1, (x_2, \dots, x_n)) \vdash \Box s$$

we have to prove

$$((x_1, (x_2, \dots, x_n)))^{\diamond} \vdash s$$

this amounts to assume that the modalities \Diamond_{nom}^S and \Diamond_{infl}^S are in turn affected to the product on the left thus replacing the former categories IP and AGRP. Thus for instance, the grammaticality of the sentence *Peter loves Mary* is established by proving the sequent:

$$(Peter, (loves, Mary)) \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S$$

This amounts to successively prove the sequents:

$$\begin{aligned} & ((Peter, (loves, Mary))) \Diamond_{nom}^S \vdash \Box_{acc} \Box_{infl}^S \\ & ((\Box_k np, (loves, Mary))) \Diamond_{nom}^S \vdash \Box_{acc} \Box_{infl}^S \\ & ((\Box_k np) \Diamond_{nom}^S, (loves, Mary))^\circ \vdash \Box_{acc} \Box_{infl}^S \\ & \quad (np, (loves, Mary))^\circ \vdash \Box_{acc} \Box_{infl}^S \\ & \quad (np, (loves, Mary)) \vdash \Box_{acc} \Box_{infl}^S \\ & \quad ((np, (loves, Mary))) \Diamond_{acc} \vdash \Box_{infl}^S \\ & \quad (np, ((loves, Mary)) \Diamond_{acc}) \vdash \Box_{infl}^S \\ & \quad (np, (loves, (Mary) \Diamond_{acc})) \vdash \Box_{infl}^S \\ & \quad (np, (loves, (\Box_k np) \Diamond_{acc})) \vdash \Box_{infl}^S \\ & \quad \dots \end{aligned}$$

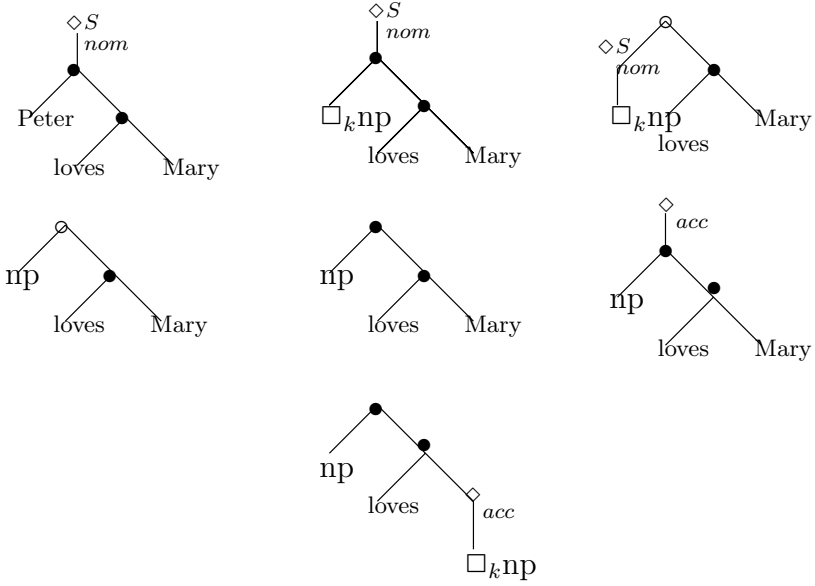
until we reach the sequent:

$$(np, ((np \backslash s) / np, np)) \vdash s$$

which obviously succeeds by $[/L]$ and $[\backslash L]$.

The first step consists in using the $[\Box R]$ -rule: the first mode which is used is then the \Diamond_{nom}^S -mode. The $[lex]$ -rule is used at the second step in order to replace the word *Peter* by its lexical type, asking for a case. At the third step, \Diamond_{nom}^S "opens" the \bullet -product by transforming it into the \circ one, and it is distributed by $[K1S]$ over the first conjunct. At the fourth step, the feature case is cancelled, by the $[\Box L]$ -rule. At the fifth step, the $[Incl]$ -rule allows us to go back to the \bullet -product. Notice that in fact the subject *np* has not moved: this is the option that will reveal to be correct, according to the type assigned to the verb. (We are not making useless moves). At the sixth step, a new cycle is beginning, with a new application of the $[\Box R]$ -rule: the second mode is the \Diamond_{acc} one. By means of several applications of $[K1W]$ and $[K2W]$, the \Diamond_{acc} -mode reaches the leaf labelled with the lexical type associated with *Mary*. At this moment still remains a new mode to be performed: the \Diamond_{infl} -mode. It will be used in order to "free" the verbal type. Finally, we get a sequent easily provable in a rough categorial grammar.

This history of moves can be represented as a series of tree-restructurings of the P-marker associated with the sentence. The following figure shows some of them. The last tree is obtained after several steps from the previous one.



In the next sections, we shall sometimes present these restructurings instead of the entire derivations by using the list representations of trees, for sake of brevity.

Minimality conditions. As often stated in the minimalist framework, minimality conditions are required to explain that phrases are moving towards their nearest target. Here, we shall assume the deductions the shortest as possible, counting their length by the number of applications of a structural rule ([Comm], [MC] or [MA]) they use.

3 Examples

3.1 Interrogatives

Let us take the example of an interrogative in French: *Quel écrivain Pierre aime*, with the assignments:

$$quel - \acute{e}crivain ::= \square_{wh} \square_k np \quad Pierre ::= \square_k np \quad aime ::= \square_{infl}((np \backslash s)/np)$$

(We omit here agreement features for simplicity)

The goal sequent is:

$$((quel, \acute{e}crivain), (Pierre, aime)) \vdash \square_{wh}^S \square_{nom}^S \square_{acc} \square_{infl}^S s$$

and a fragment of the proof is:

$$\begin{array}{c}
\dfrac{\dots}{((Pierre, (aime, \Box_k np)))^{\Diamond^S_{nom}} \vdash \Box_{acc} \Box_{infl}^S} [K1S] \\
\dfrac{((Pierre, (aime, \Box_k np)))^{\Diamond^S_{nom}} \vdash \Box_{acc} \Box_{infl}^S}{(Pierre, (aime, \Box_k np)) \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S} [\Box R] \\
\dfrac{(Pierre, (aime, \Box_k np)) \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S}{(Pierre, (\Box_k np, aime))^{\circ}} \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S [Comm] \\
\dfrac{(Pierre, (\Box_k np, aime))^{\circ}}{(\Box_k np, (Pierre, aime))^{\circ}} \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S [MC] \\
\dfrac{((\Box_{wh} \Box_k np)^{\Diamond^S_{wh}}, (Pierre, aime))^{\circ}}{((\Box_{wh} \Box_k np, (Pierre, aime)))^{\Diamond^S_{wh}}} \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S [\Box L] \\
\dfrac{((\Box_{wh} \Box_k np, (Pierre, aime)))^{\Diamond^S_{wh}} \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S}{(((quel, \acute{e}crivain), (Pierre, aime)))^{\Diamond^S_{wh}}} \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S [K1S] \\
\dfrac{(((quel, \acute{e}crivain), (Pierre, aime)))^{\Diamond^S_{wh}} \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S}{((quel, \acute{e}crivain), (Pierre, aime)) \vdash \Box_{wh}^S \Box_{nom}^S \Box_{acc} \Box_{infl}^S} [lex] \\
\dfrac{((quel, \acute{e}crivain), (Pierre, aime)) \vdash \Box_{wh}^S \Box_{nom}^S \Box_{acc} \Box_{infl}^S}{((quel, \acute{e}crivain), (Pierre, aime)) \vdash \Box_{wh}^S \Box_{nom}^S \Box_{acc} \Box_{infl}^S} [\Box R]
\end{array}$$

let us remark that the \Diamond^S_{wh} mode serves as the former functional category CP and that the first conjunct of the product now corresponds to the specifier of the CP.

3.2 Clitics

In some languages (like romance ones) some lexical items like pronouns have strong features whereas other ones that could replace them (say the full *nps* in this case) have weak features. This shows up in morphology: the full *nps* are not case-marked whereas the pronouns are (cf in French: *il*, *le*, *lui* ...). These items will occur in the lexicon as affected by \Box_k^S . Of course, the \Box^S can be cancelled only by a corresponding \Diamond^S . So, in such a case, the sequent is proved only if the base-category *s* contains in its list of modals the corresponding \Diamond^S , thus enforcing displacements. Unformally stated, the sentence $(Pierre, (le, (lui, donne)))$ will have to reduce to $\Box_{nom}^S \Box_{acc}^S \Box_{dat}^S \Box_{infl}^S$, in the following transformational steps:

$$\begin{aligned}
& (Pierre, (le, (lui, donne))) \rightarrow_{[K1S]+[\Box L]} (Pierre, (le, (lui, donne)))^{\circ} \\
& \rightarrow_{[K2]+[K1S]+[\Box L]} (Pierre, (le, (lui, donne)))^{\circ} \\
& \rightarrow_{[K2]+[K1S]+[\Box L]} (Pierre, (le, (lui, donne))^{\circ})^{\circ} \\
& \rightarrow_{[Comm]} (Pierre, (le, (donne, lui))^{\circ})^{\circ} \\
& \rightarrow_{[MA]} (Pierre, ((le, donne)^{\circ}, lui))^{\circ} \\
& \rightarrow_{[Comm]} (Pierre, ((donne, le), lui))^{\circ} \\
& \rightarrow_{[Incl]} (Pierre, ((donne, le), lui))
\end{aligned}$$

The last structure succeeds to reduce if $donne ::= ((np_{nom} \backslash s) / np_{dat}) / np_{acc}$. Let us add that in order to get the arguments in the right places in structures, we may use for *nps* disjunctive types like:

$$\Box_{nom} np_{nom} \vee \Box_{acc} np_{acc} \vee \Box_{dat} np_{dat}$$

or 'existential' types like

$$\Box_k np_k$$

where *k* is a variable on $\{nom, acc, dat\}$, with the arguments of verbs appropriately labelled.

3.3 Adverbials

A topic much addressed by the minimalist literature concerns the difference in placement of the adverbials between French and English. We argue that this is explained by the strong modality \Box_{infl}^S in French, rather than a corresponding weak modality in English. Because of that, the structure $(Pierre, (aime, (tendrement, Marie)))$ according to the sequence of modalities to check is transformed into the final sequence: $(Pierre, ((tendrement, aime), Marie))$ in order to succeed by $[/L]$ and $[\backslash L]$. In English, the structure must be initially $(Peter, ((tenderly, loves), Mary))$ in order to be reduced to s .

Let us assume the following assignments:

$$tendrement ::= V/V \quad \quad \quad aime ::= \Box_{infl}((np_{nom} \backslash s)/np_{acc})$$

where V is a meta-variable for any type of verb.

The proof for the french example is the following:

$$\begin{array}{c}
 \frac{\dots}{\frac{(np_{nom}, ((V/V, (np \backslash s)/np), np_{acc})) \vdash s}{(np_{nom}, ((V/V, (np \backslash s)/np), np_{acc}))^\circ \vdash s} [Incl]} \\
 \frac{(np_{nom}, ((V/V, (np \backslash s)/np), np_{acc}))^\circ \vdash s}{(np_{nom}, (((np \backslash s)/np, V/V)^\circ, np_{acc}))^\circ \vdash s} [Comm] \\
 \frac{(np_{nom}, (((np \backslash s)/np, V/V)^\circ, np_{acc}))^\circ \vdash s}{(np_{nom}, ((np \backslash s)/np, (V/V, np_{acc}))^\circ)^\circ \vdash s} [MA] \\
 \frac{(np_{nom}, ((np \backslash s)/np, (V/V, np_{acc}))^\circ)^\circ \vdash s}{(np_{nom}, ((aime)^\diamond_{infl}^S, (tendrement, np_{acc}))^\circ)^\circ \vdash s} [lex + \Box L] \\
 \frac{(np_{nom}, ((aime, (tendrement, np_{acc}))^\diamond_{infl}^S)^\circ \vdash s}{(np_{nom}, ((aime, (tendrement, np_{acc}))^\diamond_{infl}^S)^\circ \vdash s} [K1S] \\
 \frac{(np_{nom}, ((aime, (tendrement, np_{acc}))^\diamond_{infl}^S)^\circ \vdash s}{((np_{nom}, (aime, (tendrement, np_{acc})))^\diamond_{infl}^S \vdash s} [K2] \\
 \frac{((np_{nom}, (aime, (tendrement, np_{acc})))^\diamond_{infl}^S \vdash s}{(np_{nom}, (aime, (tendrement, np_{acc})))^\circ \vdash \Box_{infl}^S s} [\Box R] \\
 \frac{\dots}{\frac{((np_{nom}, (aime, (tendrement, marie)))^\circ)^\diamond_{acc} \vdash \Box_{infl}^S s}{(np_{nom}, (aime, (tendrement, marie)))^\circ \vdash \Box_{acc} \Box_{infl}^S s} [\Box R] \\
 \frac{(np_{nom}, (aime, (tendrement, marie)))^\circ \vdash \Box_{acc} \Box_{infl}^S s}{((\Box_k np)^\diamond_{nom}^S, (aime, (tendrement, marie)))^\circ \vdash \Box_{acc} \Box_{infl}^S s} [\Box L] \\
 \frac{((\Box_k np)^\diamond_{nom}^S, (aime, (tendrement, marie)))^\circ \vdash \Box_{acc} \Box_{infl}^S s}{((pierre)^\diamond_{nom}^S, (aime, (tendrement, marie)))^\circ \vdash \Box_{acc} \Box_{infl}^S s} [lex] \\
 \frac{((pierre)^\diamond_{nom}^S, (aime, (tendrement, marie)))^\circ \vdash \Box_{acc} \Box_{infl}^S s}{((pierre, (aime, (tendrement, marie)))^\diamond_{nom}^S \vdash \Box_{acc} \Box_{infl}^S s} [K1S] \\
 \frac{((pierre, (aime, (tendrement, marie)))^\diamond_{nom}^S \vdash \Box_{acc} \Box_{infl}^S s}{(pierre, (aime, (tendrement, marie))) \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S s} [\Box R]
 \end{array}$$

We can notice here that the [MA]-rule is used in order to adjoin the adverbial to its verb, and [Comm] in order to put it in the correct function-argument order. The french adverb *tendrement* cannot occur before the verb just because when transferring the strong mode \diamond_{infl}^S to the left, after the use of the [K2]-rule, the mode could only go to the first conjunct which would be in this case the adverb, and the feature *infl*, beared by the verb, could not be cancelled. Of course, there could be an adverbial *s/s*. In this case, only the [MC]-rule would be needed.

4 Curry-Howard Semantics

4.1 Overt Movement

One of the biggest advantages of the type-logical approach is its ability to describe by a very simple and elegant tool how to build the logical form of a sentence. This tool is the Curry-Howard isomorphism. It can be applied here modulo some minor changes in our views concerning deductions. Our first derivation, associated with the interrogative french sentence *Quel écrivain Pierre aime* is perhaps not the best we can do if we want to take semantics into account. It would be convenient to associate the interrogative-*np* (*Quel écrivain*) with a λ -expression like:

$$\lambda u.WHICH(x, writer(x) \wedge u(x))$$

In this case, if the expression *Pierre aime* can be associated with the semantics

$$\lambda y.likes(Pierre, y)$$

we shall be able to obtain, by a mere application of the first function to this argument:

$$\begin{aligned} & [\lambda u.WHICH(x, writer(x) \wedge u(x))] \lambda y.likes(Pierre, y) \\ & \longrightarrow WHICH(x, writer(x) \wedge likes(Pierre, x)) \end{aligned}$$

This requires two things we have not yet used: that a *np* be a functor and that hypothetical reasoning be used in order to give a logical form to the *s* missing an *np*: *Pierre aime*.

The strategy for that is to use a lifted type for the interrogative *np* and to show that the rest of the sentence is of the expected type under the hypothesis that we have an *np*. We know that to prove the sequent: $A / (B \setminus^* A) * \Gamma \vdash A$ amounts to prove: $\Gamma \vdash B \setminus^* A$, which amounts to prove: $B * \Gamma \vdash A$. By starting from the first sequent, we make sure that $A / (B \setminus^* A)$ is applied to Γ , thus providing a functional interpretation of quantifiers and interrogatives, and then, we prove the syntactic correctness of the rest of the sentence by means of the simpler type B. This strategy can be here applied fruitfully, giving a categorial account of *overt move* in MGs: in the reverse option we have adopted, in an overt move, the semantic interpretation of the moved constituent stays in place, when the phonological one (represented by the lower type B) gets back to its original place. From the MG point of view, that means that the whole constituent (phonological features + semantic features) moves up, in order for the semantical interpretation to get its right place.

Let us make therefore the following assignment:

$$\begin{aligned} quel - \acute{e}crivain ::= & \square_{wh} \square_{nom} s / ^\circ (np_{nom} \setminus ^\circ s) \vee \square_{wh} \square_{acc} s / ^\circ (np_{acc} \setminus ^\circ s) \quad or \\ & \square_{wh} \square_k s / ^\circ (np_k \setminus ^\circ s) \end{aligned}$$

and we get the following fragment of derivation:

$$\begin{array}{c}
\dfrac{\dots}{(s/\circ(np_{acc}\backslash^\circ s), (np_{nom}, aime))^\circ \vdash s} /^\circ L \\
\dfrac{\dots}{(\Box_{acc}s/\circ(np_{acc}\backslash^\circ s), (np_{nom}, aime)^\circ)^\circ \vdash \Box_{acc}\Box_{infl}^S s} \Box L \\
\dfrac{(\Box_{acc}s/\circ(np_{acc}\backslash^\circ s), ((\Box_k np_k)^\diamond_S, aime)^\circ)^\circ \vdash \Box_{acc}\Box_{infl}^S s}{(\Box_{acc}s/\circ(np_{acc}\backslash^\circ s), ((Pierre, aime))^\diamond_S)^\circ \vdash \Box_{acc}\Box_{infl}^S s} \Box L \\
\dfrac{(\Box_{acc}s/\circ(np_{acc}\backslash^\circ s), ((Pierre, aime))^\diamond_S)^\circ \vdash \Box_{acc}\Box_{infl}^S s}{((\Box_{acc}s/\circ(np_{acc}\backslash^\circ s), (Pierre, aime))^\circ)^\diamond_S \vdash \Box_{acc}\Box_{infl}^S s} K1S + lex \\
\dfrac{((\Box_{acc}s/\circ(np_{acc}\backslash^\circ s), (Pierre, aime))^\circ)^\diamond_S \vdash \Box_{acc}\Box_{infl}^S s}{(\Box_{acc}s/\circ(np_{acc}\backslash^\circ s), (Pierre, aime))^\circ \vdash \Box_{nom}^S \Box_{acc}\Box_{infl}^S s} [K2] \\
\dfrac{(\Box_{acc}s/\circ(np_{acc}\backslash^\circ s), (Pierre, aime))^\circ \vdash \Box_{nom}^S \Box_{acc}\Box_{infl}^S s}{((\Box_{wh}\Box_{acc}(s/\circ(np_{acc}\backslash^\circ s)))^\diamond_S, (Pierre, aime))^\circ \vdash \Box_{nom}^S \Box_{acc}\Box_{infl}^S s} [\Box R] \\
\dfrac{((\Box_{wh}\Box_{acc}(s/\circ(np_{acc}\backslash^\circ s)))^\diamond_S, (Pierre, aime))^\circ \vdash \Box_{nom}^S \Box_{acc}\Box_{infl}^S s}{((\Box_{wh}\Box_{acc}s/\circ(np_{acc}\backslash^\circ s), (Pierre, aime)))^\diamond_S \vdash \Box_{nom}^S \Box_{acc}\Box_{infl}^S s} [\Box L] \\
\dfrac{((\Box_{wh}\Box_{acc}s/\circ(np_{acc}\backslash^\circ s), (Pierre, aime)))^\diamond_S \vdash \Box_{nom}^S \Box_{acc}\Box_{infl}^S s}{(((quel, \acute{e}crivain), (Pierre, aime)))^\diamond_S \vdash \Box_{nom}^S \Box_{acc}\Box_{infl}^S s} [K1S] \\
\dfrac{(((quel, \acute{e}crivain), (Pierre, aime)))^\diamond_S \vdash \Box_{nom}^S \Box_{acc}\Box_{infl}^S s}{((quel, \acute{e}crivain), (Pierre, aime)) \vdash \Box_{wh}^S \Box_{nom}^S \Box_{acc}\Box_{infl}^S s} [lex] \\
\dfrac{((quel, \acute{e}crivain), (Pierre, aime)) \vdash \Box_{wh}^S \Box_{nom}^S \Box_{acc}\Box_{infl}^S s}{((quel, \acute{e}crivain), (Pierre, aime)) \vdash \Box_{wh}^S \Box_{nom}^S \Box_{acc}\Box_{infl}^S s} [\Box R]
\end{array}$$

The success of the derivation depends now on the proof of the sequent:
 $(np_{nom}, (np_{nom} \backslash s) / np_{acc}) \vdash np_{acc} \backslash s$ which is established by:

$$\begin{array}{c}
\dfrac{\dots}{(np_{nom}, ((np_{nom} \backslash s) / np_{acc}, np_{acc})) \vdash s} [Comm] \\
\dfrac{(np_{nom}, ((np_{nom} \backslash s) / np_{acc}, np_{acc})) \vdash s}{(np_{nom}, (np_{acc}, (np_{nom} \backslash s) / np_{acc}))^\circ \vdash s} [MC] \\
\dfrac{(np_{nom}, (np_{acc}, (np_{nom} \backslash s) / np_{acc}))^\circ \vdash s}{(np_{nom}, (np_{nom} \backslash s) / np_{acc}) \vdash np_{acc} \backslash s} [\circ R]
\end{array}$$

Remark 1: In the first part of the deduction, we could suspect the nominative case goes to the interrogative *np quel \acute{e}crivain*, and that, corollatively, the accusative case goes to the subject *Pierre*, thus resulting in a wrong interpretation of the sentence. In fact, this is not possible: if it was the case, the product *Pierre • aime* would remain a \bullet -product, and when the following modality (which is here the strong inflection) would be transmitted to this product, by [K1S], it would be distributed over the *np* instead of being distributed over the *s* like it can be the case when the product is \circ , therefore the modality \Box_{infl}^S could not be deleted, thus resulting in a failure. The only solution is that \diamond_{nom}^S goes to the *np Pierre* and the \diamond_{acc} to the extracted *np* which is already combined with the rest of the sentence by a \circ (after the cancellation of \Box_{wh}^S).

Remark 2: It is particularly interesting to notice that we get an analysis even for non-peripheral extractions (contrarily to the ordinary Lambek calculus) like for the sentence *quel livre Pierre \acute{e}tudie aujourd'hui?* where we assume *aujourd'hui* has the type $s \backslash s$. The deduction leads us from the bottom sequent:

$$((Quel, livre), (Pierre, (\acute{e}tudie, aujourd'hui))) \vdash \Box_{wh}^S \Box_{nom}^S \Box_{acc} \Box_{infl}^S s$$

to:

$$(s/\circ(np_{acc} \backslash^\circ s), (np_{nom}, ((np_{nom} \backslash s) / np_{acc}, s \backslash s)^\circ)^\circ)^\circ \vdash s$$

and then to:

$$(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, s \setminus s)^\circ)^\circ \vdash np_{acc} \setminus^\circ s$$

for which we have the following deduction:

$$\frac{\begin{array}{c} \dots \\ \hline ((np_{nom}, ((np_{nom} \setminus s)/np_{acc}, np_{acc})), s \setminus s) \vdash s \\ \hline ((np_{nom}, (np_{acc}, (np_{nom} \setminus s)/np_{acc})^\circ), s \setminus s) \vdash s \\ \hline ((np_{acc}, (np_{nom}, (np_{nom} \setminus s)/np_{acc})^\circ)^\circ, s \setminus s) \vdash s \\ \hline (np_{acc}, ((np_{nom}, (np_{nom} \setminus s)/np_{acc})^\circ, s \setminus s))^\circ \vdash s \\ \hline (np_{acc}, (np_{nom}, ((np_{nom} \setminus s)/np_{acc}, s \setminus s)^\circ)^\circ)^\circ \vdash s \\ \hline (np_{nom}, ((np_{nom} \setminus s)/np_{acc}, s \setminus s)^\circ)^\circ \vdash np_{acc} \setminus^\circ s \end{array}}{[Comm]} \quad [MC] \quad [MA] \quad [MA] \quad [\setminus^\circ R]$$

4.2 Covert Movement

The situations concerned by *covert movement* are all those which have a not moving (phonological part of a) constituent that has scope over the entire sentence. The archetype of this situation is provided by *in situ* binding, a phenomenon which has been intensively investigated in the past in the categorial framework, and particularly by M. Moortgat ([7]). We are taking here most part of the solution brought by him. This solution consists in introducing our third product: $*$. Constituents concerned by covert movement are assigned a lifted type made with $/^*$, \setminus^* and a special modality \Diamond which makes communication possible with the $*$ -product. We assume new communication rules, which are symmetrical with respect to those of \circ with regards to \bullet .

Introduction of $$:*

$$\Diamond A \rightarrow A * \mathbf{t} \quad [* \text{I}]$$

Introduction of \Diamond :

$$A \circ \mathbf{t} \rightarrow \Diamond A \quad [\Diamond \text{I}]$$

Communication postulates:

$$\begin{array}{ll} A * B \rightarrow A \circ B & [* \circ] \\ A \bullet (B * C) \rightarrow (A \bullet B) * C & [MA']_1 \\ (A * B) \bullet C \rightarrow A * (B \bullet C) & [MA']_2 \\ A \bullet (B * C) \rightarrow B * (A \bullet C) & [MC'] \end{array}$$

We give to any quantified expression the possibility of having the type

$$\Box_k \Diamond (s / ^* (\Box np_k \setminus ^* s))$$

Let us see the following example, associated with the sentence *Pierre a lu tous les livres*, where we assume *tous les livres* gets this type. A fragment of the deduction (after the removal of \Box_k and the instantiation $k = acc$) is:

$$\begin{array}{c}
\frac{(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, \mathbf{t})) \vdash \Box np_{acc} \setminus^* s \quad s \vdash s}{(s / (* (\Box np_{acc} \setminus^* s), (np_{nom}, ((np_{nom} \setminus s)/np_{acc}, \mathbf{t}))))^* \vdash s} \quad [/*L] \\
\frac{(np_{nom}, (s / (* (\Box np_{acc} \setminus^* s), ((np_{nom} \setminus s)/np_{acc}, \mathbf{t}))))^* \vdash s}{(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, (s / (* (\Box np_{acc} \setminus^* s), \mathbf{t}))))^* \vdash s} \quad [MC'] \\
\frac{(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, (s / (* (\Box np_{acc} \setminus^* s), \mathbf{t}))))^* \vdash s}{(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, \Diamond (s / (* (\Box np_{acc} \setminus^* s)))) \vdash s} \quad [*I] \\
\frac{\dots}{(Pierre, (a_lu, (tous_les_livres))) \vdash \Box_{nom}^S \Box_{acc} \Box_{infl}^S s} \quad [\Box R]
\end{array}$$

That was the first part of the deduction: the quantified expression gets its right position in order to get its scope. In the second part of it, the right rule for \setminus^* is used just before the communication rule $[*, \circ]$ in order to get a deduction similar to overt movements: $\Box np_{acc}$ gets back to its place, and its modalisation allows the cancellation of \mathbf{t} by means of the usual law $\Diamond \Box A \rightarrow A$ and the reverse postulate of $[*I]$, which is $[\Diamond I]$. We get:

$$\begin{array}{c}
\frac{\dots}{(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, np_{acc})) \vdash s} \quad [\Box L] \\
\frac{(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, \Diamond \Box np_{acc})) \vdash s}{(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, (\Box np_{acc}, \mathbf{t})^\circ)) \vdash s} \quad [\Diamond I] \\
\frac{(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, (\Box np_{acc}, \mathbf{t})^\circ)) \vdash s}{(np_{nom}, (\Box np_{acc}, ((np_{nom} \setminus s)/np_{acc}, \mathbf{t}))^\circ) \vdash s} \quad [MC'] \\
\frac{(np_{nom}, (\Box np_{acc}, ((np_{nom} \setminus s)/np_{acc}, \mathbf{t}))^\circ) \vdash s}{(\Box np_{acc}, (np_{nom}, ((np_{nom} \setminus s)/np_{acc}, \mathbf{t})))^\circ \vdash s} \quad [MC'] \\
\frac{(\Box np_{acc}, (np_{nom}, ((np_{nom} \setminus s)/np_{acc}, \mathbf{t})))^\circ \vdash s}{(\Box np_{acc}, (np_{nom}, ((np_{nom} \setminus s)/np_{acc}, \mathbf{t})))^* \vdash s} \quad [* \circ] \\
\frac{(\Box np_{acc}, (np_{nom}, ((np_{nom} \setminus s)/np_{acc}, \mathbf{t})))^* \vdash s}{(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, \mathbf{t})) \vdash \Box np_{acc} \setminus^* s} \quad [\setminus^* R]
\end{array}$$

Let us notice that the case of a subject quantified expression is solved by the use of the $[MA']_2$ -rule. We have the following steps:

$$\begin{aligned}
& (\Diamond (s / (* (\Box np_{nom} \setminus^* s)), (V, np))) \rightarrow (((s / (* (\Box np_{nom} \setminus^* s)), \mathbf{t}))^*, (V, np)) \rightarrow \\
& \quad (s / (* (\Box np_{nom} \setminus^* s), (\mathbf{t}, (V, np))))^* \rightarrow (\Box np_{nom}, (\mathbf{t}, (V, np)))^* \\
& \quad \rightarrow (\Box np_{nom}, (\mathbf{t}, (V, np)))^\circ \rightarrow ((\Box np_{nom}, \mathbf{t})^\circ, (V, np)) \\
& \quad \rightarrow (\Diamond \Box np_{nom}, (V, np)) \rightarrow (np_{nom}, (V, np))
\end{aligned}$$

5 Other Examples

5.1 English Relativization

We give here some aspects of a description of relatives in English. We assume the following assignment:

$$\begin{array}{ll}
met ::= \Box_{infl}((np_{nom} \setminus s)/np_{acc}) & Paul ::= \Box_k np_k \\
that, who ::= (n \setminus n)/\Box_k np_k \setminus^\circ \mathcal{S} & the ::= (\Box_k np_k)/n
\end{array}$$

The nominal phrases *the man that Paul met* and *the man who met Paul* receive the following derivations.

the man that Paul met:

Bottom of the deduction:

$$\frac{\frac{(n, n \setminus n) \vdash n \quad (\Box_l np_l, \Box_{infl}((np_{nom} \setminus s)/np_{acc})) \vdash \Box_k np_k \setminus^\circ \mathcal{S}}{(n, ((n \setminus n)/\Box_k np_k \setminus^\circ \mathcal{S}, (\Box_l np_l, \Box_{infl}((np_{nom} \setminus s)/np_{acc})))) \vdash n} [\backslash L]}{\frac{\Box_c np_c \vdash \Box_c np_c \quad (man, (that, (Paul, met))) \vdash n}{(the, (man, (that, (Paul, met)))) \vdash \Box_c np_c} [L]} [lex]$$

Middle of the deduction:

$$\frac{\frac{\frac{\frac{\frac{\frac{\frac{\dots}{(\Box_k np_k, (np_{nom}, \Box_{infl}((np_{nom} \setminus s)/np_{acc})))^\circ \vdash \Box_{acc} \Box_{infl}^S s} [\Box R]}{(\Box_k np_k, ((\Box_l np_l)^\diamond_{nom}, \Box_{infl}((np_{nom} \setminus s)/np_{acc}))^\circ) \vdash \Box_{acc} \Box_{infl}^S s} [\Box L]}{(\Box_k np_k, ((\Box_l np_l, \Box_{infl}((np_{nom} \setminus s)/np_{acc})))^\diamond_{nom})^\circ \vdash \Box_{acc} \Box_{infl}^S s} [K1]}{((\Box_k np_k, (\Box_l np_l, \Box_{infl}((np_{nom} \setminus s)/np_{acc})))^\diamond_{nom})^\circ \vdash \Box_{acc} \Box_{infl}^S s} [K2]}{((\Box_k np_k, (\Box_l np_l, \Box_{infl}((np_{nom} \setminus s)/np_{acc})))^\diamond_{nom})^\circ \vdash \Box_{acc} \Box_{infl}^S s} [\Box R]}{(\Box_k np_k, (\Box_l np_l, \Box_{infl}((np_{nom} \setminus s)/np_{acc})))^\circ \vdash \Box_{nom} \Box_{acc} \Box_{infl}^S s} [select]}{\frac{(\Box_k np_k, (\Box_l np_l, \Box_{infl}((np_{nom} \setminus s)/np_{acc})))^\circ \vdash \mathcal{S}}{(\Box_l np_l, \Box_{infl}((np_{nom} \setminus s)/np_{acc})) \vdash \Box_k np_k \setminus^\circ \mathcal{S}} [\setminus^\circ R]}$$

It is worthwhile to notice here that the correct analysis is provided. We could expect the mode \diamond_{nom}^S be affected to np_k . But this would result in a failure, because in this case, the mode \diamond_{acc} would necessarily be compelled to go down to the root of the subtree $(\Box_l np_l, V)$, which is a \bullet -node. The accusative modality would be checked, but without opening this \bullet -product, and no structural postulate could be applied in order to get the correct configuration for cancelling the slashes. That \diamond_{nom}^S be affected to np_l is therefore the only solution.

the man who met Paul:

Middle of the deduction:

$$\frac{\frac{\frac{\frac{\frac{\dots}{(np_{nom}, ((np_{nom} \setminus s)/np_{acc}, \Box_l np_l))^\circ \vdash \Box_{acc} s} [\bullet - \circ]}{((\Box_k np_k)^\diamond_{nom}, ((np_{nom} \setminus s)/np_{acc}, \Box_l np_l))^\circ \vdash \Box_{acc} s} [\Box L]}{((\Box_k np_k, ((np_{nom} \setminus s)/np_{acc}, \Box_l np_l)))^\diamond_{nom} \vdash \Box_{acc} s} [K1S]}{(\Box_k np_k, ((np_{nom} \setminus s)/np_{acc}, \Box_l np_l)) \vdash \mathcal{S}} [\Box R]}{\frac{(\Box_k np_k, ((np_{nom} \setminus s)/np_{acc}, \Box_l np_l)) \vdash \mathcal{S}}{((np_{nom} \setminus s)/np_{acc}, \Box_l np_l) \vdash \Box_k np_k \setminus^\circ \mathcal{S}} [\backslash R]}$$

The nominative modality directly goes to $\Box_k np_k$, thus giving the nominative case to the missing element of the relative. There is no difficulty afterwards for attributing the accusative case to an element which is already at its convenient place, and then, cancellation of slashes can be performed in the straightforward manner.

5.2 VSO Languages and Subject-Verb Inversion

It is easy to show that this framework takes SOV-languages into account: this just amounts to have a strong accusative modality. But VSO languages necessitate a more fine-grained analysis of features, splitting the mode \Box_{infl} into two separate modes: one for tense and the other for agreement. The role which was

played in our previous analysis by *infl* will now be played by *agr*, and the feature *tense* will be put ahead in the series of modalities which affects a goal, in such a way that a tensed sentence will reduce to $\Box_{tps}\Box_{nom}^S\Box_{acc}\Box_{infl}^S$. In a VSO language, \Box_{tps} is strong. This can be applied to the case of sentences with inversion like they occur in French (for instance: *je demande quand viendront les beaux jours*(I ask when will come the beautiful days)). These sentences are analyzed by means of a "subsequence" of modes $\Box_{wh}^S\Box_{tps}^S$ which compells the wh-constituent *and* the verbal head to move up. Of course, this is a facultative typing, the other possibility is $\Box_{wh}^S\Box_{tps}$ which gives only *je demande quand les beaux jours viendront*.

Actually, we can assume in this framework several choices of (sub)sequences of modalities. These sequences of modes are predetermined and associated with particular languages. They are such that some modalities (like \Box_{wh}^S) are "stuck" to other ones (for instance \Box_{tps}^S). These restrictions can be formulated apart and they resemble the well known feature co-occurrence restrictions in GPSG-style.

6 Conclusion

This paper proposes a solution to the problem of expressing some thesis of the Minimalist program in the Multimodal Categorical Framework. Very clearly, we can draw a correspondance between the two frames.

Minimalist Program	Multimodal Categorical Framework
categorial features	types
formal features	modalities
categorial features checking	residuation laws for slashes
formal features checking	$\Diamond\Box A \rightarrow A$
moves	tree-restructuring by postulates
overt moves	$\{\bullet-\circ\}$ -communication
covert moves	$\{*- \circ\}$ -communication
logical forms	λ -terms representing proofs

But there are obviously some differences that we propose to view as advantages of MCF. If the rational goal of MP is to dispense with as many devices as we can, we may pretend to be in its spirit if we show how to dispense with empty elements like traces and with empty nodes. In a very rough analyse of a sentence like *Peter loves Mary*, the generative conception would posit a trace of the subject in the first argument position of the VP-shell, resulting in a structure like:

$$Peter_1 t_1 \text{ loves } Mary$$

represented by a tree with an empty node. But this is useless because for the semantic interpretation, we only need the tree structure:

$$(Peter, (loves, Mary))$$

Even if moves change the respective positions of the constituents with regards to their semantic interpretation, traces are devices which can be dispensed with because the λ -term which is built up by the Curry-Howard isomorphism encodes the story of the transformations, like it can be particularly shown in the case of covert moves. More generally, in the Generative Framework, traces have been introduced in order to encode the story of the transformational derivation into the surface structure obtained, but in MCF, building up λ -terms already does that job, thus making traces useless.

A point where we depart from Stabler's MG concerns empty types. We call empty types those types which are associated with no string. For instance, MGs admit so called 'phonetically empty lexical items' which would correspond in MCF to uninhabited types. We think that it is more in the spirit of Categorical Grammar to dispense with those types, and this also realizes some conceptual economy to get rid of them.

Of course a trivial parsing algorithm which would be based on a blind proof search in the research space would be particularly inefficient, but it seems easy to foresee heuristics which could help the research, for instance by early establishing a one-to-one correspondance between the modalities in the goal and those in the antecedent. Future work will be devoted to a proof-net approach of these problems, which could notably improve the search.

References

1. Bouchard, D. *The Semantics of Syntax, a Minimalist Approach to Grammar*. University of Chicago Press, Chicago, 1995.
2. Chomsky, N. *The Minimalist Program*. The MIT Press, 1996.
3. Cornell, T. A type-logical perspective on minimalist derivations. In Kruijff, G. van, and R. Oehrle, editors, *Formal Grammar'97*, Aix-en-Provence, 1997. ESSLLI'97.
4. Heylen, D. Underspecification in subsomption-based type logical grammars. In Lecomte, A., F. Lamarche, and G. Perrier, editors, *Logical Aspects of Computational Linguistics*, LNCS/LNAI. Springer, forthcoming.
5. Kurtonina, N., and M. Moortgat. Structural control. In Blackburn, P., and M. de Rijke, editors, *Specifying Syntactic Structures*, Studies in Logic, Language and Information, pages 75–113. CSLI Publications, Stanford, 1997.
6. Lecomte, A. Pom-nets and minimalism. In Casadio, C., editor, *Proceedings of the IV Roma Workshop: Dynamic Perspectives in Logic and Linguistics*. SILFS Group in Logic and Natural Language, 1998.
7. Moortgat, M. In situ binding: a modal analysis. In Dekker, P., and M. Stokhof, editors, *Proceedings Tenth Amsterdam Colloquium*. ILLC, Amsterdam, 1996.
8. Moortgat, M. Categorical type logics. In Benthem, J. van, and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2, pages 93–178. Elsevier, 1997.
9. Stabler, E. Derivational minimalism. In Retoré, C., editor, *Logical Aspects of Computational Linguistics*, volume 1328 of LNCS/LNAI, pages 68–95. Springer, 1997.

Sequential Construction of Logical Forms

Wilfried Meyer-Viol

Department of Philosophy
King's College, London
meyervio@dcs.kcl.ac.uk

Abstract. In this paper we give a formal description of the parsing model that underlies the treatment of Long Distance Dependencies, Topic and Focus, Ellipsis and Quantification in, amongst others, the papers [2],[6], [4],[3],[5]. In this model, a natural language string consists of a sequence of ‘instructions packages’ to construct some term in a formal representation language, the *logical form* of the string in question. Parsing, then, is the process of executing these packages in a left to right order.

Introduction

In this paper we will give a formal description of the parsing model that underlies the treatment of Long Distance Dependencies, Topic and Focus, Ellipsis and Quantification in, amongst others, the papers [2],[6],[4],[3],[5]. Although the intuition behind the model is quite natural, nevertheless, it seems not to have been explored to any extent. The main idea is to view the natural language string as consisting of a sequence of ‘instructions packages’ to construct some term in a formal representation language, this term being the supposed *interpretation* or *logical form* of the string in question. Parsing, then, is the process of executing these packages in a left to right order. At all intermediate steps in the parse process, one will have a possibly incomplete specification of (some part of) a logical form. Incompleteness may arise in various ways: one may have a completed term which is a subterm of a logical form yet to be finished, one may have some subterm but not know yet how it is to fit within the term currently under construction, one may have an incomplete specification for a subterm with an accompanying constraint on its completion, and so on. All these possibilities are discussed in the papers mentioned above. The purpose of this paper is formalise the concepts of partiality and information growth required.

In our view, parsing can be seen as establishing an association $\langle s, D \rangle$ between a natural language string s and a set of logical formulas D , the possible interpretations of s . In our model, such a set D of logical forms is constructed by working left to right through the sequence of instruction s ,

$$PARSE(s) = (\langle s(1), D_1 \rangle, \dots \langle s(n), D_n \rangle),$$

$$[{}_0[{}_0 \text{ Fo}(\mathbf{John})] [{}_1[{}_1 \text{ Fo}(\lambda x \lambda y \mathbf{read}(x)(y))] [{}_1 [{}_0 \text{ Fo}(x, \mathbf{book}(x))] [{}_1 \text{ Fo}(\lambda P(\mathbf{some}P))]]]]$$

Fig. 1. A Term as a labelled tree

where $s(n)$ is the last element of s , each D_i is a finite set of *partial logical forms*, D_1 is the result of processing $s(1)$ in a starting context D_0 and, provided s is grammatical, D_n is a set of complete logical forms. In our model, the partial logical forms in D_i , will be represented as the points or states of some partially ordered structure where the partial order \leq represents *development* or incremental *growth* of tree structure: for each D_{i-1} in $PARSE(s)$ and partial logical form $\mathcal{T} \in D_{i-1}$ there is a form \mathcal{T}' such that $\mathcal{T} \leq \mathcal{T}'$ and $\mathcal{T}' \in D_i$.

1 Terms as Decorated Trees

In order to handle partiality of logical forms in a flexible way, we represent the elements of each D_i as *decorated finite partial trees*. Logical forms are built up by one or more ways of putting together basic semantic entities. Each of these modes of combination can be associated with a product type-constructor. A *term* in a language appropriate for these type-constructors can be represented as a *finite binary branching tree*, where every binary branching $\langle b_1, b_2 \rangle$ reflects the presence of a subterm $O(b_1, b_2)$ for some operator O . As an example, let **APL** be the operation of *function application* in a typed lambda calculus. The sentence *John read a book*, represented by the formula **read(John, some(x , book(x)))**, can be seen as resulting from the unreduced lambda term

$$\mathbf{APL}(\mathbf{APL}(\lambda x \lambda y \mathbf{read}(y)(x), \mathbf{APL}(\lambda P(\mathbf{some}P), (x, \mathbf{book}(x))), \mathbf{John})$$

by β -reduction. In Figure 1. we have represented this term as a decorated binary tree in the form of a bracketed formula. Here ‘ $[_0]$ ’ means argument- and ‘ $[_1]$ ’ function-daughter, and the predicate ‘*Fo*’ (for *formula*) holds for (sub)terms of our meaning representation language (in contrast to, for instance, the ‘*Ty*’ predicate that will hold for type expressions).

Such decorated tree structures have to be constructed in the course of a parse through an NL string. In order to deal with the partial logical forms arising during a parse we consider *T*-structures,

Definition 1 (*T*-Structures) A *T*-structure is a quintuple of the form $\mathcal{T} = \langle T, \prec_0, \prec_1, \prec_\downarrow, \prec_* \rangle$ where T is a non-empty domain of tree nodes and \prec_i , for $i \in I = \{0, 1, \downarrow, *\}$, is a set of (possibly empty) binary relations on T .

The set *BT* consists of those *T*-structures which are ordered as *binary trees*, where \prec_0 is the argument-daughter and \prec_1 the function-daughter relation, \prec_\downarrow is the immediate dominance relation ($\prec_\downarrow = \prec_0 \cup \prec_1$), and \prec_* is the dominance relation, i.e., the reflexive and transitive closure of \prec_\downarrow .

$$[_0[_0 \text{ Fo}(\mathbf{John})]] \text{ } [_*[_0 \text{ Fo}((x, \mathbf{book}x))], [_1 \text{ Fo}(\lambda P(\mathbf{some}P))]] \text{ } \dots]$$

Fig. 2. A Partial Term as a partial labelled tree

Definition 2 (Partial Trees) A function f is a *Tr-morphism* from T -structure \mathcal{T} in T -structure \mathcal{T}' if it maps T in T' such that for all $n, m \in T$, for all $i \in I$: $n \prec_i m \Rightarrow f(n) \prec_i f(m)$. The set PT , of *partial trees*, consists of all T -structures \mathcal{T} such that there is a *Tr-morphism* mapping \mathcal{T} to an element of BT , i.e., a binary tree.

Notice that in a full-blown binary tree, $n \prec_* m$ implies that there is a sequence of immediate dominance steps relating n to m , but in a *partial* tree this does not need to be the case. The *under-specified* tree relations \prec_\downarrow , and in particular \prec_* , will play an essential role in constructing the logical form while traversing the string in a left to right fashion: we cannot always decide on the spot where a certain subterm has to function in the eventual term. Given the string *A book John read*, we do not yet know what to do with *A book* at the start of the sentence. After having parsed *A book John* a possible partial logical form constructed is shown in Figure 2, which gives a partial tree model with an under-specified tree relation. This under-specified relation *constrains* the completions to those binary trees which have this relation witnessed by an immediate dominance sequence.

In a set-up where partial trees are constructed in stages, we need a pointer to identify the nodes at which action is to take place, that is, our basic units have to be representations of the form $\langle \mathcal{T}, n \rangle$, a partial tree \mathcal{T} together with a *pointer* indicating some node $n \in T$, which we will write as $\mathcal{T}n$.

Definition 3 (Structure of Pointed Partial Trees) Let $PPT = \{\mathcal{T}n \mid \mathcal{T} \in PT, n \in T\}$ be the set of *pointed partial trees*. For $i \in I$ we set $\mathcal{T}n \prec_i \mathcal{T}'n'$ if $\mathcal{T} = \mathcal{T}'$ and $n \prec_i n'$, and $\mathcal{T}n \leq \mathcal{T}'n'$ if there is a *Tr-morphism* $f : \mathcal{T} \mapsto \mathcal{T}'$ such that $f(n) = n'$. The frame of *Pointed Partial Tree structures* can now be defined as

$$\mathcal{PPT} = \langle PPT, \prec_i, \leq \rangle_{i \in \{0, 1, \downarrow, *\}}.$$

Along \leq , a pair $n, m \in T$ such that $n \prec_* m$ may be mapped by *Tr-morphism* f to a pair such that $f(n) \prec_\downarrow f(m)$ and later, by some *Tr-morphism* g , to a *fully specified* relation $g(f(n)) \prec_0 g(f(m))$.

In the general model we also use a relation $\prec_L \subseteq PPT \times PPT$ where $\mathcal{T}n \prec_L \mathcal{T}'n'$ implies that $T \cap T' = \emptyset$ and n' is a top node of \mathcal{T}' . Thus, this relation connects two disjoint trees where an arbitrary node of the first tree is *Linked* to the top node of the second tree. These disjoint, but connected, trees will represent linguistic “islands” in our model. This relation is used in the example of Figure 6.

1.1 The Language DU

On these pointed tree structures we can interpret the Language of Finite Trees, LFT (see [1]), a propositional modal language with the modalities $\langle 0 \rangle \phi$ (“ ϕ holds on the argument daughter”), $\langle 1 \rangle \phi$ (“ ϕ holds on the function daughter”), $\langle \downarrow \rangle \phi$ (“ ϕ holds on some daughter”), $\langle * \rangle \phi$ (“ ϕ holds here or somewhere below”), $\langle L \rangle \phi$ (“ ϕ holds on a linked node”), their converses $\langle i^{-1} \rangle$, and their universal variants $[i]$, $[i^{-1}]$, for $i \in \{0, 1, \downarrow, *\}$. In the tree of Figure 1, for instance the following holds

- $\langle 0 \rangle Fo(\mathbf{John})$ and $\langle 1 \rangle \langle 1 \rangle Fo(\lambda x \lambda y \mathbf{read}(x)(y))$ at the top node 0.
- $\langle \downarrow^{-1} \rangle \langle * \rangle Fo(x, \mathbf{book}x)$ at node 00 decorated by $Fo(\mathbf{John})$.

where the atomic formulas $Fo(\mathbf{John})$ etc., are designed to describe *Declarative Units* decorating binary (linked) tree structures. Declarative units are pairs consisting of a sequence of labels followed by a content formula:

$$\underbrace{\langle l_1, \dots, l_n \rangle}_{\text{labels}} : \underbrace{\Psi}_{\text{Formula}} .$$

We have seen examples of content formulas in the denotations \mathbf{John} , $(x, \mathbf{Book}x)$ and $\lambda x \lambda y \mathbf{read}(x)(y)$. The types e , t and $e \rightarrow t$ from these examples are instances of labels. The descriptions of declarative units determine the *atomic vocabulary*. So, our language has *monadic* predicates La_1, \dots, La_n, Fo , standing for n label dimensions and a formula dimension and individual constants from $D_{La_1}, \dots, D_{La_n}, D_{Fo}$ respectively, denoting values on these dimensions. The *atomic propositions* of the language then have the form $La_i(t)$ or $Fo(t)$ where t is either an element of the appropriate domain D_{La_i} , D_{Fo} , or it is a *meta variable*. A declarative unit $\langle l_1, \dots, l_n \rangle : \Psi$ can then be completely represented by a description, the finite set of atomic propositions satisfied by that unit.

$$\{La_1(l_1), \dots, La_n(l_n), Fo(\Psi)\},$$

And a *partial* declarative unit, an object naturally arising in the course of a parse, is merely a subset of a description of a declarative unit.

The language, DU , we have settled on to describe partial declarative units and their developments towards logical forms includes the tree modalities from LFT , the standard Boolean constants and connectives and existential and universal quantifiers ranging over the set of label and formula *values*.

Definition 4 (The Representation Language DU) A proposition A of the language DU has one of the following shapes:

$$A ::= \top \mid \perp \mid La_1(l_1) \mid \dots \mid La_n(l_n) \mid Fo(\phi) \mid Eq(t_1, t_2) \mid A \wedge A \mid A \vee A \mid \\ \mid A \rightarrow A \mid \exists \mathbf{x} A \mid \forall \mathbf{x} A \mid \langle \# \rangle A \mid [\#] A$$

for $VAR = \{\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{y}, \dots\}$ a denumerable set of individual variables, MV

a denumerable set of *meta variables*, predicate values $l_i \in D_{La_i} \cup VAR \cup MV$, for each i : $1 \leq i \leq n$, ϕ a logical form in $D_{Fo} \cup Var \cup MV$, $t_1, t_2 \in D_{La_i} \cup D_{Fo} \cup VAR \cup MV$ and $\#$ a modality i or i^{-1} for $i \in \{0, 1, \downarrow, *\}$. The quantifier variables are rendered in boldface to distinguish them from the variable bound by quantifiers in the domain D_{Fo} , i.e., variables occurring in the logical forms under construction.

As is standard, this language is interpreted over Pointed Partial Trees by means of *Valuation* functions V .

Definition 5 (Pointed Partial Tree Models) A *Pointed Partial Tree Model* M is a pair $M = \langle PPT, V \rangle$ consisting of a Pointed Partial Tree Structure PPT and a valuation V assigning finite sets of atomic formulas to elements $\mathcal{T}n \in PPT$ and satisfying the following principle

$$\mathcal{T}n \leq \mathcal{T}'n' \Rightarrow V(\mathcal{T}n) \subseteq V(\mathcal{T}'n').$$

This principle guarantees that once an atomic proposition has been established at some node in a partial tree, this proposition will remain to hold there throughout all future developments of that tree. These pointed partial tree will now be used to represent (unreduced) lambda terms as in Figure 1.

Definition 6 (Truth Definition for DU) Given a Pointed Partial Tree Model $M = \langle PPT, V, \rangle$, a set $\mathcal{D} = \bigcup_{i \leq n} D_{La_i} \cup D_{Fo}$ of label and formula values, a set MV of meta variables, $t_1, t_2 \in \mathcal{D} \cup MV$ we say that pointed tree $\mathcal{T}n \in PPT$ of M *satisfies* formula ϕ , with the notation

$$\mathcal{T}n \models_M \phi,$$

if

ϕ is atomic and $\phi \in V(\mathcal{T}n)$

$\phi \neq \perp$

$\phi = \top$

$\phi = Eq(t_1, t_2)$ and $t_1 = t_2$

$\phi = \psi \wedge \chi$ and $\mathcal{T}n \models \psi$ & $\mathcal{T}n \models \chi$

$\phi = \psi \vee \chi$ and $\mathcal{T}n \models \psi$ or $\mathcal{T}n \models \chi$

$\phi = \psi \rightarrow \chi$ and for all $\mathcal{T}'n' : \mathcal{T}n \leq \mathcal{T}'n'$,

if $\mathcal{T}'n' \models_M \psi$ then $\mathcal{T}'n' \models_M \chi$

$\phi = \exists \mathbf{x} \psi$ and there is a $t \in \mathcal{D} : \mathcal{T}n \models_M \psi[t/\mathbf{x}]$

$\phi = \forall \mathbf{x} \psi$ and for all $\mathcal{T}'n' : \mathcal{T}n \leq \mathcal{T}'n'$ and all $t \in \mathcal{D}$
 $\mathcal{T}'n' \models_M \psi[t/\mathbf{x}]$

if $i \in \{0, 1, \downarrow, *\}$ and

$\phi = \langle i \rangle \psi$ and $\exists \mathcal{T}'n' \in PPT : \mathcal{T}n \prec_i \mathcal{T}'n'$ and $\mathcal{T}'n' \models_M \psi$

$\phi = \langle i^{-1} \rangle \psi$ and $\exists \mathcal{T}'n' \in PPT : \mathcal{T}'n' \prec_i \mathcal{T}n$ and $\mathcal{T}'n' \models_M \psi$

$\phi = [i] \psi$ and for all $\mathcal{T}'n' : \mathcal{T}n \leq \mathcal{T}'n'$ and all $\mathcal{T}''n'' \in PPT$
 if $\mathcal{T}'n' \prec_i \mathcal{T}''n''$ then $\mathcal{T}''n'' \models \psi$

$\phi = [i^{-1}] \psi$ and for all $\mathcal{T}'n' : \mathcal{T}n \leq \mathcal{T}'n'$ and all $\mathcal{T}''n'' \in PPT$
 if $\mathcal{T}''n'' \prec_i \mathcal{T}'n'$ then $\mathcal{T}''n'' \models \psi$

So no decorated node w satisfies \perp , \top holds on every node, the atomic formulas hold according to the valuation function. The Boolean connectives and the modal operators have their standard interpretation. As usual, we can introduce negation by the following definition.

$$\neg\phi \equiv_{df} \phi \rightarrow \perp.$$

So, $\neg\phi$ holds at a pointed tree $\mathcal{T}n$ if ϕ does not hold there nor at any $\mathcal{T}'n'$ such that $\mathcal{T}n \leq \mathcal{T}'n'$. The operators and modalities with universal force (\rightarrow , \forall , $[\#]$) quantify not only over (nodes of the) current (partial) decorated trees, but over possible developments of the current structure. For instance, the top node of the current decorated partial tree need not be the *root* node of the eventual tree. Given that we have a falsum \perp (satisfied by no node) and verum \top (satisfied by all nodes) we can ‘close off’ the top node by expressing facts like $\mathcal{T}n \models_M [\downarrow^{-1}] \perp$ meaning “at all mother nodes of $\mathcal{T}n$ the formula \perp holds”, i.e., $\mathcal{T}n$ has no mother node (and will have none). This is an operation that can take place on a tree the moment all words of the NL string have been processed: the node that happens to be the top one at that moment is turned into a root node. At the other end of the tree we can declare bottom nodes to be terminal nodes by annotating them with $[\downarrow]\perp$. This is a task of the lexical entries associated with the words: a word closes off a branch downwards.

By definition we have *persistence* of atomic *DU*-formulas. By the form of the Truth definition, this can be lifted to the whole of *DU*. So, if ϕ is a *DU*-formula, $\mathcal{T}n \models_M \phi$ and $\mathcal{T}n \leq \mathcal{T}'n'$, then $\mathcal{T}'n' \models_M \phi$.

It may be illuminating to view some typical interactions between the tree modalities and connectives, like implication, with universal force. In a model M we can have $\mathcal{T}n \models_M \langle * \rangle \phi$ without there being a sequence $\mathcal{T}n \prec_{\downarrow} \dots \prec_{\downarrow} \mathcal{T}'n'$ such that $\mathcal{T}'n' \models_M \phi$. In a *partial* tree the relation \prec_* between two nodes entails only that the path between them can always be completed to a fully specified one: by definition, a partial tree can always be *Tr*-morphically embedded in a binary tree (where \prec_{\downarrow} is the real immediate dominance relation and \prec_* its real reflexive and transitive closure). On Pointed Partial Tree Models the basic logic of finite trees holds ‘under double negation’. By definition, every Pointed Partial Tree can be extended in M to a full-blown binary tree. Such a tree satisfies all the principles of the Logic of Finite Trees as listed in [1]. Consequently, if ϕ is an *LFT* theorem, i.e., $\models_{LFT} \phi$, then $\mathcal{T}n \models_M \neg\neg\phi$ for every model M .

The meta variables can be distinguished from the proper values by the fact that only for a proper value l_i we have the satisfaction of $\exists \mathbf{x}La_i(\mathbf{x})$. $\exists \mathbf{x}\psi$ holds at a node if $\psi[t/\mathbf{x}]$ holds there for some $t \in \mathcal{D}$ and a meta variable \mathbf{U} is not an element of \mathcal{D} . We want the existential quantifier to be able express that some label or feature predicate has a value. And a meta variable, or the ‘undefined’ symbol just won’t do as a proper value. In the next paragraph we will see some uses of this fact.

Apart from the *LFT* principles we will have to introduce axioms regulating the behaviour of the *Fo* and *Ty* predicates on the trees. For instance, a node may be annotated by at most one type. This requires a principle of the form

$$\forall \mathbf{x} \forall \mathbf{y} ((Ty(\mathbf{x}) \wedge Ty(\mathbf{y})) \rightarrow Eq(\mathbf{x}, \mathbf{y})).$$

Furthermore, the *Fo* and *Ty* values at the daughters of some node have to be related to the values on those predicates at the node itself; for the type predicate, for instance, this is forced by the principle

$$\forall \mathbf{x} \forall \mathbf{y} ((\langle 0 \rangle Ty(\mathbf{x}) \wedge \langle 1 \rangle Ty(\mathbf{x} \rightarrow \mathbf{y})) \rightarrow Ty(\mathbf{y})).$$

2 Goal-Directedness

We still have to add one refinement to the pointed partial trees. Within the domain *PPT* of a partial tree model $M = \langle PPT, V \rangle$, we identify the set *LoFo* consisting of the decorated partial trees that correspond to (unreduced) terms of the typed lambda calculus. Grammatical strings, the words of which project actions mapping one pointed partial tree to a next one, must lead us into this subset of *PPT*. In all elements of *LoFo*, the root node, for instance, will be annotated by a lambda term of type *t*. Thus, in any partial stage, that root node will have a *requirement* that it be annotated by a lambda-term of type *t*, another node that it be annotated by a term of type *e*, and so on. This use of requirements on the development of a tree node has some resemblance to the familiar concept of ‘sub-categorization’, as a node decorated by the labelled formula $Ty(e \rightarrow (e \rightarrow t))$, *Fo(read)* within a tree may have a mother node which is decorated with a requirement $\langle 1 \rangle Ty(e)$ (that is, a requirement for an internal argument for ‘*read*’). However, the concept of requirement is much more general than sub-categorization statements: all nodes are introduced with requirements. Requirements form an essential feature of a partial tree as they determine a set of ‘successful’ extensions, namely those in which all requirements are satisfied. Consequently, our basic data structures are tree structures, the nodes of which consist of (partial) declarative units paired with finite sets of requirements.

To model these requirements we add a *requirement function* *R* to the model assigning a finite number of (arbitrary) *DU*-formulas to elements $\mathcal{T}n$. These formulas represent a finite number of requirements on that node (as opposed to the *facts* at that node assigned by *V*). Figure 3 represents a decorated partial tree resulting from having parsed *John read*. This tree includes a node with a requirement for an object of type *e* rendered by $?Ty(e)$ (this node-plus-requirement is introduced, “sub-categorized for”, by the verb **read**).

Definition 7 (Models with Requirements) A Pointed Partial Tree Model with *requirements* is a triple $\mathcal{M} = \langle M, R \rangle$, where $M = \langle PPT, V \rangle$ is a Pointed Partial Tree Model and *R* is a function assigning finite sets of *DU* formulas to elements of *PPT* and satisfying the following constraint:

$$\mathcal{T}n \leq \mathcal{T}'n' \Rightarrow R(\mathcal{T}n) \subseteq (Th(\mathcal{T}'n') \cup R(\mathcal{T}'n')).$$

Here the theory $Th(\mathcal{T}n)$ of a node $\mathcal{T}n$ is given by $\mathcal{T}n = \{\phi \in DU \mid \mathcal{T}n \models_M \phi\}$. Notice that, unlike the valuation function, the requirement function is not

restricted to atomic propositions; we are free to require any (finite number of) DU-formula(s) at some tree node.

Along the growth relation \leq requirements may disappear, but only by becoming facts. For instance, we have

$$[{}_a \text{ ?}Ty(e)] \leq [{}_a \text{ } Fo(\phi), Ty(e), \text{ ?}Ty(e)] \leq [{}_a \text{ } Fo(\phi), Ty(e)].$$

But also,

$$[{}_a \text{ ?}\exists \mathbf{x} Fo(\mathbf{x}), \text{ ?}Ty(e)] \leq [{}_a \text{ } Fo(John), Ty(e)].$$

The successful developments, that is, the developments in which all requirements are satisfied, of a pointed partial tree $\mathcal{T}n$ we collect in the set $LoFo(\mathcal{T}n)$ of (supposed) Logical Forms into which $\mathcal{T}n$ can develop.

$$LoFo(\mathcal{T}n) = \{\mathcal{T}'n' \in PPT \mid \mathcal{T}n \leq \mathcal{T}'n' : R(\mathcal{T}'n') = \emptyset\}.$$

In fact, the set $LoFo$ represents an ‘internal’ version of the set of logical forms we are after. The object is now to make $LoFo$ and the set of real logical forms coincide. Below we will discuss this more extensively.

Having introduced the concept of a requirement over Pointed Partial Tree Models we will exploit it by introducing some constants to the language DU which address the status of the requirements.

Definition 8 (Truth Definition for Requirements) Given a Pointed Partial Tree Model with *Requirements*, $\mathcal{M} = \langle M, R \rangle$, we say that pointed tree $\mathcal{T}n$ of \mathcal{M} *satisfies* formula ϕ , with the notation

$$\mathcal{T}n \models_{\mathcal{M}} \phi,$$

if

$$\begin{aligned} &\phi \in DU \text{ and } \mathcal{T}n \models_M \phi, \\ &\phi = \text{?}\psi \text{ and } \psi \in R(\mathcal{T}n) \\ &\phi = \text{?}\emptyset \text{ and } R(\mathcal{T}n) = \emptyset \\ &\phi = \mathbf{F}\psi \text{ and } \exists \mathcal{T}'n' \in LoFo(\mathcal{T}n) : \mathcal{T}'n' \models_{\mathcal{M}} \psi \\ &\phi = \mathbf{G}\psi \text{ and } \forall \mathcal{T}'n' \in LoFo(\mathcal{T}n) : \mathcal{T}'n' \models_{\mathcal{M}} \psi \end{aligned}$$

A requirement $\text{?}\phi$ holds if the formula ϕ occurs on the the requirement list of that node, and $\text{?}\emptyset$ is a constant which holds at a node if it has an empty requirement list. Proposition $\mathbf{F}\psi$ holds at node $\mathcal{T}n$ if there is at least one development of $\mathcal{T}n$ to (a node in) a logical form where ψ holds, and $\mathbf{G}\psi$ holds at node $\mathcal{T}n$ if ψ holds at all developments of $\mathcal{T}n$ to logical forms.

A tree node $\mathcal{T}n$ has requirements that can be satisfied iff $\mathbf{F}\top$ holds at $\mathcal{T}n$. This allows us to define a conditional: $\phi \rightarrow_{gr} \psi \equiv_d (\mathbf{F}\top \wedge \phi) \rightarrow \psi$. This conditional expresses invariances shared *only* by successful developments. The principles of binary trees we express using \rightarrow (for instance, $\langle 1 \rangle \top \rightarrow \langle 0 \rangle \top$, “if there is a function daughter then there is an argument daughter”, after all, all PPT ’s can

$$[_0 [_0 \text{Fo}(\mathbf{John})], [_1 [_0 ? \mathbf{Ty}(\mathbf{e})], [_1 \text{Fo}(\lambda x \lambda y \mathbf{read}(\mathbf{x})(\mathbf{y}))]]]]$$

Fig. 3. Partial tree with requirements

map to binary trees, by definition). But, for instance, β -reduction need only function in trees with successors in *LoFo*. Here we use the axiom schema

$$(\langle 0 \rangle \lambda x \phi \wedge \langle 1 \rangle \psi) \rightarrow_{gr} \phi[\psi/x].$$

2.1 The Uses of Requirements

Consider all terms of the typed lambda calculus we use to formulate our logical forms, the representations of the formal meanings of the NL strings under consideration. The elements of this set are our target structures the NL strings have to construct. We represent these terms as trees by their applicative structure (that is, a subterm $(\lambda x \phi, \psi)$ is represented as a binary branching point with $\lambda x \phi$ annotating the function - and ψ the argument daughter). At the nodes of these representations we hang empty sets. These empty sets represent empty sets of requirements. This we now turn into the set *LoFo* of some Pointed Partial Tree Model \mathcal{M} , by adding all *partial* versions that can be extended to elements of the *LoFo*. Now, in order to *guarantee* that the set *LoFo*, defined in terms of fulfilled requirements, and the set of representations of terms in our typed lambda calculus coincide, every abstraction of such a term to a partial object has to be compensated by the introduction of a requirement. Starting, for instance, from the term $\mathbf{read}(\mathbf{John}, (\mathbf{a}, x, \mathbf{book}x))$ we can create a partial term by abstraction over **John**. This abstraction is not a term of our typed lambda calculus, so it should not belong to *LoFo*. By definition, this means that there must be some unfulfilled requirement associated with it. But this does not have to be a requirement for exactly ‘**John**’. It may (and, in fact, will) be merely a requirement for type *e*. So here is where the invariances of the process come in.

Given a specific feature of the logical forms we are interested in, the first question is now always: can we devise a system of requirement introduction such that the fulfillment of all requirements annotating a given tree corresponds exactly to a completing this tree to a term of our typed lambda calculus with the desired feature?. We will give three examples of our use of this principle.

Binary trees: as a first example, we will consider a requirement strategy to the effect that the set *LoFo* coincides with the set of a binary tree structures. That is, no under-specified tree relations of the form \prec_* or \prec_\downarrow are left in *LoFo* that are not the reflexive transitive closure of the immediate dominance relation or the union of argument and function daughter relation, respectively. The idea is to introduce a *Tree node* label, a monadic predicate ‘*Tn*’ with values in $D_{Tn} =$

$\{a \cdot x, x \mid a \in A, x \in \{0, 1, L\}^*\}$. That is, a value of this predicate is a finite sequence of elements of $\{0, 1, L\}$ possibly preceded by a constant from a set A . When trees are constructed in the parsing process, in general it is not known whether a description that starts off as a top node will remain so (and thus be the *root* node of the eventual tree). This is why we introduce a new node invariably with ‘address’ $a \in A$, satisfying a formula $Tn(a)$, where a is a constant not yet occurring in the construction. In interaction with the tree modalities various constellations are expressible. So, given the formula $Tn(a)$, expressing the location of a node in the tree under construction, we can fix

$$\begin{aligned} Tn(a0) &\leftrightarrow \langle 0^{-1} \rangle Tn(a), \\ Tn(a1) &\leftrightarrow \langle 1^{-1} \rangle Tn(a),^1 \end{aligned}$$

and we can fix the root node of a tree as follows,

$$Tn(0) \leftrightarrow [\downarrow^{-1}] \perp.$$

(Significantly, we do *not* ‘internalize’ the under-specified modalities $\langle * \rangle$ and $\langle \downarrow \rangle$ as values of the Tn predicate.) The Tree node formula $Tn(0)$ holds at a node if it is a top node and remains so throughout all developments. (Note the use of the ‘falsum’ - “At every node above the current one \perp holds.” As \perp is satisfied by no node at all (Definition 6) this means that there are no nodes above the current one.)

Now when we introduce a tree node with an under-specified relation to the source node, as we do in Figure 5, we add the requirement for $\exists \mathbf{x} Tn(\mathbf{x})$ to the node with under-specified address:

$$[a \ ?Ty(t)], a \quad \Rightarrow \quad [a \ ?Ty(t), [* \ ?Ty(e), ?\exists \mathbf{x} Tn(\mathbf{x})]], a*$$

The point is that this requirement can only be satisfied when the node it decorates is merged with one that has a fully specified relation to the top node of the tree: only in that case will there be a $t \in D_{Tn}$ such that $Tn(t)$ holds at the node. So, if, starting from the Axiom, we introduce tree nodes with under-specified tree *only* if they are accompanied by requirements for values on the Tn predicate, then in all elements of $LoFo$ that we can reach, all underspecification with respect to tree relations will have been resolved.

Pronouns: an analogous mechanism allows us to introduce meta variables or *placeholder values* with the requirement that they be substituted in order for the term to be complete.

$$[a \ ?Ty(e)], a \quad \Rightarrow \quad [a \ Fo(\mathbf{U}), Ty(e), ?\exists \mathbf{x} Fo(\mathbf{x})], a$$

An object of type e has been supplied, but a new, weaker, requirement has taken its place. Only a development that supplies a concrete value for the meta variable \mathbf{U} in $Fo(\mathbf{U})$ can end up in $LoFo$.

¹ We will also need $Tn(aL) \leftrightarrow \langle L^{-1} \rangle Tn(a)$ when we consider Linked Trees.

Resumptive pronouns: On the other hand, in the example of Figure 6 we will introduce the tree node

$$[*?Fo(\mathbf{a}, x, \mathbf{book}x))]$$

which contains a requirement that can only be fulfilled by a concrete value for the Fo predicate. Such a value, a *copy* of a value already introduced, can, in our model, not derive from the sentence string *except* by means of a pronoun (as a ‘copy instruction’). In parsing the string “*A book, John read it*”, the pronominal *it* can copy any accessible value: this is the contribution of the pronoun. But only if it copies the value from the head “*A book*” will the requirement for the exact formula value that has been introduced be fulfilled: this is the contribution of the goal-directed framework. A pronoun in such a situation is a resumptive one in our model.

3 The Parsing Process

The object of the parsing process is to construct a binary tree structure the top node of which is decorated by a formula of type t while using all information in an NL string. The minimal element in the model \mathcal{M} , the starting point of every parse² is the *Axiom*

Axiom:
$$[{}_a \ ?Ty(t)], a.$$

The Axiom is the Pointed Partial Tree Model consisting of a single node (thus the location of the pointer), the putative top node, with empty valuation function and requirement for an object of type t .

In the course of a parse starting from the Axiom a tree is created. A parse ends essentially after the last word of an NL string has been processed. A *successful* parse of a *grammatical* NL string must result in a logical form, that is, a Pointed Binary Tree in the set $LoFo$ of \mathcal{M} . So, it must end with a tree where all nodes are associated with an empty list of requirements. Moreover, if a node $\mathcal{T}n$ is a *terminal* node in \mathcal{T} , then $\mathcal{T}n \models_{\mathcal{M}} [\downarrow] \perp$. Thus we have a tree that can no longer be extended with descriptions of new nodes. So, a Goal has the form

Goal
$$[{}_a, \dots, Ty(t)], [{}_a 0 \dots], [{}_a 1 \dots] \dots, a \in LoFo.$$

This is a tree all nodes of which have empty requirement lists. If the procedure that leads from the Axiom to an element of $LoFo$ is *sound*, then the final tree represents — is *isomorphic* to — an unreduced term in the language of our logical forms. When an NL string is finished, the words have performed their task, then, if it is grammatical, a binary tree structure has been constructed the terminal nodes of which are decorated and have no unsatisfied requirements. Now, this (representation of an) unreduced lambda term may be normalized in ways depending on a variety of labels collected on the tree during the parse.

² of an NL string in isolation, i.e., without a context.

By parsing a grammatical NL string the Axiom is connected to a Goal by a finite sequence of Pointed Partial Trees. Each tree \mathcal{T}_{i+1} in this sequence is a development of the previous tree \mathcal{T}_i , i.e., $\mathcal{T}_i \leq \mathcal{T}_{i+1}$. Figure 4 shows the structures arising in the course of parsing “*John read a book*”. Notice that each new structure is a development under the \leq relation. The progress from Axiom to

$[_a \text{ ?}Ty(t)], a$	<i>Axiom</i>
$\leq [_a [_0 \text{ ?}Ty(e)], [_1 \text{ ?}Ty(e \rightarrow t)]], a0$	\downarrow
$\leq [_a [_0, Fo(\mathbf{John}), Ty(e)], [_1]], a1$	John
$\leq [_a [_0], [_1 [_1 Fo(\lambda x \lambda y \mathbf{read}(x)(y)), Ty(e \rightarrow (e \rightarrow t))], [_0 \text{ ?}Ty(e)]]], a10$	read
$\leq [_a [_0], [_1 [_1], [_0 Fo(\mathbf{a}, x, \mathbf{book}x), Ty(e)]]], a10$	a book
$\leq [_a [_0 [_1 Fo(\lambda y \mathbf{read}(\mathbf{a}, x, \mathbf{book}x)(y), Ty(e \rightarrow t), [_1], [_0]]], a1$	\downarrow
$\leq [_a \mathbf{read}(\mathbf{a}, x, \mathbf{book}x)(\mathbf{John}), Ty(t), [_0], [_1 [_1] [_0]]], a$	$\in LoFo$

Fig. 4. Pointed Partial Tree Models arising in the course of parsing “*John read a book*.” At every transition only the new aspects are highlighted.

Goal is non-deterministic: at every state of the parse the word currently under consideration can generally be assigned more than one structural role in the tree. That is, the path from Axiom to Goal is one through a space of *alternative parse courses*. For instance, an NP heading an NL string may end up as subject, *John read a book*, it may be a fronted object, *A book John read*, or it may be a topic constituent, *(As for) a book John read it*. The last two possibilities are worked out in Figures 5 and 6 respectively.

Notice that parse of “*John read a book*” and “*A book John read*” end with the same logical form, but this form is reached starting from the Axiom along different routes. Thus the Axiom must be able to access a (finite!) number of alternative partial tree descriptions to accommodate the first NP of the sentence.

$[_a \text{ ?}Ty(t)], a$	<i>Axiom</i>
$\leq [_a \text{ ?}Ty(t), [_* \text{ ?}Ty(e)]], a*$	\downarrow
$\leq [_a \text{ ?}Ty(t), [_* (\mathbf{a}, x, \mathbf{book}x), Ty(e)]], a$	a book
$\leq [_a [_0 \text{ ?}Ty(e)], [_1], [_*]], a0$	\downarrow
$\leq [_a [_0, Fo(\mathbf{John}), Ty(e)], [_1 \text{ ?}Ty(e \rightarrow t)], [_*]], a1$	John
$\leq [_a [_0], [_1 [_1 Fo(\lambda x \lambda y \mathbf{read}(x)(y)), Ty(e \rightarrow (e \rightarrow t))], [_0 \text{ ?}Ty(e)]][_*]], a10$	read
$\leq [_a [_0], [_1 [_1], [_0 Fo(\mathbf{a}, x, \mathbf{book}x), Ty(e)]]], a10$	$* := 10$
$\leq [_a [_0 [_1 Fo(\lambda y \mathbf{read}(\mathbf{a}, x, \mathbf{book}x)(y), Ty(e \rightarrow t), [_1], [_0]]], a1$	\downarrow
$\leq [_a \mathbf{read}(\mathbf{a}, x, \mathbf{book}x)(\mathbf{John}), Ty(t), [_0], [_1 [_1] [_0]]], a$	$\in LoFo$

Fig. 5. Pointed Partial Tree Models arising in the course of parsing “*A book John read*.” At every transition only the new aspects are highlighted.

The presence of alternative, and mutually exclusive, continuations is of course not restricted to the start of an NL string.

The move from Axiom to Goal in a parse of an NL string s is driven by **actions** which are either projected by the words supplied by s or by the computational ‘background’ mechanism.

3.1 Actions

An action α is of the form

$$\alpha \subseteq PPT \times PPT,$$

that is, actions are *relations* over PPT. An action α may map a Pointed Partial Tree to a number of other such trees. The actions we will introduce are fitted to the Pointed Partial Tree Models in that they are *incremental* in the following sense.

Definition 9 (Incremental Actions) An action α is *incremental* if for every $\mathcal{T}n \in PPT$, if $\mathcal{T}'n' = \alpha(\mathcal{T}n)$, then either $\mathcal{T}n \leq \mathcal{T}'n'$ or $\mathcal{T} = \mathcal{T}'$.

So an incremental action is either some *construction* or a *pointer movement*. The actions are based on a set of *basic* or *atomic* actions. The basic actions consist of: creation of new nodes relative to old ones; decoration of nodes, moving to nodes, and substitution at nodes. These action may be combined to give complex ones by, essentially, the PDL operations.

Definition 10 (Actions) For \mathcal{M} a Pointed Partial Tree Model with Requirements, where $\#$ is i or i^{-1} for $i \in \{0, 1, \cdot, \downarrow, *, L\}$, ϕ is either an atomic *DU* formula or of the form $? \psi$ for arbitrary $\psi \in DU$, and $\mathbf{U} \in MV$, the set *ACT* of actions contains the following elements:

Basic Actions

1. $1 = \{\langle \mathcal{T}n, \mathcal{T}n \rangle \mid \mathcal{T}n \in \mathcal{M}\}$,
 $AB = \emptyset$.
 These represent the halting action and the Abort action respectively.
2. **make**($\#$) : $\mathcal{M} \mapsto \mathcal{M}$ This action creates a node $\mathcal{T}'n'$ such $\mathcal{T}' = \mathcal{T} \cup \{n'\}$ and that $\mathcal{T}'n \prec_{\#} \mathcal{T}'n'$.
3. **go**($\#$) : $\mathcal{M} \mapsto \mathcal{M}$ Here **go**($\mathcal{T}n$) = ($\mathcal{T}'n'$) implies $\mathcal{T}n \prec_{\#} \mathcal{T}'n'$.
4. **put**(ϕ) : $\mathcal{M} \mapsto \mathcal{M}$ Here, **put**(ϕ)($\mathcal{T}n$) = $\mathcal{T}'n'$ implies that $\mathcal{T} = \mathcal{T}'$ except that $V(\mathcal{T}'n') = V(\mathcal{T}n) \cup \{\phi\}$ if ϕ is atomic and $R(\mathcal{T}'n') = R(\mathcal{T}n) \cup \{\psi\}$ if $\phi = ?\psi$.
5. **subst**(ϕ, \mathbf{U}) : $\mathcal{M} \mapsto \mathcal{M}$ Here **subst**(ϕ, \mathbf{U})($\mathcal{T}n$) = ($\mathcal{T}'n'$) implies that $\mathcal{T} = \mathcal{T}'$ except that $V(\mathcal{T}'n') = V(\mathcal{T}n)[\phi/\mathbf{U}]$.

Complex Actions We can put actions together by executing one after the other (sequential composition ‘;’) and doing that any finite number of times (finite iteration ‘*’), or by indeterministically choosing between them (choice ‘+’).

$$\begin{array}{ll}
[a \ ?Ty(t)], a & \text{Axiom} \\
\leq [a \ ?Ty(t)], [a_L \ ?Ty(e)], aL & \downarrow \\
\leq [a \ ?Ty(t), [* \ ?Fo(\mathbf{a}, x, \mathbf{book}(x)), []], [a_L \ (\mathbf{a}, x\mathbf{book}x), Ty(e)], a & \mathbf{a} \ \mathbf{book} \\
\leq [a \ [0 \ ?Ty(e)], [1 \ ?Ty(e \rightarrow t)], [* \] [a_L \], a0 & \downarrow \\
\leq [a \ [0 \ , Fo(\mathbf{John}), Ty(e)], [1 \], [* \], [a_L \], a1 & \mathbf{John} \\
\leq [a \ [0 \], [1 \ [1 \ Fo(\lambda x \lambda y \mathbf{read}(x)(y)), Ty(e \rightarrow (e \rightarrow t))], [0 \ ?Ty(e)] [* \]], & \\
& [a_L \], a10 \ \mathbf{read} \\
\leq [a \ [0 \], [1 \ [1 \ Fo(\lambda x \lambda y \mathbf{read}(x)(y)), Ty(e \rightarrow (e \rightarrow t))], & \\
& [0 \ Fo(\mathbf{a}, x, \mathbf{book}x)]] [* \]], [a_L \], a10 \ \mathbf{it} \\
\leq [a \ [0 \], [1 \ [1 \ Fo(\lambda x \lambda y \mathbf{read}(x)(y)), Ty(e \rightarrow (e \rightarrow t))], & \\
& [0 \ Fo(\mathbf{a}, x, \mathbf{book}x)]] [a_L \], a10 \ * := 10 \\
\leq [a \ [0 \ [1 \ Fo(\lambda y \mathbf{read})(\mathbf{a}, x, \mathbf{book}x)(y), Ty(e \rightarrow t)], [1 \], [0 \]], [a_L \] a1 & \downarrow \\
\leq [a \ \mathbf{read}(\mathbf{a}, x\mathbf{book}x)(\mathbf{John}), Ty(t), [0 \], [1 \ [1 \] [0 \]], [a_L \], a & \in LoFo
\end{array}$$

Fig. 6. Pointed Partial Tree Models arising in the course of parsing “A book, John read it.” At every transition only the new aspects are highlighted.

6. if α, α' are actions, then so are $\alpha; \alpha', \alpha + \alpha', \alpha^*$.

Finally, we can put actions together in a conditional IF THEN ELSE statement.

7. If Σ is a set of formulas all variables of which occur in $\bar{\mathbf{x}}$ and α, α' are actions then $\langle \Sigma(\bar{\mathbf{x}}), \alpha, \alpha' \rangle$ is a *conditional* action with the definition:

$$\begin{aligned}
\langle \Sigma(\bar{\mathbf{x}}), \alpha, \alpha' \rangle = & \\
& \{ \langle \mathcal{T}n, \mathcal{T}'n' \rangle \in \alpha[\bar{t}/\bar{\mathbf{x}}] \mid \bar{t} \in (\mathcal{D} \cup MV)^*, \mathcal{T} \models_{\mathcal{M}} \Sigma[\bar{t}/\bar{\mathbf{x}}] \} \cup \\
& \{ \langle \mathcal{T}n, \mathcal{T}'n' \rangle \in \alpha' \mid \neg \exists \bar{t} \in (\mathcal{D} \cup MV)^*, \mathcal{T}n \models_{\mathcal{M}} \Sigma[\bar{t}/\bar{\mathbf{x}}] \}.
\end{aligned}$$

That is, if Σ holds at $\mathcal{T}n$ for some substitution of t for variable \mathbf{x} , then action α is executed where in the body of this action the variable \mathbf{x} is also replaced by t . If Σ does not hold at $\mathcal{T}n$ for any substitution for \mathbf{x} , then action α' is executed.³ For instance we can define actions which transport features from one node (e.g. a ‘head’) to another.

- $\langle \{Fo(\mathbf{x})\}, \mathbf{go}(\langle 0 \rangle); \mathbf{put}(Fo(\mathbf{x})), AB \rangle$ maps the value of the Fo feature at the current node, if there is such a value, to the Fo feature at the left daughter, otherwise it aborts.

We may also define an action that enables pointer movement to the closest clausal node (eg for tense suffixes):

- $\mathbf{gofirst}(\ ?Ty(t)) = \langle \{ \ ?Ty(t) \}, 1, \mathbf{go}(\langle \downarrow^{-1} \rangle)^*; \langle \{ \ ?Ty(t) \}, AB, 1 \rangle$.

The action $\mathbf{gofirst}(X)$ action moves the pointer upwards to the first higher node with annotation X , eg. the requirement $Ty(t)$, and then stops.

³ The PDL tests do not suffice over this propositional logic because we want the ability to formulate IF THEN ELSE statements and only in the context of excluded middle can these be defined in terms of the PDL test.

We have two ‘pure’ classes of actions. Complex *constructions* (that is, compositions of actions without the *go* action) represent complex *tree growth*. Complex *movements* on the other hand (the family of $\text{go}(\langle \# \rangle)$ instructions closed under the given operations) represent *pointer strategies*. In general, for instance in case of actions projected by words, the actions are a mixture of these classes.

Along a path in the space of alternative parse courses, the transitions leading from one partial tree to the next are of two fundamentally different kinds: they are either *Computational Transitions* or *Lexical Transitions*.

3.2 Computational Transitions

Computational transitions develop the information contained in the current tree, they *bring out* information, but they do not add information. A typical example of such a transition is the addition of the formula $\langle 0 \rangle Ty(e)$ to a node (description) $\mathcal{T}n$ in the case there is a node $\mathcal{T}n'$ in the same tree such that $\mathcal{T}n \prec_0 \mathcal{T}n'$ and the formula $Ty(e)$ is an element of $V(\mathcal{T}n')$: a conclusion is ‘computed’ from the information contained in the tree. Figure 7 illustrates this rule.

$$[{}_a [{}_1 Fo(\lambda x \phi)], [{}_0 Fo(\psi)]], a1 \leq [{}_a \langle 1 \rangle Fo(\lambda x \phi), [{}_1 Fo(\lambda x \phi)], [{}_0 Fo(\psi)]], a$$

Fig. 7. Transfer of information up the tree.

This transition sets the stage for an application of $\langle 0 \rangle Fo(\lambda x \phi) \wedge \langle 1 \rangle Fo(\psi) \rightarrow Fo(\phi[\psi/x])$.

Another computational rule is used to *merge* tree nodes. Figure 8 illustrates a situation arising in *Wh*-questions like “*what did John read?*”.

$$[{}_a [{}_1], [{}_0 Tn(t), ?Ty(e)], [*_Fo(Wh), Ty(e), ?\exists \mathbf{x} Tn(\mathbf{x})]], a0 \leq [{}_a [{}_1], [{}_0 Tn(t), Fo(Wh), Ty(e)]], a0$$

Fig. 8. Merging of nodes $a0$ and $a*$.

The *Wh*-element in a question has content of type e but no location in the term, a ‘free-floating’ decoration; the gap lacks type e decoration but has a fixed position in the term. Merging fulfills requirements for position and content of both nodes at once.

A final example of a computational rule is the introduction of a *linked* structure adjoined to an argument of type e .

Here, having processed the word ‘*John*’, the top node of a new (partial) tree is adjoined which requires at some as yet under-specified location the head $Fo(\mathbf{John})$

$$\begin{aligned}
& [a \dots [{}_b Fo(\mathbf{John}), Ty(e)],], a - b \\
& \leq [a \dots [{}_b Fo(\mathbf{John}), Ty(e)],], [{}_{a-bL} ?Ty(t), [{}_* ?Fo(\mathbf{John})]], a - bL
\end{aligned}$$

Fig. 9. Adjoining of a linked tree.

as an argument. This rule sets the stage for a sentence like “*John, who read a book, ...*”. Here, the complementiser ‘*who*’, as any anaphoric object, copies the head into the relative clause i projection. The main clause and the relative clause *have to* share some argument (the ‘head’), this is essential to the Link Construction (see [4]).

3.3 Lexical Transitions

Lexical Transitions, on the other hand, map one tree to a next one adding information in the process. These transitions are projected by the Natural Language *words*. They are defined as conditional actions of the IF THEN ELSE variety. A lexical transitions test IF the some finite set of formulas (the *condition*), holds at the node where the pointer is located — this may include modal statements about decorations located at nodes related to the pointed one and it may also involve requirements. If the condition holds there, THEN a (sequence of) action(s) is undertaken resulting in a new tree description, ELSE (i.e., the condition does not hold) a second action is undertaken, usually an ‘abort’ action.

The actions projected by a lexical item may carry out a whole range of different constructions. Some items project only an annotation (proper names), or only a requirement (case particles), or an annotation plus requirement (tense specifications), or, even, only a pointer movement(expletives).

Definition 11 (Lexical Actions) A *Lexical Action* w is an action of the form

$$w = \langle \Sigma(\bar{x}), \alpha, \alpha' \rangle$$

where Σ is a finite set of *DU*-formulas, \bar{x} is a sequence containing all (free) variables occurring in Σ , and α, α' are complex actions as given by Definition 10.

For example, we have *John*, and *read* project the following actions

```

John IF    {?Ty(e)}
  THEN put(Fo(John), Ty(e), [↓]⊥)
  ELSE AB

read IF    {?Ty(e → t)}
  THEN make(⟨1⟩); go(⟨1⟩)
    put(Fo(read), Ty(e → (e → t)), [↓]⊥;
    go(⟨1-1⟩); put(?⟨0⟩(Ty(e)))
  ELSE AB

```

In case of the proper name *John* the actions are released by the presence of the requirement for an object of type e . These actions then consist in annotating the node with the content formula **John** the type label e , and closing it off as a terminal node. In the case of the transitive verb ‘*read*’, the presence at the pointed node of a requirement for $Ty(e \rightarrow t)$ releases the actions: creation of a function daughter node, annotation of that node with a type and a formula and the imposition on the mother node of a requirement for an argument daughter of type e . Actions annotating the node with facts are the analogue of lexical substitution processes in other frameworks (though it should be remembered that it is not lexical items which are inserted in the tree in this framework, but their logical correlate). Actions decorating a node with requirements are analogous to sub-categorization statements. However all nodes are introduced with requirements, so unlike standard sub-categorization statements which are imposed on sisters to terminal nodes as a requirement on lexical insertion in a tree, in this framework, the use of requirements is much more widespread.

4 Natural Languages

Having defined our representations of the logical forms, we now turn to the language strings which create these forms. A natural language \mathcal{L} is determined by a pair of \mathcal{L} -words, $W_{\mathcal{L}}$, and a \mathcal{L} -alternative-function $ALT_{\mathcal{L}}$,

$$\mathcal{L} = \langle W_{\mathcal{L}}, ALT_{\mathcal{L}} \rangle.$$

— The set $W_{\mathcal{L}}$ of \mathcal{L} -words consists of a set of IF $\Sigma(\bar{x})$ THEN α ELSE α' statements, as we have described them. Traditionally the words of a language relate to syntactic and semantic construction of the clause merely by projecting the decorations of the terminal nodes of some (binary) tree. In our model, on the other hand, words are modelled as *incremental actions* on \mathcal{PPT} . A word may add annotations and requirements at various nodes, it may satisfy requirements, add tree structure, unify nodes and draw conclusions from the annotations across various nodes.

— The function $ALT_{\mathcal{L}}$ of assigns to a pointed partial tree a set of *alternative* developments, alternative structural analyses which are typical for the language \mathcal{L} . No *successful* development of $\mathcal{T}n$ is gained or lost by considering $ALT_{\mathcal{L}}(\mathcal{T}n)$, that is the point of Definition 12. The alternative-function represents that part of a language \mathcal{L} that can be most interestingly described as being external to the lexicon.

Consider the sentence initial occurrence of *John* with three alternative procedural analyses (a structural analysis plus a pointer) induced by the Axiom: *John* as subject, *John* as fronted object, and *John* as topicalised object. To say that the set of these three analyses constitutes an *alternative-set* for the Axiom means first of all that if string s maps one of the alternatives of the axiom into a logical form then s is grammatical (the *soundness* part), but it also means this set is *complete* in the sense that every grammatical string can be accommodated

by the alternative set: if string s is grammatical, then s can map at least one alternative of the Axiom to a logical form. So we may replace the Axiom by its image under the alternative function without losing logical forms.

The set of the three possible analyses of the sentence initial *John* is of course not complete in this sense, so it is not an alternative-set for the Axiom. But we assume that this set can be extended by some finite number of other analyses to a complete set of alternatives. Obviously, the finiteness assumption is crucial here for theoretical analysis of linguistic structure. At every point in the parse of a sentence we can only prepare for a limited variation in structure.⁴ These informal insights we will now define more carefully.

Definition 12 (Alternative Actions) An action α gives *alternatives* to $\mathcal{T}n \in PPT$ if for all $\mathcal{T}'n' \in LoFo$:

$$\mathcal{T}n \leq \mathcal{T}'n' \iff \exists \mathcal{T}''n'' : \langle \mathcal{T}n, \mathcal{T}''n'' \rangle \in \alpha \ \& \ \mathcal{T}''n'' \leq \mathcal{T}'n'.$$

A function $ALT : PPT \mapsto ACT$ is an *alternative function* if it assigns to every element $\mathcal{T}n$ of PPT an action α which gives *alternatives* to $\mathcal{T}n$ such that $\langle \mathcal{T}n, \mathcal{T}'n' \rangle \in ALT(\mathcal{T}n)$ implies that $ALT(\mathcal{T}'n') = 1$.

Action α gives alternatives to $\mathcal{T}n$ if every possible successful development $\mathcal{T}'n'$ is accessible from some element in $\{\mathcal{T}''n'' \in PPT \mid \langle \mathcal{T}n, \mathcal{T}''n'' \rangle \in \alpha\}$. Now we can consider a (finite) set of actions $\{\alpha_1 \dots \alpha_n\}$ to be *complete* w.r.t. $\mathcal{T}n$ if $\alpha_1 + \dots + \alpha_n$ gives alternatives to $\mathcal{T}n$. An alternative function assigns to every Pointer Partial Tree an action giving alternatives, but alternatives are only ‘unfolded’ one level: an alternative of an alternative is a fixpoint.

For purposes of presentation we will abuse this notation and use the formulation $\mathcal{T}'n' \in ALT(\mathcal{T}n)$ to mean that $\langle \mathcal{T}n, \mathcal{T}'n' \rangle \in \alpha$, where $ALT(\mathcal{T}n) = \alpha$. In figures 4 and 5 we have seen two images of the action $ALT([_a ?Ty(t)])$ on the argument $[_a ?Ty(t)]$, namely,

$$[_a ?Ty(t), [_0 ?Ty(e)], [_1 ?Ty(e \rightarrow t)]] , a0 \quad \text{and} \quad [_a ?Ty(t), [_* ?Ty(e)]] , a* .$$

That is, the Axiom can be expanded to a subject verb-phrase structure with the pointer at the subject, or it may be expanded to a structure expecting an ‘unfixed’ node annotated by a type e object. A third alternative is given in Figure 6. Of course, this does not exhaust the alternatives that will be required for a full analysis of English.

If we now reconsider the lexical action projected by the word *John*, we see that it wants a decoration $?Ty(e)$ for its actions to be executed. The Axiom itself does not have such a decoration, but by the ALT function (at least) two pointed partial trees are accessible with the pointer at a node with the right requirement.

⁴ Notice that the analyses are not purely structural for the sentence *John reads a book* and *a book John reads* eventually develop the same logical form starting from the Axiom. The analyses give a structural analysis *plus a pointer*. The location of this pointer determines the acceptability of a lexical item at that point in the parse.

The notion of an alternative-set can also have more formal content especially in case of structural underspecification. By *LFT* principles we have the following possible *ALT* function. $ALT([_a \downarrow \phi])$ has the elements

$$[_a [0 \phi], [1 \]] \text{ and } [_a [0 \], [1 \phi]]$$

The tree relation of *immediate dominance* can be developed into either function- or argument-daughter. This reflects the *LFT* principle $\langle \downarrow \rangle \leftrightarrow (\langle 0 \rangle \phi \vee \langle 1 \rangle \phi)$. As an other example of a formal alternative set consider $ALT([_a [* \phi]])$ which gives

$$[_a \phi], \text{ and } [_a [0[* \phi]], [1 \]], \text{ and } [_a [0 \], [1[* \phi]]].$$

This reflects the *LFT* principle $\langle * \rangle \phi \leftrightarrow (\phi \vee \langle \downarrow \rangle \langle * \rangle \phi)$.

Finally, a pointed partial tree model with requirement $\mathcal{M} = \langle \mathcal{PPT}, V, R \rangle$ together with language $\mathcal{L} = \langle W_{\mathcal{L}}, ALT_{\mathcal{L}} \rangle$ determine a *Parsing Model* $\mathcal{M}_{\mathcal{L}}$ in the following way.

Definition 13 (Parsing Models) A *Parsing Model* $\mathcal{M}_{\mathcal{L}}$ is a pair

$$\mathcal{M}_{\mathcal{L}} = \langle \mathcal{M}, \vdash_{\mathcal{L}} \rangle$$

where \mathcal{M} is a Pointed Partial Tree Model with Requirements, $\mathcal{L} = \langle W_{\mathcal{L}}, ALT_{\mathcal{L}} \rangle$ is a natural language, and $\vdash_{\mathcal{L}} \subseteq PPT \times PPT$ is the smallest set of \mathcal{L} -transitions $\{\vdash_{\mathcal{L}}^s \mid s \in W_{\mathcal{L}}^*\}$, satisfying the definition

- $\mathcal{T}n \vdash_{\mathcal{L}}^w \mathcal{T}'n'$ for $w \in W_{\mathcal{L}}$ iff there is a $\mathcal{T}''n'' \in ALT(\mathcal{T}n) : w(\mathcal{T}''n'') = \mathcal{T}'n'$,
- $\mathcal{T}n \vdash_{\mathcal{L}}^{\mathfrak{s}, s'} \mathcal{T}'n'$ iff there is a $\mathcal{T}''n'' : \mathcal{T}n \vdash_{\mathcal{L}}^{\mathfrak{s}} \mathcal{T}''n''$ and $\mathcal{T}''n'' \vdash_{\mathcal{L}}^{s'} \mathcal{T}'n'$.

With a Parsing Model $\mathcal{M}_{\mathcal{L}}$ we can associate the function $GRAM_{\mathcal{L}} : PPT \rightarrow \mathcal{P}(\mathcal{L})$, which maps a pointed partial tree to the set of $\mathcal{T}n$ -grammatical strings, that is, the strings that satisfy all of $R(\mathcal{T}n)$:

$$GRAM_{\mathcal{L}}(\mathcal{T}n) = \{s \in W_{\mathcal{L}}^* \mid \exists \mathcal{T}'n' \in LoFo(\mathcal{T}n) : \mathcal{T}n \vdash_{\mathcal{L}}^{\mathfrak{s}} \mathcal{T}'n'\},$$

If we start $GRAM_{\mathcal{L}}$ from the Axiom, $GRAM_{\mathcal{L}}(Axiom)$, then we get all \mathcal{L} -strings which produce logical forms starting from the empty context, that is, the grammatical \mathcal{L} sentences. We may start however from a more structured context, that is, we consider strings from \mathcal{L} that need a context to be grammatical, for instance, the string *Bill a newspaper* in the following example of *VP*-ellipsis: “*John reads a book. Bill a newspaper*”. Now, “John reads a book” may give the starting structure for “Bill a newspaper”

$$\mathcal{T}a0 = [_a \ ?Ty(t), [0 \ ?Ty(e)], [1 \ [0Fo(\lambda x \lambda y \mathbf{read}(x)(y)) [1 \ ?Ty(e)] \ ?]], a0$$

This is the \leq lowest upper bound of the structures that result from parsing the strings “*John reads a book*” and “*Bill reads a newspaper*”. “Abstraction” over ‘John’ and ‘a book’ creates this structure and $Bill\ a\ newspaper \in GRAM_{\mathcal{L}}(\mathcal{T}a0)$.

A Parsing Model $\mathcal{M}_{\mathcal{L}}$ is specific for a language \mathcal{L} . The vocabulary $W_{\mathcal{L}}$ of the language determines the ‘instruction packages’ that are present to construct transitions from the Axiom to a logical form. It creates a classification of the partial forms by the labels referred to in the *conditions* of the lexical actions. The conditions of these actions may be sensitive both to the structure-plus-annotations that has been constructed upto that point and to the requirements that have not yet been fulfilled.

The alternative function $ALT_{\mathcal{L}}$ gives the possible structural analyses that are typical of \mathcal{L} but, as such an analysis includes a pointer location, this function is also a determining factor in the *word order* of the strings in $GRAM_{\mathcal{L}}(Axiom)$. An extensive study of the variations and invariances across languages that Parsing Models can handle can be found in [5].

References

1. Blackburn, P., and W. Meyer-Viol. Linguistics, logic and finite trees, *Bulletin of IGPL*, 1994.
2. Kempson, R., W. Meyer-Viol and D. Gabbay. Language Understanding: a Procedural Perspective, in C. Retore (ed.), *Logical Aspects of Computational Linguistics*, First International Conference, LACL 1996, 228–247. Lecture Notes in Computer Science Vol 1328, Springer Verlag, 1997.
3. Kempson, R., W. Meyer-Viol and D. Gabbay. VP-ellipsis: towards a dynamic structural account. In Lappin, S., E. Benmamoun, *Fragments: studies in ellipsis and gapping*, OUP, 1999.
4. Kempson, R., and W. Meyer-Viol. Topic and Focus Structures: the Dynamics of Tree Growth, in *Proceedings of the 4th Formal Grammar Conference*, Saarbrücken, 1998.
5. Kempson, R., W. Meyer-Viol and D. Gabbay. *Dynamic Syntax*. Blackwell’s, Oxford, 2000.
6. Meyer-Viol, W., R. Kibble, R. Kempson and D. Gabbay. Indefinites as Epsilon Terms: A Labelled Deduction Account, In Bunt, H., and R. Muskens (eds.). *Computing Meaning: Current Issues in Computational Semantics*. Kluwer Academic Publishers, Dordrecht and Boston, 1997.

Derivational Minimalism Is Mildly Context–Sensitive*

Jens Michaelis

Universität Potsdam, Institut für Linguistik, PF 601553, 14415 Potsdam, Germany
michael@ling.uni-potsdam.de

Abstract. The change within the linguistic framework of transformational grammar from GB–theory to minimalism brought up a particular type of formal grammar, as well. We show that this type of a minimalist grammar (MG) constitutes a subclass of mildly context–sensitive grammars in the sense that for each MG there is a weakly equivalent linear context–free rewriting system (LCFRS). Moreover, an infinite hierarchy of MGs is established in relation to a hierarchy of LCFRSs.

1 Introduction

The change within the linguistic framework of transformational grammar from GB–theory to minimalism brought up a new formal grammar type, the type of a minimalist grammar (MG) introduced by Stabler (see e.g. [6,7]), which is an attempt of a rigorous algebraic formalization of the new linguistic perspectives. One of the questions that arise from such a definition concerns the weak generative power of the corresponding grammar class. Stabler [6] has shown that MGs give rise to languages not derivable by any tree adjoining grammar (TAG). But he leaves open the “... problem to specify how the MG–definable string sets compare to previously studied supersets of the TAG language class.” We address this issue here by showing that each MG as defined in [6] can be converted into a linear context–free rewriting system (LCFRS) which derives the same (string) language. In this sense MGs fall into the class of mildly context–sensitive grammars (MCSGs) rather informally introduced in [2] and described in e.g. [3].

The paper is structured as follows. We start by briefly repeating the definition of an LCFRS and the language it derives (Sect. 2). Turning to MGs, we then introduce the concept of a *relevant expression* in order to reduce the closure of an MG to such expressions (Sect. 3). Depending on this *relevant closure*, for a given MG we construct an LCFRS in detail and prove both grammars to be weakly equivalent (Sect. 4). Finally, an infinite hierarchy of MGs is introduced in relation to a hierarchy of LCFRSs. The former is unboundedly increasing, which is shown by presenting for each finite number an MG that derives a language with counting dependencies in size of this number (Sect. 5).

* This work has been carried out within the Innovationskolleg ‘Formal Models of Cognitive Complexity’ (INK 12) funded by the DFG. I especially wish to thank Marcus Kracht for inspiring discussions, and Peter Staudacher as well as an anonymous referee for a lot of valuable comments on a previous version of this paper.

2 Linear Context-Free Rewriting Systems

In order to keep the paper self-contained, in this section we quickly go through a number of definitions, which will be of interest in Sect. 4 again.

Definition 2.1 ([4]). A *generalized context-free grammar (GCFG)* is a five-tuple $G = (N, O, F, R, S)$ for which the conditions (G1)–(G5) hold.

- (G1) N is a finite non-empty set of *nonterminal symbols*.
- (G2) $O \subseteq \bigcup_{n \in \mathbb{N}} (\Sigma^*)^{n+1}$ for some finite non-empty set Σ of *terminal symbols* with $\Sigma \cap N = \emptyset$,¹ hence O is a set of finite tuples of finite strings in Σ .
- (G3) F is a finite subset of $\bigcup_{n \in \mathbb{N}} F_n$, where F_n is the set of partial functions from O^n to O , i.e. F_0 is the set of constants in O .
- (G4) $R \subseteq \bigcup_{n \in \mathbb{N}} (F \cap F_n) \times N^{n+1}$ is a finite set of (*rewriting*) *rules*.²
- (G5) $S \in N$ is the distinguished *start symbol*.

Let $G = (N, O, F, R, S)$ be a GCFG. A rule $r = (f, A_0, A_1, \dots, A_n) \in F_n \times N^{n+1}$ is generally written $A_0 \rightarrow f(A_1, \dots, A_n)$, and just $A_0 \rightarrow f$ in case $n = 0$. If the latter, i.e. if $f \in O$ then r is *terminating*, otherwise r is *nonterminating*. For $A \in N$ and $k \in \mathbb{N}$ the set $L_G^k(A) \subseteq O$ is given recursively in the following sense:

- (L1) $\theta \in L_G^0(A)$ for each terminating rule $A \rightarrow \theta \in R$.
- (L2) $\theta \in L_G^{k+1}(A)$, if $\theta \in L_G^k(A)$ or if there is $A \rightarrow f(A_1, \dots, A_n) \in R$ and there are $\theta_i \in L_G^k(A_i)$ for $1 \leq i \leq n$ such that $\theta = f(\theta_1, \dots, \theta_n)$ is defined.

We say A *derives* θ (in G) if $\theta \in L_G^k(A)$ for some $k \in \mathbb{N}$. In this case θ is called an A -*phrase* (in G). The *language derivable from A (by G)* is the set $L_G(A)$ of all A -phrases (in G), i.e. $L_G(A) = \bigcup_{k \in \mathbb{N}} L_G^k(A)$. The set $L(G) = L_G(S)$ is the *generalized context-free language (GCFL) (derivable by G)*.

Definition 2.2 ([5]). For every $m \in \mathbb{N}$ with $m \neq 0$ an m -*multiple context-free grammar (m -MCFG)* is a GCFG $G = (N, O, F, R, S)$ which satisfies (M1)–(M4).

- (M1) $O = \bigcup_{i=1}^m (\Sigma^*)^i$.
- (M2) For $f \in F$ let $n(f) \in \mathbb{N}$ be the number of arguments of f , i.e. $f \in F_{n(f)}$. For each $f \in F$ there are $r(f) \in \mathbb{N}$ and $d_i(f) \in \mathbb{N}$ for $1 \leq i \leq n(f)$ such that f is a (total) function from $(\Sigma^*)^{d_1(f)} \times \dots \times (\Sigma^*)^{d_{n(f)}(f)}$ to $(\Sigma^*)^{r(f)}$ for which (f1) and, in addition, the anti-copying condition (f2) hold.
 - (f1) Let $X = \{x_{ij} \mid 1 \leq i \leq n(f), 1 \leq j \leq d_i(f)\}$ be a set of pairwise distinct variables, and let $x_i = (x_{i1}, \dots, x_{id_i(f)})$ for $1 \leq i \leq n(f)$. For $1 \leq h \leq r(f)$ let f^h be the h -th component of f , i.e. $f(\theta) = (f^1(\theta), \dots, f^{r(f)}(\theta))$ for all $\theta = (\theta_1, \dots, \theta_{n(f)}) \in (\Sigma^*)^{d_1(f)} \times \dots \times (\Sigma^*)^{d_{n(f)}(f)}$. Then for each component f^h there is an $l_h(f) \in \mathbb{N}$ such that f^h can be represented by

¹ \mathbb{N} denotes the set of all non-negative integers. For any non-empty set M and $n \in \mathbb{N}$, M^{n+1} is the set of all $n+1$ -tuples in M , i.e. the set of all finite strings in M with length $n+1$. M^* is the set of all finite strings in M including the empty string ϵ .

² For any two sets M_1 and M_2 , $M_1 \times M_2$ is the set of all pairs with 1st component in M_1 and 2nd component in M_2 .

$$(c_h) \quad f^h(x_1, \dots, x_{n(f)}) = \zeta_{h0} z_{h1} \zeta_{h1} \dots z_{hl_h(f)} \zeta_{hl_h(f)}(f)$$

with $\zeta_{hl} \in \Sigma^*$ for $0 \leq l \leq l_h(f)$ and $z_{hl} \in X$ for $1 \leq l \leq l_h(f)$.

- (f2) For each $1 \leq i \leq n(f)$ and $1 \leq j \leq d_i(f)$ there is at most one $1 \leq h \leq r(f)$ and at most one $1 \leq l \leq l_h(f)$ such that $x_{ij} = z_{hl}$, i.e. z_{hl} is the only occurrence of $x_{ij} \in X$ in all righthand sides of $(c_1) - (c_{r(f)})$.
- (M3) There is a function d from N to \mathbb{N} such that, if $A_0 \rightarrow f(A_1, \dots, A_{n(f)}) \in R$ then $r(f) = d(A_0)$ and $d_i(f) = d(A_i)$ for $1 \leq i \leq n(f)$.
- (M4) $d(S) = 1$ for the start symbol S .

The language $L(G)$ is an *m-multiple context-free language (m-MCFL)*.

In case that $m = 1$ and that each $f \in F \setminus F_0$ is the concatenation function from $(\Sigma^*)^{n+1}$ to Σ^* for some $n \in \mathbb{N}$, G is a *context-free grammar (CFG)* and $L(G)$ a *context-free language (CFL)* in the usual sense.

Definition 2.3 ([8]). For $m \in \mathbb{N}$ with $m \neq 0$ an *m-MCFG* $G = (N, O, F, R, S)$ according to Definition 2.2 is an *m-linear context-free linear rewriting system (m-LCFRS)* if for all $f \in F$ the non-erasure condition (f3) holds in addition to (f1) and (f2).

- (f3) For each $1 \leq i \leq n(f)$ and $1 \leq j \leq d_i(f)$ there are $1 \leq h \leq r(f)$ and $1 \leq l \leq l_h(f)$ such that $x_{ij} = z_{hl}$, i.e. each $x_{ij} \in X$ has to appear in one of the righthand sides of $(c_1) - (c_{r(f)})$.

The language $L(G)$ is an *m-linear context-free rewriting language (m-LCFRL)*.

A grammar is also called an *MCFG (LCFRS)* if it is an *m-MCFG (m-LCFRS)* for some $m \in \mathbb{N} \setminus \{0\}$. A language is an *MCFL (LCFRL)* if it is derivable by some MCFG (LCFRS). The class of MCFGs is essentially the same as the class of LCFRSs. The latter was first described in [8] and has been studied in some detail in [9]. The “non-erasing property” (f3), motivated by linguistic considerations, is omitted in the general MCFG-definition. [5] shows that for each $m \in \mathbb{N} \setminus \{0\}$ the class of *m-MCFLs* and that of *m-LCFRLs* are equal. In Sect. 4 we in fact construct an LCFRS that is weakly equivalent to a given minimalist grammar.

3 Minimalist Grammars

We first give the definition of a minimalist grammar along the lines of [6].³ Then, we introduce a “concept of relevance” being of central importance later on.

Definition 3.1. A five-tuple $\tau = (N_\tau, \triangleleft_\tau^*, \prec_\tau, <_\tau, \text{Label}_\tau)$ fulfilling (E1)–(E3) is called an *expression (over a feature-set F)*.

³ Recall that we use ϵ to denote the empty string, whereas [6] uses λ .

- (E1) $(N_\tau, \triangleleft_\tau^*, \prec_\tau)$ is a finite, binary ordered tree. N_τ denotes the non-empty set of nodes. \triangleleft_τ^* and \prec_τ denote the usual relations of *dominance* and *precedence* defined on a subset of $N_\tau \times N_\tau$, respectively. I.e. \triangleleft_τ^* is the reflexive and transitive closure of \triangleleft_τ , the relation of *immediate dominance*.⁴
- (E2) $\prec_\tau \subseteq N_\tau \times N_\tau$ denotes the asymmetric relation of (*immediate*) *projection* which holds for any two siblings in $(N_\tau, \triangleleft_\tau^*, \prec_\tau)$, i.e. each node different from the root either (*immediately*) *projects* over its sibling or vice versa.
- (E3) The function $Label_\tau$ assigns a string from F^* to every leaf of $(N_\tau, \triangleleft_\tau^*, \prec_\tau)$, i.e. a leaf-label is a finite sequence of features from F .

The set of all expressions over F is denoted by $Exp(F)$.

Let F be a set of features. Consider $\tau = (N_\tau, \triangleleft_\tau^*, \prec_\tau, Label_\tau) \in Exp(F)$.

A node $x \in N_\tau$ is a *maximal projection*, if it is the root of τ or if x 's sister projects over x . Each $x \in N_\tau$ has a *head* $h(x) \in N_\tau$, a leaf such that $x \triangleleft_\tau^* h(x)$, and such that each $y \in N_\tau$ on the path from x to $h(x)$ with $y \neq x$ projects over its sister. The *head* of τ is the head of τ 's root r_τ .

τ has *feature* $f \in F$ if τ 's head-label starts with f . τ is *simple* (a *head*) if it consists of exactly one node, otherwise τ is *complex* (a *non-head*).

Suppose v and $\phi \in Exp(F)$ to be subtrees of τ with roots r_v and r_ϕ , respectively, such that $r_\tau \triangleleft_\tau r_v, r_\phi$. Then we take $[<v, \phi]$ ($[>\phi, v]$) to denote τ in case that $r_v <_\tau r_\phi$ and $r_v \prec_\tau r_\phi$ ($r_\phi \prec_\tau r_v$).

Definition 3.2 ([6]). A 4-tuple $G = (V, Cat, Lex, \mathcal{F})$ that obeys (N1)–(N4) is called a *minimalist grammar* (MG).

- (N1) $V = P \cup I$ is a finite set of *non-syntactic features*, where P is a set of *phonetic features* and I is a set of *semantic features*.
- (N2) Cat is a finite set of *syntactic features* partitioned into the sets *base*, *select*, *licensees* and *lensors* such that for each (*basic*) category $\mathbf{x} \in base$ the existence of $=\mathbf{x}$, $=\mathbf{X}$ and $\mathbf{X} =$ in *select* is possible, and for each $-\mathbf{x} \in licensees$ the existence of $+\mathbf{x}$ and $+\mathbf{X} \in lensors$. Moreover, the set *base* contains at least the category \mathbf{c} .
- (N3) Lex is a finite set of expressions over $V \cup Cat$ such that for each tree $\tau = (N_\tau, \triangleleft_\tau^*, \prec_\tau, Label_\tau) \in Lex$ the function $Label_\tau$ assigns a string from $select^*(lensors \cup \{\epsilon\})select^*(base \cup \{\epsilon\})licensees^*P^*I^*$ to each leaf in $(N_\tau, \triangleleft_\tau^*, \prec_\tau)$.
- (N4) The set \mathcal{F} consists of the structure building functions *merge* and *move* as defined in (me) and (mo), respectively.
- (me) The function *merge* is a partial mapping from $Exp(V \cup Cat) \times Exp(V \cup Cat)$ to $Exp(V \cup Cat)$. A pair of expressions (v, ϕ) belongs to $Dom(merge)$ if v has *feature* $=\mathbf{x}$, $=\mathbf{X}$ or $\mathbf{X} =$ and ϕ has category \mathbf{x} for some $\mathbf{x} \in base$.⁵ Then,

⁴ Up to an isomorphism N_τ is a unique *prefix closed* and *left closed* subset of \mathbb{N}^* , i.e. $\chi \in N_\tau$ if $\chi\chi' \in N_\tau$, and $\chi i \in N_\tau$ if $\chi j \in N_\tau$ for $\chi, \chi' \in \mathbb{N}^*$ and $i, j \in \mathbb{N}$ with $i < j$, such that for $\chi, \psi \in N_\tau$ hold: $\chi \triangleleft_\tau \psi$ iff $\psi = \chi i$ for some $i \in \mathbb{N}$, and $\chi \prec_\tau \psi$ iff $\chi = \omega i \chi'$ and $\psi = \omega j \psi'$ for some $\omega, \chi', \psi' \in \mathbb{N}^*$ and $i, j \in \mathbb{N}$ with $i < j$.

⁵ For each (partial) mapping f from a set M_1 into a set M_2 we take $Dom(f)$ to denote the *domain* of f , the subset of M_1 for which f is defined.

(me.1) $merge(v, \phi) = [_{<}v', \phi']$ if v is simple and has feature $=x$,

where v' and ϕ' are expressions resulting from v and ϕ , respectively, by deleting the feature the respective head-label starts with.

(me.2) $merge(v, \phi) = [_{<}v', \phi']$ if v is simple and has feature $=X$,

where v' and ϕ' are expressions resulting from v and ϕ , respectively, by deleting the feature the respective head label starts with. In addition the phonetic features π_ϕ of the head of ϕ are canceled in ϕ' , and the phonetic features π_v of the head of v are replaced by $\pi_v\pi_\phi$ in v' .

(me.3) $merge(v, \phi) = [_{<}v', \phi']$ if v is simple and has feature $X=$,

where v' and ϕ' are expressions resulting from v and ϕ , respectively, by deleting the feature the respective head label starts with. In addition the phonetic features π_ϕ of the head of ϕ are canceled in ϕ' , and the phonetic features π_v of the head of v are replaced by $\pi_\phi\pi_v$ in v' .

(me.4) $merge(v, \phi) = [_{>} \phi', v']$ if v is complex and has feature $=x$,

where v' and ϕ' are expressions as in case (me.1).

(mo) The function *move* is a partially defined mapping from $Exp(V \cup Cat)$ to $Exp(V \cup Cat)$. An expression v belongs to $Dom(move)$ in case that v has feature $+x$ or $+X \in licensors$, and v has exactly one maximal subtree ϕ that has feature $-x \in licensees$. Then,

(mo.1) $move(v) = [_{>} \phi', v']$ if v has feature $+X$

Here v' results from v by deleting the feature $+x$ from v 's head-label, while the subtree ϕ is replaced by a single node labeled ϵ . ϕ' is the expression resulting from ϕ just by deleting the licensee feature $-x$ that ϕ 's head-label starts with.

(mo.2) $move(v) = [_{>} \phi', v']$ if v has feature $+x$

Here v' results from v by deleting the feature $+x$ from v 's head-label, while within the subtree ϕ all non-phonetic features are deleted. ϕ' is the expression resulting from ϕ by deleting the licensee feature $-x$ that ϕ 's head-label starts with, and all phonetic features that appear in ϕ .

A feature of the form $=X$, $X=$ or $+X$ is called *strong*, one of the form $=x$ or $+x$ is called *weak*. A strong selection feature $=X$ or $X=$ triggers (*overt*) *head movement*, i.e. incorporation of the phonetic head-features of a possibly complex expression into the selecting head (cf. (me.2), (me.3)). A strong licensor $+X$ triggers *overt (phrasal) movement*, also called *pied-piping* (cf. (mo.1)). A weak licensor $+x$ triggers *covert (phrasal) movement* (cf. (mo.2)).

Example 3.3. Assume G_2 to be the MG for which $I = \emptyset$ and $P = \{ /a_1/, /a_2/ \}$, while $base = \{ c \} \cup \{ b_1, b_2, c_1, c_2, d_1, d_2 \}$, $select = \{ =b_1, =b_2, =c_1, =c_2, =d_1, =d_2 \}$, $licensees = \{ -l_1, -l_2 \}$ and $licensors = \{ +L_1, +L_2 \}$, and while *Lex* consists of

$$\begin{aligned}
\alpha_0 &= \mathbf{c} & \gamma_1 &= \mathbf{b}_2 + \mathbf{L}_1 \mathbf{c}_1 - \mathbf{l}_1 / \mathbf{a}_2 / & \delta_1 &= \mathbf{b}_2 + \mathbf{L}_1 \mathbf{d}_1 & \zeta_0 &= \mathbf{d}_2 \mathbf{c} \\
\beta_1 &= \mathbf{b}_1 - \mathbf{l}_1 / \mathbf{a}_2 / & \gamma'_1 &= \mathbf{c}_2 + \mathbf{L}_1 \mathbf{c}_1 - \mathbf{l}_1 / \mathbf{a}_2 / & \delta'_1 &= \mathbf{c}_2 + \mathbf{L}_1 \mathbf{d}_1 \\
\beta_2 &= \mathbf{b}_1 \mathbf{b}_2 - \mathbf{l}_2 / \mathbf{a}_1 / & \gamma_2 &= \mathbf{c}_1 + \mathbf{L}_2 \mathbf{c}_2 - \mathbf{l}_2 / \mathbf{a}_1 / & \delta_2 &= \mathbf{d}_1 + \mathbf{L}_2 \mathbf{d}_2
\end{aligned}$$

Then e.g. $\text{move}(\text{merge}(\gamma_2, \text{move}(\text{merge}(\gamma_1, \text{merge}(\beta_2, \beta_1)))) \in \text{Exp}(V \cup \text{Cat})$.

Let $G = (V, \text{Cat}, \text{Lex}, \mathcal{F})$ be an MG. Then $CL(G) = \bigcup_{k \in \mathbb{N}} CL^k(G)$ is the *closure of Lex (under the functions in \mathcal{F})*. For $k \in \mathbb{N}$ the sets $CL^k(G) \subseteq \text{Exp}(V \cup \text{Cat})$ are inductively defined by

$$(C1) \quad CL^0(G) = \text{Lex}$$

$$(C2) \quad CL^{k+1}(G) = CL^k(G)$$

$$\begin{aligned}
&\cup \{ \text{merge}(v, \phi) \mid (v, \phi) \in \text{Dom}(\text{merge}) \cap CL^k(G) \times CL^k(G) \} \\
&\cup \{ \text{move}(v) \mid v \in \text{Dom}(\text{move}) \cap CL^k(G) \}
\end{aligned}$$

Each $\tau \in CL(G)$ is called an *expression in G* . Such a τ is *complete (in G)* if its head-label is in $\{\mathbf{c}\}P^*I^*$ and each other of its leaf-labels is in P^*I^* . Hence, a complete expression has category \mathbf{c} , and this instance of \mathbf{c} is the only instance of a syntactic feature within all leaf-labels.

The (*phonetic*) *yield* $Y(\tau)$ of an expression $\tau \in \text{Exp}(V \cup \text{Cat})$ is the string created by concatenating τ 's leaf-labels “from left to right” and stripping off all non-phonetic features. $L(G) = \{Y(\tau) \mid \tau \in CL(G) \text{ with } \tau \text{ is complete}\}$ is the (*string*) *language (derivable by G)* and is called a *minimalist language (ML)*.

Example 3.4. Consider the MG G_2 from Example 3.3. Let $\tau^{(1)} = \text{merge}(\beta_2, \beta_1)$, $\tau^{(2)} = \text{merge}(\gamma_1, \tau^{(1)})$ and $v^{(2)} = \text{merge}(\delta_1, \tau^{(1)})$. For $k \in \mathbb{N}$ with $k \neq 0$ define

$$\begin{aligned}
\tau^{(2k+1)} &= \text{move}(\tau^{(2k)}), \quad \tau^{(4k)} = \text{merge}(\gamma_2, \tau^{(4k-1)}), \quad \tau^{(4k+2)} = \text{merge}(\gamma'_1, \tau^{(4k+1)}), \\
v^{(2k+1)} &= \text{move}(v^{(2k)}), \quad v^{(4k)} = \text{merge}(\delta_2, v^{(4k-1)}), \quad v^{(4k+2)} = \text{merge}(\delta'_1, v^{(4k+1)})
\end{aligned}$$

and $\phi^{(4k+2)} = \text{merge}(\zeta_0, v^{(4k+1)})$. Then we have

$$CL^1(G_2) \setminus CL^0(G_2) = \{\tau^{(1)}\} \quad \text{and} \quad CL^2(G_2) \setminus CL^1(G_2) = \{\tau^{(2)}, v^{(2)}\},$$

while for $k \in \mathbb{N} \setminus \{0\}$ and $-1 \leq i \leq 1$ we have

$$\begin{aligned}
CL^{4k+i}(G_2) \setminus CL^{4k+i-1}(G_2) &= \{\tau^{(4k+i)}, v^{(4k+i)}\} \quad \text{and} \\
CL^{4k+2}(G_2) \setminus CL^{4k+1}(G_2) &= \{\tau^{(4k+2)}, v^{(4k+2)}, \phi^{(4k+2)}\}.
\end{aligned}$$

The set of complete expressions in G_2 is $\{\alpha_0\} \cup \{\phi^{(4k+2)} \mid k \in \mathbb{N}, k \neq 0\}$, and the language derivable by G_2 is $\{\mathbf{a}_1/\mathbf{n}/\mathbf{a}_2/\mathbf{n} \mid n \in \mathbb{N}\}$.

Definition 3.5. For each MG $G = (V, \text{Cat}, \text{Lex}, \mathcal{F})$, an expression $\tau \in CL(G)$ is called *relevant (in G)* if it has property (R).

(R) For any $-\mathbf{x} \in \text{licensees}$ there is at most one maximal proper subtree $\tau_{-\mathbf{x}}$ of τ that has feature $-\mathbf{x}$.⁶

We take $\text{Rel}(G)$ to denote the set of all relevant expressions $\tau \in CL(G)$.

⁶ In fact, this kind of structure is characteristic of each $\tau \in CL(G)$ involved in creating a complete expression in G as will become clear immediately.

Let $G = (V, Cat, Lex, \mathcal{F})$ be an MG and consider $RCL(G) = \bigcup_{k \in \mathbb{N}} RCL^k(G)$, a particularly restricted closure of G , the *relevant closure (of G)*. For $k \in \mathbb{N}$ the sets $RCL^k(G)$ are inductively defined w.r.t. $Rel(G)$ by

- (R1) $RCL^0(G) = \{\tau \in Rel(G) \mid \tau \in Lex\}$
 (R2) $RCL^{k+1}(G)$
 $\quad = RCL^k(G)$
 $\quad \cup \{merge(v, \phi) \in Rel(G) \mid (v, \phi) \in \text{Dom}(merge) \cap RCL^k(G) \times RCL^k(G)\}$
 $\quad \cup \{move(v) \in Rel(G) \mid v \in \text{Dom}(move) \cap RCL^k(G)\}$

Lemma 3.6. *If $\tau \in CL^k(G) \cap Rel(G)$ for some $k \in \mathbb{N}$, then $\tau \in RCL^k(G)$.*

A proof of Lemma 3.6 can straightforwardly be obtained by an induction on $k \in \mathbb{N}$.⁷ On the other hand, it is an immediate consequence of the respective definitions that $RCL^k(G) \subseteq CL^k(G) \cap Rel(G)$ for each $k \in \mathbb{N}$. Thus,

Proposition 3.7. $Rel(G) = RCL(G)$.

Consequently, since each complete $\tau \in CL(G)$ has property (R), we can fix

Corollary 3.8. $L(G) = \{Y(\tau) \mid \tau \in RCL(G) \text{ with } \tau \text{ is complete}\}$.

This points out, why it is reasonable to call $RCL(G)$ the relevant closure (of G).

Remark 3.9. For G_2 as in Example 3.4, $RCL^k(G_2) = CL^k(G_2)$ for each $k \in \mathbb{N}$.

4 Weak Generative Power

Let $G_{MG} = (V, Cat, Lex, \mathcal{F})$ be an MG with $\{-1_i \mid 1 \leq i \leq m\}$ an enumeration of *licensees* for some $m \in \mathbb{N}$. We will construct an $m+2$ -MCFG $G = (N, O, F, R, S)$ that derives the same language as G_{MG} (Corollary 4.5).

Thus, in G the start symbol S will derive exactly those strings of phonetic features that are the yield of some complete $\tau \in CL(G_{MG})$. In order to achieve this, G will operate w.r.t. equivalence classes of a finite partition of $RCL(G_{MG})$ rather than on single expressions. For each $\tau \in RCL(G_{MG})$ there will be some nonterminal $T \in N$ coding τ 's structure as it matters to *merge* and *move*, but ignoring non-syntactic features (cf. (D1),(D2)). τ 's phonetic yield will be separately coded by some $p_T \in O$, a finite tuple of strings of phonetic features, that takes into account the structural information stored in T (cf. (D3),(D4)). p_T will be derivable from T in G as a finite recursion on functions in F , since for each particular application of *merge* or *move* in G_{MG} there will be some nonterminating rule in R simulating the corresponding structure building step in G_{MG}

⁷ Recall that $move(\tau)$ is defined for $\tau \in CL(G)$ only in case that there is exactly one maximal subtree of τ that has a particular licensee feature allowing the subtree's "movement into specifier position."

(Proposition 4.3).⁸ Vice versa, whenever some $p_T \in O$ will be derivable in G from some $T \in N$ that is different from S , there will be some $\tau \in RCL(G_{MG})$ to which T and p_T correspond as outlined above (Proposition 4.4).

W.l.o.g. we may assume the head-label of each $\tau \in Lex$ to contain at least some category feature $\mathbf{x} \in base$.⁹ Moreover, w.l.o.g. we may assume each $\tau \in Lex$ to be simple (a head). Thus, we can identify τ with its head-label. Doing so, for technical reasons we define sets $\text{suf}(Cat)$ and $\text{suf}(-1_i)$ for $1 \leq i \leq m$ by

$$\begin{aligned}\text{suf}(Cat) &:= \{\kappa \in Cat^* \mid \text{ex. } \kappa' \in Cat^* \text{ and } \pi\iota \in P^*I^* \text{ with } \kappa'\kappa\pi\iota \in Lex\} \\ \text{suf}(-1_i) &:= \{\kappa \in \text{suf}(Cat) \mid \kappa = \epsilon \text{ or } \kappa = -1_i\lambda \text{ for some } \lambda \in Cat^*\}\end{aligned}$$

By (N3) each $\text{suf}(-1_i)$ as well as $\text{suf}(Cat)$ is finite, and $\text{suf}(-1_i) \subseteq licensees^*$. Furthermore, we define

$$R_m := \{i_1 \dots i_n \mid n \in \mathbb{N}, i_1, \dots, i_n \in \{1, \dots, m\} \text{ with } i_j \neq i_k \text{ if } j \neq k\}$$

Note that R_m is finite, because in particular $|\alpha| \leq m$ for each $\alpha \in R_m$. Finally, we take **strong**, **weak**, **overt**, **covert**, **true**, **false**, **sim** and **com** to be pairwise distinct new symbols and now give the formal definitions of N and O , the set of nonterminals and the set of tuples of terminal strings, respectively, while we motivate these definitions in more detail, afterwards (cf. Definition 4.1).

• Each nonterminal $T \in N$ is either the start symbol S or an $m+2$ -tuple of the form $(\widehat{\mu}_0, \widehat{\mu}_1, \dots, \widehat{\mu}_m, t)$ with $t \in \{\mathbf{sim}, \mathbf{com}\}$ and $\widehat{\mu}_i$ a triple (μ_i, a_i, α_i) , where

- (n1) $\mu_0 \in \text{suf}(Cat)$ with $\mu_0 \neq \epsilon$ and $a_0 \in \{\mathbf{strong}, \mathbf{weak}\}$,
- (n2) $\mu_i \in \text{suf}(-1_i)$ and $a_i \in \{\mathbf{overt}, \mathbf{covert}, \mathbf{true}, \mathbf{false}\}$ for $1 \leq i \leq m$,
- (n3) $\alpha_i \in \{1, \dots, m\}^*$ for $0 \leq i \leq m$ with $\alpha_0\alpha_1 \dots \alpha_m \in R_m$

such that for $1 \leq j \leq m$, in addition, (n4) and (n5) hold.

- (n4) If $\alpha_j \neq \epsilon$ then $\alpha_i = \beta j \gamma$ for some $0 \leq i \leq m, i \neq j$, and $\beta, \gamma \in \{1, \dots, m\}^*$.
- (n5) $\mu_j \neq \epsilon$ iff $a_j \neq \mathbf{false}$ iff $\alpha_i = \beta j \gamma$ for some $0 \leq i \leq m, \beta, \gamma \in \{1, \dots, m\}^*$.

Take \triangleleft_T to be the following binary relation on $\{0, 1, \dots, m\}$ induced by the α_i 's:

$$(\triangleleft_T) \quad i \triangleleft_T j \text{ iff } \alpha_i = \beta j \gamma \text{ for some } \beta, \gamma \in \{1, \dots, m\}^*.$$

Hence, if $i \triangleleft_T j$ then $i \neq j$, $\mu_i \neq \epsilon$ and $a_i \neq \mathbf{false}$ by (n1), (n3)–(n5). Let \triangleleft_T^+ and \triangleleft_T^* denote the transitive and the reflexive, transitive closure of \triangleleft_T , respectively. Then take \prec_T to be the following binary relation on $\{0, 1, \dots, m\}$:

$$\begin{aligned}(\prec_T) \quad j \prec_T k \text{ iff } \alpha_i &= \beta j' \gamma k' \delta \\ &\text{for some } 0 \leq i, j', k' \leq m \text{ and } \beta, \gamma, \delta \in \{1, \dots, m\}^* \text{ such that } j' \triangleleft_T^* j, k' \triangleleft_T^* k.\end{aligned}$$

⁸ Note that for each relevant $\tau \in Lex$ there will be two terminating rules $T \rightarrow p_T \in R$ with $T \in N$ and $p_T \in O$ coding τ as just mentioned (cf. (r5)).

⁹ Recall that we are actually interested in complete expressions in $CL(G_{MG})$, created from expressions in Lex by a finite number of applications of *merge* and *move*.

It is easy to verify that the set N is in fact finite. Disregarding non-syntactic features, we can use N to characterize the relevant expressions in G_{MG} , which constitute the set $RCL(G_{\text{MG}})$ by Proposition 3.7. This set is generally not finite, itself. The phonetic yield of an expression from $RCL(G_{\text{MG}})$ can be characterized then as a particular tuple from $(P^*)^{m+2}$ depending on a corresponding nonterminal from N .

- We let $O = \bigcup_{i=0}^m (P^*)^{i+2}$, P the set of phonetic features in G_{MG} .

Consider $\tau \in RCL(G_{\text{MG}})$. For $1 \leq i \leq m$ take, if existing, τ_i to be the unique maximal proper subtree of τ that has licensee -1_i .¹⁰ Otherwise, take τ_i to be a single node labeled ϵ . Set $\tau_0 = \tau$ and for $0 \leq i \leq m$ let r_i denote the root of τ_i .

Now, let $T = (\hat{\mu}_0, \hat{\mu}_1, \dots, \hat{\mu}_m, t) \in N$ with $t \in \{\mathbf{sim}, \mathbf{com}\}$ and $\hat{\mu}_i = (\mu_i, a_i, \alpha_i)$ for $0 \leq i \leq m$ according to (n1)–(n5), let $p_T = (\pi_H, \pi_0, \pi_1, \dots, \pi_m) \in (P^*)^{m+2}$.

Definition 4.1. The pair (T, p_T) corresponds to τ if (D1)–(D4) are true.

- (D1) For $0 \leq i \leq m$, μ_i is the prefix of τ_i 's head-label consisting of just the syntactic features, and $t = \mathbf{sim}$ iff τ is simple.
- (D2) For $0 \leq i, j \leq m$ with $\mu_i, \mu_j \neq \epsilon$, $i \triangleleft_T^+ j$ iff $r_i \triangleleft_\tau^+ r_j$, and $i \prec_T j$ iff $r_i \prec_\tau r_j$.
- (D3) If $a_0 = \mathbf{weak}$ then $\pi_H = \epsilon$ and π_0 is the phonetic yield of $\tau_0 = \tau$ except for each substring that is the phonetic yield of some τ_i with $1 \leq i \leq m$ and $0 \triangleleft_T^+ i$ such that there is $1 \leq j \leq m$ with $0 \triangleleft_T^+ j \triangleleft_T^* i$ and $a_j = \mathbf{overt}$.
If $a_0 = \mathbf{strong}$ then π_H consists of the (ordered) phonetic features π of the head-label of $\tau_0 = \tau$, while π_0 is as in case $a_0 = \mathbf{weak}$ but lacking the substring π .
- (D4) For $1 \leq i \leq m$, if $a_i \in \{\mathbf{covert}, \mathbf{true}, \mathbf{false}\}$ then $\pi_i = \epsilon$. If $a_i = \mathbf{overt}$ then π_i is the phonetic yield of τ_i except for each substring that is the phonetic yield of some τ_j with $1 \leq j \leq m$ and $i \triangleleft_T^+ j$ such that there is $1 \leq k \leq m$ with $i \triangleleft_T^+ k \triangleleft_T^* j$ and $a_k = \mathbf{overt}$.

Note that (D1) provides a method to install a finite partition \mathcal{P} on $RCL(G_{\text{MG}})$: In the given manner, to each $\tau \in RCL(G_{\text{MG}})$ exactly one element belonging to the product $\text{suf}(Cat) \times \text{suf}(-1_1) \times \dots \times \text{suf}(-1_m) \times \{\mathbf{sim}, \mathbf{com}\}$ can be assigned.¹¹ (D2) can be seen then as introducing a refinement \mathcal{P}_{ref} of \mathcal{P} : Expressions τ from one equivalence class are distinguished w.r.t. proper dominance, \triangleleft_τ^+ , and precedence, \prec_τ , as it holds between each two distinct maximal projections r_i and r_j whose head-labels start with some licensee -1_i and -1_j , respectively. This can be achieved by assigning to each $\tau \in RCL(G_{\text{MG}})$ a particular $m+1$ -tuple $(\alpha_0, \alpha_1, \dots, \alpha_m)$ with $\alpha_i \in \{1, \dots, m\}^*$ for $0 \leq i \leq m$ according to (n3)–(n5).

Again let $T = (\hat{\mu}_0, \dots, \hat{\mu}_m, t) \in N$ with $t \in \{\mathbf{sim}, \mathbf{com}\}$ and $\hat{\mu}_i = (\mu_i, a_i, \alpha_i)$ for $0 \leq i \leq m$ as in (n1)–(n5), let $p_T = (\pi_H, \pi_0, \dots, \pi_m) \in (P^*)^{m+2}$ such that (T, p_T) corresponds to $\tau \in RCL(G_{\text{MG}})$ according to Definition 4.1. For $0 \leq i \leq m$ each μ_i and α_i as well as t is unique, because (D1) and (D2) hold.

¹⁰ Recall fn. 6.

¹¹ As a finite product of finite sets this product is also a finite set.

For each possible combination of a_i 's, $0 \leq i \leq m$, there is exactly one p_T that satisfies the requirements of (D3) and (D4).

The μ_i 's, the α_i 's and t determine the equivalence class of τ w.r.t. the refined partition \mathcal{P}_{ref} on $RCL(G_{\text{MG}})$. Since either $a_0 = \text{strong}$ or $a_0 = \text{weak}$, we have added the possibility to respectively code whether the category \mathbf{x} (of the head-label) of τ has to be selected by strong $=\mathbf{x}$ or \mathbf{x} or by weak $=\mathbf{x}$. For $1 \leq i \leq m$ we have $a_i = \text{false}$ iff there is no subtree τ_i that has licensee $-l_i$. By $a_i = \text{overt}$, $a_i = \text{covert}$ or $a_i = \text{true}$ we are able to respectively code, whether we expect the maximal subtree τ_i with licensee $-l_i$ to move overtly, covertly or just to move in a later derivation step. In this sense, according to (D3) and (D4), for $0 \leq i \leq m$ the component π_i of p_T specifies the “non-extractable” part of the phonetic yield of τ_i , i.e. no overt movement can apply such that a proper subconstituent of τ_i is extracted pied piping some (proper) subpart of π_i . Recall that $\tau_0 = \tau$, here.

Example 4.2. Let the MG G_2 be as in Example 3.4. Consider the partition \mathcal{P} on $RCL(G_2)$ induced by $\text{suf}(Cat) \times \text{suf}(-l_1) \times \text{suf}(-l_2) \times \{\text{sim}, \text{com}\}$. In case of G_2 the corresponding refinement \mathcal{P}_{ref} is identical with \mathcal{P} . $RCL(G_2) \setminus RCL^0(G_2)$, the set of complex expressions belonging to $RCL(G_2)$, divides into ten equivalence classes. One of which is finite, namely $\{\tau^{(1)}\}$, represented by $(b_2-l_2, -l_1, \epsilon, \text{com})$. The other classes and their respective representatives are

$$\begin{aligned} & \{\tau^{(4k+2)} \mid k \in \mathbb{N}\} \text{ and } (+L_1 c_1 - l_1, -l_1, -l_2, \text{com}), \\ & \{v^{(4k+2)} \mid k \in \mathbb{N}\} \text{ and } (+L_1 d_1, -l_1, -l_2, \text{com}), \\ & \{\tau^{(4k-1)} \mid k \in \mathbb{N}, k \neq 0\} \text{ and } (c_1 - l_1, \epsilon, -l_2, \text{com}), \\ & \{v^{(4k-1)} \mid k \in \mathbb{N}, k \neq 0\} \text{ and } (d_1, \epsilon, -l_2, \text{com}), \\ & \{\tau^{(4k)} \mid k \in \mathbb{N}, k \neq 0\} \text{ and } (+L_2 c_2 - l_2, -l_1, -l_2, \text{com}), \\ & \{v^{(4k)} \mid k \in \mathbb{N}, k \neq 0\} \text{ and } (+L_2 d_2, \epsilon, -l_2, \text{com}), \\ & \{\tau^{(4k+1)} \mid k \in \mathbb{N}, k \neq 0\} \text{ and } (c_2 - l_2, -l_1, \epsilon, \text{com}), \\ & \{v^{(4k+1)} \mid k \in \mathbb{N}, k \neq 0\} \text{ and } (d_2, \epsilon, \epsilon, \text{com}), \end{aligned}$$

and finally $\{\phi^{(4k+2)} \mid k \in \mathbb{N}, k \neq 0\}$ and $(c, \epsilon, \epsilon, \text{com})$. Now, let N_2 be the nonterminal set according to (n1)–(n5) for G_2 and consider e.g.

$$T = ((+L_1 c_1 - l_1, \text{weak}, 2), (-l_1, \text{overt}, \epsilon), (-l_2, \text{overt}, 1), \text{com}) \in N_2,$$

$$U = ((+L_1 d_1, \text{weak}, 2), (-l_1, \text{overt}, \epsilon), (-l_2, \text{overt}, 1), \text{com}) \in N_2$$

and $p_T = (\epsilon, /a_2/, /a_2/^{k+1}, /a_1/^{k+1})$, $p_U = (\epsilon, \epsilon, /a_2/^{k+1}, /a_1/^{k+1})$ with $k \in \mathbb{N}$. Then (T, p_T) and (U, p_U) correspond to $\tau^{(4k+2)}$ and $v^{(4k+2)}$, respectively.

Turning back to the general case of the MG G_{MG} , for the corresponding LCFRS G we will now define the set F of functions, manipulating tuples of tuples of terminal strings, and the set R of rewriting rules. In particular for all τ, v and $\phi \in RCL(G_{\text{MG}})$, if $\tau = \text{merge}(v, \phi)$ then rules of the form $T \rightarrow \text{merge}_{U,V}(U, V)$ and sometimes $T \rightarrow \text{Merge}_{U,V}(U, V)$ will belong to R (cf. (r1),(r2)), where $\text{merge}_{U,V}$ and $\text{Merge}_{U,V} \in F$ will be applicable to the pair (p_U, p_V) resulting in p_T . Similarly, if $\tau = \text{move}(v)$ there will be rules $T \rightarrow \text{move}_U(U)$ and sometimes $T \rightarrow \text{Move}_U(U) \in R$ (cf. (r3),(r4)), where move_U and $\text{Move}_U \in F$ will

be applicable to p_U calculating p_T as value. Here we have T, U and $V \in N$, while p_T, p_U and $p_V \in O$ such that (T, p_T) , (U, p_U) and (V, p_V) respectively correspond to τ, v and ϕ in the way given with Definition 4.1.

- The set F of functions and the set R of rewriting rules are simultaneously defined w.r.t. the occurrence of an $f \in F$ within an $r \in R$.

Nonterminating rules: First of all we define two initial rules by

$$(r0) \quad S \rightarrow \text{con}(T) \in R \text{ for } T = (\hat{\mu}_0, \hat{\mu}_1, \dots, \hat{\mu}_m, t) \in N$$

with $\hat{\mu}_0 = (\mathbf{c}, \mathbf{weak}, \epsilon)$, $\hat{\mu}_i = (\epsilon, \mathbf{false}, \epsilon)$ for $1 \leq i \leq m$ and $t \in \{\mathbf{sim}, \mathbf{com}\}$. The concatenation function $\text{con} : (P^*)^{m+2} \rightarrow P^*$ is given by $x \mapsto x_H x_0 x_1 \dots x_m$, where x denotes the $m+2$ -tuple $(x_H, x_0, x_1, \dots, x_m)$ consisting of the variables $x_H, x_0, x_1, \dots, x_m$.

For $\mathbf{x} \in \text{base}$ suppose that

$$\begin{aligned} \mathbf{x}\lambda &\in \text{suf}(Cat) \text{ with } \lambda \in Cat^*, \text{ i.e. } \lambda \in \text{licensees}^*, \\ s\kappa &\in \text{suf}(Cat) \text{ with } s \in \{=\mathbf{x}, =\mathbf{X}, \mathbf{X}=\} \text{ and } \kappa \in Cat^*, \\ \nu_i, \xi_i &\in \text{suf}(-1_i) \text{ for } 1 \leq i \leq m \text{ with } \nu_i = \epsilon \text{ or } \xi_i = \epsilon \end{aligned}$$

such that for $1 \leq j \leq m$,

$$\nu_j = \xi_j = \epsilon \text{ if } \lambda = -1_j \lambda' \text{ with } \lambda' \in Cat^*.$$

We choose $b_0, c_0 \in \{\mathbf{strong}, \mathbf{weak}\}$, $b_i, c_i \in \{\mathbf{overt}, \mathbf{covert}, \mathbf{true}, \mathbf{false}\}$ for $1 \leq i \leq m$, $\beta_i, \gamma_i \in \{1, \dots, m\}^*$ for $0 \leq i \leq m$, and $u, v \in \{\mathbf{sim}, \mathbf{com}\}$ such that

$$\begin{aligned} U &= ((s\kappa, b_0, \beta_0), (\nu_1, b_1, \beta_1), \dots, (\nu_m, b_m, \beta_m), u) \in N, \\ V &= ((\mathbf{x}\lambda, c_0, \gamma_0), (\xi_1, c_1, \gamma_1), \dots, (\xi_m, c_m, \gamma_m), v) \in N, \end{aligned}$$

and such that, additionally,

$$\begin{aligned} \text{if } s &\in \{=\mathbf{x}\} \text{ then } c_0 = \mathbf{weak}, \\ \text{if } s &\in \{=\mathbf{X}, \mathbf{X}=\} \text{ then } c_0 = \mathbf{strong} \text{ and } u = \mathbf{sim}. \end{aligned}$$

Proceeding, if $\lambda = \epsilon$ we set $j = 0$ and take

$$T' = ((\kappa, b_0, \gamma_0 \beta_0), \hat{\mu}'_1, \dots, \hat{\mu}'_m, \mathbf{com}) \in N,$$

whereas, if $\lambda = -1_j \lambda'$ for some $1 \leq j \leq m$ and $\lambda' \in Cat^*$ we take

$$\begin{aligned} T' &= ((\kappa, b_0, j\beta_0), \hat{\mu}'_1, \dots, \hat{\mu}'_m, \mathbf{com}) \in N, \\ T'' &= ((\kappa, b_0, j\beta_0), \hat{\mu}''_1, \dots, \hat{\mu}''_m, \mathbf{com}) \in N, \end{aligned}$$

where for $1 \leq i \leq m$ we have

$$\hat{\mu}'_i = \hat{\mu}''_i = \begin{cases} (\nu_i, b_i, \beta_i) & \text{if } i \neq j \text{ and } \xi_i = \epsilon \\ (\xi_i, c_i, \gamma_i) & \text{if } i \neq j \text{ and } \xi_i \neq \epsilon \end{cases}$$

$$\left. \begin{array}{l} \widehat{\mu}'_i = (\lambda, \mathbf{covert}, \gamma_0) \\ \widehat{\mu}''_i = (\lambda, \mathbf{overt}, \gamma_0) \end{array} \right\} \text{ if } i = j \text{ for } j \neq 0$$

Then, for $\text{merge}_{U,V}$ and $\text{Merge}_{U,V} \in F$ as defined below, we finally let

(r1) $T' \rightarrow \text{merge}_{U,V}(U, V) \in R$, and

(r2) $T'' \rightarrow \text{Merge}_{U,V}(U, V) \in R$ if $\lambda = -1_j \lambda'$ for $1 \leq j \leq m$ and $\lambda' \in \text{Cat}^*$.

Take x and y to be the $m+2$ -tuples $(x_H, x_0, x_1, \dots, x_m)$ and $(y_H, y_0, y_1, \dots, y_m)$ consisting of the variables $x_H, x_0, x_1, \dots, x_m$ and $y_H, y_0, y_1, \dots, y_m$, respectively.

The function $\text{merge}_{U,V} : (P^*)^{m+2} \times (P^*)^{m+2} \rightarrow (P^*)^{m+2}$ is defined by

$$(x, y) \mapsto (\widetilde{x}_H, \widetilde{x}_0, x_1 y_1, \dots, x_m y_m)$$

with $\left\{ \begin{array}{ll} \widetilde{x}_H = x_H y_H, \widetilde{x}_0 = x_0 y_0 & \text{in case } s = \mathbf{=x}, u = \mathbf{sim} \\ \widetilde{x}_H = x_H y_H, \widetilde{x}_0 = y_0 x_0 & \text{in case } s = \mathbf{=x}, u = \mathbf{com} \\ \widetilde{x}_H = x_H y_H, \widetilde{x}_0 = x_0 y_0 & \text{in case } s = \mathbf{=X}, b_0 = \mathbf{strong} \\ \widetilde{x}_H = x_H, \widetilde{x}_0 = x_0 y_H y_0 & \text{in case } s = \mathbf{=X}, b_0 = \mathbf{weak} \\ \widetilde{x}_H = y_H x_H, \widetilde{x}_0 = x_0 y_0 & \text{in case } s = \mathbf{X=}, b_0 = \mathbf{strong} \\ \widetilde{x}_H = x_H, \widetilde{x}_0 = y_H x_0 y_0 & \text{in case } s = \mathbf{X=}, b_0 = \mathbf{weak} \end{array} \right.$

The function $\text{Merge}_{U,V} : (P^*)^{m+2} \times (P^*)^{m+2} \rightarrow (P^*)^{m+2}$ is defined by

$$(x, y) \mapsto (\widetilde{x}_H, \widetilde{x}_0, x_1 y_1, \dots, x_{j-1} y_{j-1}, x_j y_j y_0, x_{j+1} y_{j+1}, \dots, x_m y_m)$$

with $\left\{ \begin{array}{ll} \widetilde{x}_H = x_H y_H, \widetilde{x}_0 = x_0 & \text{in case } s = \mathbf{=x} \\ \widetilde{x}_H = x_H y_H, \widetilde{x}_0 = x_0 & \text{in case } s = \mathbf{=X}, b_0 = \mathbf{strong} \\ \widetilde{x}_H = x_H, \widetilde{x}_0 = x_0 y_H & \text{in case } s = \mathbf{=X}, b_0 = \mathbf{weak} \\ \widetilde{x}_H = y_H x_H, \widetilde{x}_0 = x_0 & \text{in case } s = \mathbf{X=}, b_0 = \mathbf{strong} \\ \widetilde{x}_H = x_H, \widetilde{x}_0 = y_H x_0 & \text{in case } s = \mathbf{X=}, b_0 = \mathbf{weak} \end{array} \right.$

In order to illustrate the way in which G “does its job” concerning the operation merge , consider v and $\phi \in \text{RCL}(G_{\text{MG}})$ with respective head-labels $s\kappa\zeta$ and $\mathbf{x}\lambda\eta$ for some $\mathbf{x} \in \text{base}$, $s \in \{\mathbf{=x}, \mathbf{=X}, \mathbf{X=}\}$, $\kappa, \lambda \in \text{Cat}^*$ and some $\zeta, \eta \in P^* I^*$ such that $\tau = \text{merge}(v, \phi) \in \text{RCL}(G_{\text{MG}})$. Assume $U, V \in N$ and $p_U = (\rho_H, \rho_0, \dots, \rho_m)$, $p_V = (\sigma_H, \sigma_0, \dots, \sigma_m) \in (P^*)^{m+2}$ to be such that (U, p_U) and (V, p_V) respectively correspond to v and ϕ in the sense of Definition 4.1. Then U and V are as in (r1), and also as in (r2) in case $\lambda \neq \epsilon$.¹²

For T' as in (r1) and $p_{T'} = \text{merge}_{U,V}(p_U, p_V)$, $(T', p_{T'})$ corresponds to τ in any case. For T'' as in (r2) and $p_{T''} = \text{Merge}_{U,V}(p_U, p_V)$, also $(T'', p_{T''})$ corresponds to τ in case that $\lambda = -1_j \lambda'$ for some $1 \leq j \leq m$ and $\lambda' \in \text{Cat}^*$. In the latter case, in terms of the MG G_{MG} , by canceling the category feature \mathbf{x} from ϕ 's head-label while merging v and ϕ an expression τ_j that has licensee

¹² In particular, $\rho_j \sigma_j = \epsilon$ in case that $\lambda = -1_j \lambda'$ for some $1 \leq j \leq m$ and $\lambda' \in \text{Cat}^*$.

-1_j becomes a proper subtree of τ . Up to the deletion of the instance of \mathbf{x} , τ_j is identical with ϕ . I.e. in particular the phonetic yield of both is identical. In a derivation creating a complete expression, τ_j must move to check its licensee at some later derivation step. In (r1) this later application of *move* is expected to be covert, coded in T' by $\hat{\mu}'_j$ stating that the $j + 2$ -th component of $p_{T'}$ is empty. This chimes in with the definition of $\text{merge}_{U,V}$ according to which σ_0 , the “non-extractable” part of the yield of ϕ (i.e. of τ_j) specified by V , is “frozen” within the 2nd component of $p_{T'}$, the “non-extractable” part of the yield of τ specified by T' . In (r2) the later application of *move* is expected to be overt, coded in T'' by $\hat{\mu}''_j$. Here, applying $\text{Merge}_{U,V}$ to (p_U, p_V) , σ_0 remains a part on its own as $j + 2$ -th component of $p_{T''}$, since $\rho_j \sigma_j = \epsilon$.

If $s \in \{=\mathbf{x}, \mathbf{x}=\}$ then ϕ is selected strongly and v is simple. In this case $c_0 = \mathbf{strong}$, and therefore the (ordered) phonetic features σ of ϕ 's head coincide with σ_H , the 1st component of p_V . Applying $\text{Merge}_{U,V}$ or $\text{merge}_{U,V}$ to the pair (p_U, p_V) , σ_H will be incorporated into the selecting head v , i.e. concatenated with the phonetic features ρ of v “in the right manner.” Note that in terms of the LCFRS G depending on whether the category feature of v is expected to be selected strong or weak, i.e. whether $b_0 = \mathbf{strong}$ or $b_0 = \mathbf{weak}$, ρ is either ρ_H or ρ_0 according to (D3).¹³ If $s = =\mathbf{x}$ then ϕ is selected weakly. Thus, $c_0 = \mathbf{weak}$. Therefore, the phonetic features σ of ϕ 's head are a substring of σ_0 , the “non-extractable” part of the yield of ϕ , and $\sigma_H = \epsilon$.

Now, for some $1 \leq j \leq m$, suppose that

$$\begin{aligned} \nu_j &\in \text{suf}(-1_j) \text{ with } \nu_j = -1_j \lambda \text{ for some } \lambda \in \text{licensees}^*, \\ l\kappa &\in \text{suf}(Cat) \text{ with } l \in \{+1_j, +L_j\} \text{ and } \kappa \in Cat^*, \\ \nu_i &\in \text{suf}(-1_i) \text{ for } 1 \leq i \leq m \text{ with } i \neq j \end{aligned}$$

such that for $1 \leq k \leq m$ with $k \neq j$,

$$\nu_k = \epsilon \text{ if } \lambda = -1_k \lambda' \text{ with } \lambda' \in Cat^*.$$

Choose $b_0 \in \{\mathbf{strong}, \mathbf{weak}\}$, $b_i \in \{\mathbf{overt}, \mathbf{covert}, \mathbf{true}, \mathbf{false}\}$ for $1 \leq i \leq m$, and $\beta_i \in \{1, \dots, m\}^*$ for $0 \leq i \leq m$ such that

$$U = ((l\kappa, b_0, \beta_0), (\nu_1, b_1, \beta_1) \dots, (\nu_m, b_m, \beta_m), \mathbf{com}) \in N,$$

and such that, additionally,

$$\begin{aligned} \text{if } l = +L_j &\text{ then } b_j \in \{\mathbf{overt}, \mathbf{true}\}, \\ \text{if } l = +1_j &\text{ then } b_i \in \{\mathbf{covert}, \mathbf{true}\} \text{ for } 1 \leq i \leq m \text{ with } j \triangleleft_U^* i. \end{aligned}$$

If $\lambda = \epsilon$ we set $k = 0$ and take

$$T' = ((\kappa, b_0, \beta_j \beta), \hat{\mu}'_1, \dots, \hat{\mu}'_m, \mathbf{com}) \in N,$$

if $\lambda = -1_k \lambda'$ for some $1 \leq k \leq m$ and $\lambda' \in Cat^*$ we take

¹³ The existence of both possibilities is granted by the nonterminating rules (cf. (r5)).

$T' = ((\kappa, b_0, k\beta), \widehat{\mu}'_1, \dots, \widehat{\mu}'_m, \text{com}) \in N$ in general, and

$T'' = ((\kappa, b_0, k\beta), \widehat{\mu}''_1, \dots, \widehat{\mu}''_m, \text{com}) \in N$ in case that $b_j = \text{overt}$.

Here, $\beta = \zeta_0\eta_0$ if $0 \triangleleft_U j$, where $\zeta_0, \eta_0 \in \{1, \dots, m\}^*$ with $\beta_0 = \zeta_0j\eta_0$, and $\beta = \beta_0$ otherwise. Further, if $b_j = \text{overt}$ then for $1 \leq i \leq m$ we have

$$\left. \begin{array}{l} \widehat{\mu}'_i = (\lambda, \text{covert}, \beta_j) \\ \widehat{\mu}''_i = (\lambda, \text{overt}, \beta_j) \end{array} \right\} \text{ if } i = k$$

$$\widehat{\mu}'_i = \widehat{\mu}''_i = \begin{cases} (\epsilon, \text{false}, \epsilon) & \text{if } i = j \text{ and } j \neq k \\ (\nu_i, b_i, \zeta_i\eta_i) & \text{if } i \triangleleft_U j, \text{ where } \zeta_i, \eta_i \in \{1, \dots, m\}^* \text{ with } \beta_i = \zeta_ij\eta_i \\ (\nu_i, b_i, \beta_i) & \text{otherwise} \end{cases}$$

and, if $b_j \in \{\text{covert}, \text{true}\}$ then for $1 \leq i \leq m$ we have

$$\widehat{\mu}'_i = \begin{cases} (\lambda, \text{true}, \beta_j) & \text{if } i = k \\ (\epsilon, \text{false}, \epsilon) & \text{if } i = j \text{ and } j \neq k \\ (\nu_i, \text{true}, \beta_i) & \text{if } j \triangleleft_U^+ i \\ (\nu_i, b_i, \zeta_i\eta_i) & \text{if } i \triangleleft_U j, \text{ where } \zeta_i, \eta_i \in \{1, \dots, m\}^* \text{ with } \beta_i = \zeta_ij\eta_i \\ (\nu_i, b_i, \beta_i) & \text{otherwise} \end{cases}$$

Now, for the functions $\text{move}_U, \text{Move}_U \in F$ as defined below we let

(r3) $T' \rightarrow \text{move}_U(U) \in R$ in any case, and

(r4) $T'' \rightarrow \text{Move}_U(U) \in R$ if $b_j = \text{overt}$, $\lambda = -1_k\lambda'$ for $1 \leq k \leq m$, $\lambda' \in \text{Cat}^*$.

Again let x denote the $m+2$ -tuple $(x_H, x_0, x_1, \dots, x_m)$ consisting of the variables $x_H, x_0, x_1, \dots, x_m$.

The function $\text{move}_U : (P^*)^{m+2} \rightarrow (P^*)^{m+2}$ is defined by

$$x \mapsto (x_H, x_jx_0, x_1, \dots, x_{j-1}, \epsilon, x_{j+1}, \dots, x_m)$$

The function $\text{Move}_U : (P^*)^{m+2} \rightarrow (P^*)^{m+2}$ is defined by

$$x \mapsto \begin{cases} (x_H, x_0, \dots, x_{j-1}, x_j, x_{j+1}, \dots, x_m) & \text{for } k = j \\ (x_H, x_0, \dots, x_{k-1}, x_jx_k, x_{k+1}, \dots, x_{j-1}, \epsilon, x_{j+1}, \dots, x_m) & \text{for } k < j \\ (x_H, x_0, \dots, x_{j-1}, \epsilon, x_{j+1}, \dots, x_{k-1}, x_jx_k, x_{k+1}, \dots, x_m) & \text{for } k > j \end{cases}$$

Let us briefly discuss how the operation move is mimicked by G . Consider τ and $v \in RCL(G_{\text{MG}})$ for which $\tau = \text{move}(v)$. Hence v has head-label $l\kappa\zeta$ and a maximal subtree ϕ with head-label $-1_j\lambda\eta$ for some $1 \leq j \leq m$, $l \in \{+L_j, +1_j\}$, $\kappa, \lambda \in \text{Cat}^*$ and $\zeta, \eta \in P^*I^*$. For $1 \leq i \leq m$ let, if existing, v_i be the maximal subtree of v that has licensee -1_i , otherwise let v_i be the simple expression labeled ϵ . Thus, $\phi = v_j$. Take $U \in N$ and $p_U = (\rho_H, \rho_0, \dots, \rho_m) \in (P^*)^{m+2}$ to be such that (U, p_U) corresponds to v according to Definition 4.1. Then U is as in (r3), and also as in (r4) in case $\lambda \neq \epsilon$ and $b_j = \text{overt}$.

In case that $l = +1_j$ covert movement applies in terms of the MG G_{MG} . Looking at (D4) we see that in terms of the LCFRS G , by the respective $U \in N$ we ensure that $\rho_j = \epsilon$, but also that $\rho_i = \epsilon$ for each $1 \leq i \leq m$ with $j \triangleleft_T^+ i$. I.e. for each v_i that is a subtree of v_j we demand that ρ_i , the “non-extractable” part of the yield of v_i , is empty. As for the general MG-definition we must be aware of the linguistically rather pathological case that v_j in fact “hosts” some proper subtree v_i and at some later derivation step overt movement will apply to v_i but with empty phonetic yield. This becomes possible since v_j is moved covertly before v_i has been extracted such that v_i ’s yield gets “frozen” within v_j ’s yield which is “left behind.”¹⁴ After having lost the phonetic features this way, in terms of the LCFRS G the component b_i gets the value **true**, which triggers equal behavior w.r.t. a strong licenser and its weak counterpart. This reflects the fact that in terms of the MG G_{MG} overt movement of a constituent with empty phonetic yield has the same effect as moving this constituent covertly (up to leaving behind a “totally empty” structure in the latter case).

For T' as in (r3) and $p_{T'} = move_U(p_U)$, $(T', p_{T'})$ corresponds to τ in any case. For T'' as in (r4) and $p_{T''} = Move_U(p_U)$, also $(T'', p_{T''})$ corresponds to τ in case that $b_j = \mathbf{overt}$ and $\lambda = -1_k \lambda'$ for some $1 \leq k \leq m$ and $\lambda' \in Cat^*$. Whenever $\lambda = -1_k \lambda'$ for some $1 \leq k \leq m$ and $\lambda' \in Cat^*$, in terms of the MG G_{MG} an expression τ_k that has licensee -1_k becomes a proper subtree of τ by canceling the licensee -1_j from ϕ ’s head-label while moving ϕ to specifier position of v . In order to derive a complete expression, the licensee of τ_k has to be canceled by moving τ_k at some later derivation step. Thus, we again can distinguish two general possibilities:¹⁵ Of course, the corresponding instance of -1_k can be checked overtly or covertly. But, here we pay somewhat more attention than in the analogous “merge-case,” since it might be that v_k has already “lost” its phonetic yield by a particular application of covert movement at some earlier derivation step (see above). According to (D4), only in case that $b_j = \mathbf{overt}$ the corresponding component ρ_j of p_U may include some non-empty phonetic material, and only in this case we have to state explicitly two cases (r3) and (r4), analogous to (r1) and (r2) in the “merge-case.” The later application of *move* is “anticipated” as being covert in (r3), and as being overt in (r4).

Terminating rules: Let $\kappa\pi\iota \in Lex$ for some $\kappa \in Cat^*$, $\pi \in P^*$ and $\iota \in I^*$. Then, consider $a_0 \in \{\mathbf{strong}, \mathbf{weak}\}$ and $\pi_H, \pi_0 \in \{\pi, \epsilon\}$ with $\pi_H \neq \pi_0$ such that $\pi_0 = \pi$ iff $a_0 = \mathbf{weak}$. We define two terminating rules by

$$(r5) \quad T \rightarrow p_T \in R$$

with $T = ((\kappa, a_0, \epsilon), \hat{v}_1, \dots, \hat{v}_m, \mathbf{sim}) \in N$ and $p_T = (\pi_H, \pi_0, \epsilon, \dots, \epsilon) \in (P^*)^{m+2}$, where $\hat{v}_i = (\epsilon, \mathbf{false}, \epsilon)$ for $1 \leq i \leq m$.

¹⁴ This case is exemplified by the MG G_{con} , where P is $\{/e_1/, /e_2/, /e_3/\}$, I is \emptyset , *base* is $\{c, a_1, a_2, a_3\}$, *select* is $\{=a_1, =a_2, =a_3\}$, *licensor* is $\{+B_1, +B_2\}$, *licensees* is $\{-b_1, -b_2\}$, *Lex* consists of $a_1 - b_1 / e_1 /$, $=a_1 a_2 - b_2 / e_2 /$, $=a_2 + b_2 a_3 / e_3 /$ and $=a_3 + B_1 c$. The language $L(G_{con})$ derivable by G_{con} consists of the single string $/e_3 / e_2 / e_1 /$.

¹⁵ Like in the case when a subtree with licensee -1_j is introduced applying *merge*.

We will continue by proving the weak equivalence of G and G_{MG} . In order to finally do so, we show two propositions in advance.

Proposition 4.3. *Consider $\tau \in RCL(G_{\text{MG}})$. Let $q_0 \in \{\text{strong}, \text{weak}\}$, and let $q_i \in \{\text{overt}, \text{covert}\}$ for $1 \leq i \leq m$. Then there is some $T = (\hat{\mu}_0, \dots, \hat{\mu}_m, t) \in N$ with $t \in \{\text{sim}, \text{com}\}$ and $\hat{\mu}_i = (\mu_i, a_i, \alpha_i)$ for $0 \leq i \leq m$ as in (n1)–(n5), and there is some $p_T \in (P^*)^{m+2}$ with $p_T \in L_G(T)$ such that (a) and (b) hold.*

(a) (T, p_T) corresponds to τ according to Definition 4.1.

(b) $a_0 = q_0$ and $a_i \in \{q_i, \text{true}\}$ for $1 \leq i \leq m$ in case $\mu_i \neq \epsilon$.

Proof. We have $RCL(G_{\text{MG}}) = \bigcup_{k \in \mathbb{N}} RCL^k(G_{\text{MG}})$ and $L_G(T) = \bigcup_{k \in \mathbb{N}} L_G^k(T)$ for $T \in N$. Showing (4.3_k) by induction on $k \in \mathbb{N}$ we will prove the proposition.

(4.3_k) If $q_0 \in \{\text{strong}, \text{weak}\}$ and $q_i \in \{\text{overt}, \text{covert}\}$ for $1 \leq i \leq m$ then $\tau \in RCL^k(G_{\text{MG}})$ implies that there are $T = (\hat{\mu}_0, \dots, \hat{\mu}_m, t) \in N$ and $p_T \in (P^*)^{m+2}$ with $p_T \in L_G^k(T)$ fulfilling (a) and (b).

Since $RCL^0(G_{\text{MG}}) = \text{Lex}$, (4.3₀) holds according to (r5). Considering the induction step, let $\tau \in RCL^{k+1}(G_{\text{MG}})$. There is nothing to show if $\tau \in RCL^k(G_{\text{MG}})$. Otherwise, one of two general cases arises.

Either, there are v and $\phi \in RCL^k(G_{\text{MG}})$ with respective head-labels $s\kappa\zeta$ and $x\lambda\eta$ for some $x \in \text{base}$, $s \in \{=\mathbf{x}, \mathbf{x}=\}$, $\kappa, \lambda \in \text{Cat}^*$ and $\zeta, \eta \in P^*I^*$ such that $\tau = \text{merge}(v, \phi)$ holds. Let $b_0 = q_0$, let $c_0 = \text{strong}$ iff $s \in \{=\mathbf{x}, \mathbf{x}=\}$. Now choose

$$U = ((s\kappa, b_0, \beta_0), (\nu_1, b_1, \beta_1), \dots, (\nu_m, b_m, \beta_m), u) \in N,$$

$$V = ((x\lambda, c_0, \gamma_0), (\xi_1, c_1, \gamma_1), \dots, (\xi_m, c_m, \gamma_m), v) \in N$$

and $p_U, p_V \in (P^*)^{m+2}$ such that $p_U \in L_G^k(U)$, $p_V \in L_G^k(V)$, and such that (U, p_U) and (V, p_V) correspond to v and ϕ , respectively. Here $u, v \in \{\text{sim}, \text{com}\}$, $\nu_i, \xi_i \in \text{suf}(-1_i)$, $b_i, c_i \in \{\text{overt}, \text{covert}, \text{true}, \text{false}\}$ for $1 \leq i \leq m$, and $\beta_i, \gamma_i \in \{1, \dots, m\}$ for $0 \leq i \leq m$. In particular, each ν_i and ξ_i for $1 \leq i \leq m$ is unique. By induction hypothesis U, V and p_U, p_V not only exist, but for $1 \leq i \leq m$ they can also be chosen such that $b_i \in \{q_i, \text{true}\}$ for $\nu_i \neq \epsilon$, and $c_i \in \{q_i, \text{true}\}$ for $\xi_i \neq \epsilon$.

Recalling that merge is defined for the pair (v, ϕ) , we conclude that $u = \text{sim}$ if $s \in \{=\mathbf{x}, \mathbf{x}=\}$. Because, $\text{merge}(v, \phi) \in RCL(G_{\text{MG}})$ we also have $\nu_i, \xi_i \in \text{suf}(-1_i)$ for $1 \leq i \leq m$ with $\nu_i = \epsilon$ or $\xi_i = \epsilon$ such that $\nu_i = \xi_i = \epsilon$ if $\lambda = -1_i\lambda'$ with $\lambda' \in \text{Cat}^*$. Therefore, U and V are as in (r1) in any case, and also as in (r2) in case that $\lambda \neq \epsilon$. Hence (r1') is true in any case, and (r2') in case $\lambda \neq \epsilon$.

(r1') $T' \rightarrow \text{merge}_{U,V}(U, V) \in R$ and $p_{T'} = \text{merge}_{U,V}(p_U, p_V) \in L_G^{k+1}(T')$,

(r2') $T'' \rightarrow \text{Merge}_{U,V}(U, V) \in R$ and $p_{T''} = \text{Merge}_{U,V}(p_U, p_V) \in L_G^{k+1}(T'')$

with $T' \in N$ and $\text{merge}_{U,V} \in F$ as in (r1), $T'' \in N$ and $\text{Merge}_{U,V} \in F$ as in (r2).

Let $T = T''$ and $p_T = p_{T''}$ in case that $q_j = \text{overt}$ and $\lambda = -1_j\lambda'$ for some $1 \leq j \leq m$ and $\lambda' \in \text{Cat}^*$. Otherwise let $T = T'$ and $p_T = p_{T'}$. Comparing the definition of $\text{merge} \in \mathcal{F}$ to the definitions of T and $\text{merge}_{U,V}$ or $\text{Merge}_{U,V}$,

respectively, we see that (T, p_T) corresponds to $\tau = \text{merge}(v, \phi)$, and that T also satisfies the conditions imposed by (b).

The second general case provides an $v \in RCL^k(G_{MG})$ for which $\tau = \text{move}(v)$. Thus, v has head-label $l\kappa\zeta$ and a maximal subtree ϕ with head-label $-1_j\lambda\eta$ for some $1 \leq j \leq m$, $l \in \{+L_j, +1_j\}$, $\kappa, \lambda \in Cat^*$ and $\zeta, \eta \in P^*I^*$. For $b_0 = q_0$, by induction hypothesis we can fix existing

$$U = ((l\kappa, b_0, \beta_0), (\nu_1, b_1, \beta_1), \dots, (\nu_m, b_m, \beta_m), \text{com}) \in N,$$

and $p_U \in (P^*)^{m+2}$ with $p_U \in L_G^k(U)$ such that (U, p_U) corresponds to v . Again we have $\nu_i \in \text{suf}(-1_i)$, $b_i \in \{\text{overt}, \text{covert}, \text{true}, \text{false}\}$ for $1 \leq i \leq m$, and $\beta_i \in \{1, \dots, m\}$ for $0 \leq i \leq m$.¹⁶ By induction hypothesis, for all $1 \leq i \leq m$ with $\mu_i \neq \epsilon$ we can choose U even such that $b_j \in \{\text{overt}, \text{true}\}$ and $b_i \in \{q_i, \text{true}\}$ for $i \neq j$ in case $l = +L_j$, and such that $b_j \in \{\text{covert}, \text{true}\}$, $b_i \in \{\text{covert}, \text{true}\}$ for $j \triangleleft_T^+ i$ and $b_i \in \{q_i, \text{true}\}$ in case $l = +1_j$. Because $\text{move}(v) \in RCL(G_{MG})$, we conclude that (r3') holds in any case, and (r4') in case that $\lambda \neq \epsilon$ and $b_j = \text{overt}$.

$$(r3') \quad T' \rightarrow \text{move}_U(U) \in R \text{ and } p_{T'} = \text{move}_U(p_U) \in L_G^{k+1}(T')$$

$$(r4') \quad T'' \rightarrow \text{Move}_U(U) \in R \text{ and } p_{T''} = \text{Move}_U(p_U) \in L_G^{k+1}(T'')$$

with $T' \in N$ and $\text{move}_U \in F$ as in (r3), $T'' \in N$ and $\text{Move}_U \in F$ as in (r4).

Let $T = T''$ and $p_T = p_{T''}$ in case that $b_j = q_k = \text{overt}$ and $\lambda = -1_k\lambda'$ for some $1 \leq k \leq m$ and $\lambda' \in Cat^*$. Otherwise let $T = T'$ and $p_T = p_{T'}$. Looking at the definition of $\text{move} \in \mathcal{F}$ and the definitions of T and $\text{move}_{U,V}$ or $\text{Move}_{U,V}$, respectively, we see that (T, p_T) corresponds to τ , and that also (b) is true. \square

Let $T \in N$ and $p_T \in (P^*)^{m+2}$ be such that (a) and (b) of Proposition 4.3 are true w.r.t. given $\tau \in RCL(G_{MG})$, $q_0 \in \{\text{strong}, \text{weak}\}$ and $q_i \in \{\text{overt}, \text{covert}\}$ for $1 \leq i \leq m$. Note that this does not automatically imply that $p_T \in L_G(T)$.

Proposition 4.4. *If p_T is a T -phrase in G , i.e. if $p_T \in L_G(T)$ for some $T \in N$ with $T \neq S$ and $p_T \in (P^*)^{m+2}$, then there is some $\tau \in RCL(G_{MG})$ such that (T, p_T) corresponds to τ according to Definition 4.1.*

Proof. Recalling again that $RCL(G_{MG}) = \bigcup_{k \in \mathbb{N}} RCL^k(G_{MG})$ holds as well as $L_G(T) = \bigcup_{k \in \mathbb{N}} L_G^k(T)$, we also prove this proposition by induction on $k \in \mathbb{N}$.

$$(4.4_k) \quad \text{If } p_T \in L_G^k(T) \text{ then } (T, p_T) \text{ corresponds to some } \tau \in RCL^k(G_{MG}).$$

Since $\text{Lex} = RCL^0(G_{MG})$, (4.4₀) holds according to (r5). Considering the induction step, suppose that (4.4_k) is true for $k \in \mathbb{N}$. The crucial case arises from $p_T \in L_G^{k+1}(T) \setminus L_G^k(T)$ dividing into two general possibilities.

Either, $U, V \in N$ and $p_U, p_V \in (P^*)^{m+2}$ exist with $p_U \in L_G^k(U)$, $p_V \in L_G^k(V)$. U and V fulfill the restrictions applying in (r1) such that (r1'') is true for $T' \in N$ and $\text{merge}_{U,V} \in F$ as in (r1), or U and V even satisfy the restrictions applying in (r2) such that (r2'') is true for $T'' \in N$ and $\text{Merge}_{U,V} \in F$ as in (r2).

¹⁶ Recall that each ν_i for $0 \leq i \leq m$ and each β_i for $0 \leq i \leq m$ is unique.

(r1'') $T \rightarrow \text{merge}_{U,V}(U, V) \in R$, $p_T = \text{merge}_{U,V}(p_U, p_V)$ and $T = T'$

(r2'') $T \rightarrow \text{Merge}_{U,V}(U, V) \in R$, $p_T = \text{Merge}_{U,V}(p_U, p_V)$ and $T = T''$

Then, by induction hypothesis there are v and $\phi \in RCL^k(G_{MG})$ such that (U, p_U) and (V, p_V) respectively correspond to v and ϕ in the sense of Definition 4.1. Recall the restrictions that apply to U and V in (r1) or (r2), respectively. Because of these restrictions we may conclude that $\tau = \text{merge}(v, \phi)$ is not only defined according to (me), but also in $RCL^{k+1}(G_{MG})$ according to (R2). Since (r1'') or (r2'') is true, we refer to the respective definitions of T' and $\text{merge}_{U,V}$ or T'' and $\text{Merge}_{U,V}$ to see that (T, p_T) corresponds to τ .

Secondly, $U \in N$ and $p_U \in (P^*)^{m+2}$ may exist with $p_U \in L_G^k(U)$. The restrictions given with (r3) apply to U and (r3'') holds for T' and $\text{move}_U \in F$ as in (r3), or even the restrictions given with (r4) apply to U and (r4'') holds for T'' and $\text{Move}_U \in F$ as in (r4).

(r3'') $T \rightarrow \text{move}_U(U) \in R$, $p_T = \text{move}_U(p_U)$ and $T = T'$

(r4'') $T \rightarrow \text{Move}_U(U) \in R$, $p_T = \text{Move}_U(p_U)$ and $T = T''$

Here, by hypothesis there is an $v \in RCL^k(G_{MG})$ such that (U, p_U) corresponds to v in the sense of Definition 4.1. Similar as for (r1'') and (r2''), in cases (r3'') and (r4'') it is straightforward to show that $\text{move} \in \mathcal{F}$ is defined for v , and that (T, p_T) corresponds to $\tau = \text{move}(v) \in RCL^{k+1}(G_{MG})$. \square

Corollary 4.5. $\pi \in L(G)$ iff $\pi \in L(G_{MG})$ for each $\pi \in P^*$.

Proof. As for the “if”-part consider complete $\tau \in CL(G_{MG})$ with phonetic yield $\pi \in P^*$. Let $T = (\hat{\mu}_0, \dots, \hat{\mu}_m, t) \in N$ with $t \in \{\text{sim}, \text{com}\}$ and $\hat{\mu}_i = (\mu_i, a_i, \alpha_i)$ for $0 \leq i \leq m$ as in (n1)–(n5), let $p_T = (\pi_H, \pi_0, \dots, \pi_m) \in (P^*)^{m+2}$. Assume that (T, p_T) corresponds to τ according to (D1)–(D4). By Proposition 4.3 these T and p_T exist even such that $p_T \in L_G(T)$ and $a_0 = \text{weak}$. Since τ is complete, $\hat{\mu}_0 = (\text{c}, \text{weak}, \epsilon)$ and $\hat{\mu}_i = (\epsilon, \text{false}, \epsilon)$ for $1 \leq i \leq m$ by (D1), and therefore $\pi_1 = \dots = \pi_m = \epsilon$ by (D4). Moreover, τ 's phonetic head-features are “at the right place,” i.e. $\pi_H = \epsilon$ and $\pi_0 = \pi$ by (D3). Looking at (r0) and (L2), we conclude that $\pi \in L_G(S) = L(G)$.

To prove the “only if”-part, we start with some $\pi \in L(G) = L_G(S)$. The definition of R yields that each rule applying to S is of the form (r0). Thus, according to (L2) there is some $p_T = (\pi_H, \pi_0, \dots, \pi_m) \in (P^*)^{m+2}$ such that $p_T \in L_G(T)$ and $\pi = \text{con}(p_T)$ for $T \in N$ as in (r0). (T, p_T) corresponds to some $\tau \in RCL(G_{MG})$ by Proposition 4.4. This τ is complete by (D1), π is the yield of τ , since $\pi_H = \pi_1 = \dots = \pi_m = \epsilon$ and $\pi_0 = \pi$ by (D3) and (D4). \square

Consider the $m+2$ -LCFRS G as constructed above for a given MG G_{MG} whose set of licensees has cardinality $m \in \mathbb{N}$.

If all licensors in G_{MG} are strong, i.e. only overt movement is available, we do not have to define productions of the form (r1) and (r3) in case $\lambda \neq \epsilon$ for the corresponding $\lambda \in \text{licensees}^*$. More concretely, whenever in terms of the MG G_{MG} a subtree that has licensee $-x$ arises from applying *merge* or *move*, in

terms of the LCFRS G we do not have to predict the case that this licensee will be canceled by “covert movement.” Moreover, according to (D2), the structural relation of any two subtrees with different licensees is of interest only in (r3) for $\lambda \neq \epsilon$. Since productions of this kind are of no use at all, assuming all licensors in G_{MG} are strong, each $\hat{\mu}_i = (\mu_i, a_i, \alpha_i)$ of some $T = (\hat{\mu}_0, \dots, \hat{\mu}_m, t) \in N$ according to (n1)–(n5) can be reduced to its 1st component μ_i without losing any “necessary information.” This means that expressions from $RCL(G_{MG})$ in terms of the LCFRS G have to be distinguished only w.r.t. the partition \mathcal{P} induced by $\text{suf}(Cat) \times \text{suf}(-1_1) \times \dots \times \text{suf}(-1_m) \times \{\text{sim}, \text{com}\}$.

In case that all selection features in G_{MG} are weak, G is reducible even to an $m + 1$ -LCFRS. This is due to the fact that the 1st component of any $p_T \in (P^*)^{m+2}$ appearing in some complete derivation in G_{MG} is necessarily empty in this case. Therefore, if additionally $m = 0$, G_{MG} is a CFG. Vice versa, each CFG is weakly equivalent to some MG of this kind. This can be verified rather straightforwardly e.g. by starting with a CFG in Chomsky normal form.

5 A Hierarchy of MGs

Several well-known grammar types constitute a subclass of MCSGs. There are a.o. the two classes of head grammars (HGs) and TAGs as well as their generalized extensions, the classes of LCFRSs and multicomponent TAGs (MCTAGs), respectively.¹⁷ Like HGs and TAGs, LCFRSs and MCTAGs are weakly equivalent. LCFRSs and MCTAGs are the union of an infinite hierarchy of grammar classes, the respective hierarchy of m -LCFRSs and m -TAGs ($m \in \mathbb{N} \setminus \{0\}$). It is known that each m -LCFRL is an m -TAL, a language derivable by some m -TAG, and that each m -TAL is an $2m$ -LCFRL (cf. [9]). We can introduce an infinite hierarchy on the MG-class, as well.

Definition 5.1. For each $m \in \mathbb{N}$ an MG $G = (V, Cat, Lex, \mathcal{F})$ according to Definition 3.2 is an m -minimalist grammar (m -MG) if the cardinality of licensees is at most m . Then, the ML derivable by G is an m -minimalist language (m -ML).

Let $m \in \mathbb{N}$. It is clear that each m -ML is also an $m + 1$ -ML.

In Sect. 4 we have shown that each m -ML is an $m + 2$ -LCFRL. This result can be strengthened for $m = 0$, since the inclusion of 1-TALs within 2-LCFRLs is known to be proper (cf. [5]). Due to its “restricted type,” the 2-LCFRS that we have constructed for a given 0-MG can be transformed to a weakly equivalent 1-TAG. Thus, each 0-ML, each language whose realization plainly relies on the “extended” merging-type allowing for overt head movement, is even a 1-TAL, a tree adjoining language. Indeed the class of 0-MLs is a proper extension of the class of CFLs. Referring to the rather categorial type logical approach of [1], [6] presents a 0-MG that derives the copy language $\{ww \mid w \in \{1, 2\}^*\}$.

¹⁷ We define an MCTAG as in [9] and call it an m -TAG if derived sequences of auxiliary trees can be (simultaneously) adjoined to elementary tree-sequences of length at most $m \in \mathbb{N} \setminus \{0\}$. Then, 1-TAGs are TAGs in the usual sense, and vice versa.

Generalizing Example 3.3, for $m \in \mathbb{N}$ we consider the m -MG G_m with $I = \emptyset$, $P = \{/a_i/ \mid 1 \leq i \leq m\}$ and $base = \{c\} \cup \{b_i, c_i, d_i \mid 1 \leq i \leq m\}$, while $select = \{=b_i, =c_i, =d_i \mid 1 \leq i \leq m\}$, $licensees = \{-l_i \mid 1 \leq i \leq m\}$ and $lensors = \{+L_i \mid 1 \leq i \leq m\}$. Lex consists of the simple expressions c and $b_1-l_1/a_m/$, further $=b_i b_{i+1}-l_{i+1}/a_{m-i}/$, $=c_i+L_{i+1}c_{i+1}-l_{i+1}/a_{m-i}/$ and $=d_i+L_{i+1}d_{i+1}$ for $1 \leq i < m$, finally the 5 expressions $=b_m+L_1c_1-l_1/a_m/$, $=b_m+L_1d_1$, $=c_m+L_1c_1-l_1/a_m/$, $=c_m+L_1d_1$ and $=d_m c$. G_m derives the language $\{/a_1/^n \dots /a_m/^n \mid n \in \mathbb{N}\}$. We omit a proof here, pointing to the rather “deterministic manner” in which expressions in G_m can be derived.

Proposition 5.2. *For each $m \in \mathbb{N}$, $\{a_1^n \dots a_m^n \mid n \in \mathbb{N}\}$ is an m -ML.*

As shown in [5], for each $m \in \mathbb{N} \setminus \{0\}$, $\{a_1^n \dots a_{2m}^n \mid n \in \mathbb{N}\}$ is an m -LCFRL, while $\{a_1^n \dots a_{2m+1}^n \mid n \in \mathbb{N}\}$ is not. Because each m -ML is an $m+2$ -LCFRL, we therefore conclude that the hierarchy of ML-classes is infinitely increasing, i.e. there is no $m_b \in \mathbb{N}$ such that for all $m \in \mathbb{N}$ each m -ML is also an m_b -ML.

6 Conclusion

We have shown that MGs as defined in [6] constitute a weakly equivalent (sub-)class of MCSGs as described in e.g. [3]. Thus, the result contributes to solve a problem that has remained open in [6]. Further, we have established an infinite hierarchy on the MG-class in relation to other hierarchies of MCSG-formalisms.

References

1. Cornell, Th. L. A minimalist grammar for deriving the copy language. Report no.79, Working papers of the SFB 340, University Tübingen, 1996.
2. Joshi, A. K. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In Dowty, D., L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing. Theoretical, Computational and Psychological Perspective*, pages 206–250. Cambridge University Press, New York, NY, 1985.
3. Joshi, A. K., K. Vijay-Shanker, and D. J. Weir. The convergence of mildly context-sensitive grammar formalisms. In Sells, P., S. Shieber, and T. Wasow, editors, *Foundational Issues in Natural Language Processing*, pages 31–81. MIT Press, Cambridge, MA, 1991.
4. Pollard, C. J. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*. PhD thesis, Stanford University, Stanford, CA, 1984.
5. Seki, H., T. Matsumura, M. Fujii, and T. Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229, 1991.
6. Stabler, E. Derivational minimalism. In Retoré, C., editor, *Logical Aspects of Computational Linguistics*. LNCS No.1328, pages 68–95. Springer, Berlin, 1997.
7. Stabler, E. Acquiring languages with movement. *Syntax*, 1:72–97, 1998.
8. Vijay-Shanker, K., D. J. Weir, and A. K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the ACL (ACL '87)*, Stanford, CA, pages 104–111. ACL, 1987.
9. Weir, D. J. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1988.

Dominance Constraints in Context Unification

Joachim Niehren¹ and Alexander Koller²

¹ Programming Systems Lab, Universität des Saarlandes, Saarbrücken, Germany, www.ps.uni-sb.de/~niehren

² Department of Computational Linguistics, Universität des Saarlandes, Saarbrücken, Germany, www.coli.uni-sb.de/~koller

Abstract. Tree descriptions based on dominance constraints are popular in several areas of computational linguistics including syntax, semantics, and discourse. Tree descriptions in the language of context unification have attracted some interest in unification and rewriting theory. Recently, dominance constraints and context unification have both been used in different underspecified approaches to the semantics of scope, parallelism, and their interaction. This raises the question whether both description languages are related. In this paper, we show for a first time that dominance constraints can be expressed in context unification. We also prove that dominance constraints extended with parallelism constraints are equal in expressive power to context unification.

Keywords. Computational linguistics, underspecification, tree descriptions, computational logics, unification theory.

1 Introduction

Logical tree descriptions are popular in many areas of computational linguistics and computer science. They are used to model data structures in logic programming, to reason with propositions and proofs in automated deduction, and to represent all kinds of syntactic or semantic structures in computational linguistics. In this paper, we investigate the relationship between tree descriptions based on dominance constraints and those in the language of context unification.

Two Languages of Tree Descriptions. Dominance constraints are popular for describing trees throughout computational linguistic. In syntax, they serve for deterministic parsing [18] and to combine TAG and unification grammars [39]. In underspecified treatments of scope ambiguities, variants of dominance constraints appear somewhat implicitly in many places [28,5] and explicitly in two recent approaches [10,23]. Even more recently, dominance constraints have been applied to discourse semantics [11], and they have been used to model information growth and partiality [20].

In general, the problem of solving dominance constraints is NP-complete [15]. Nevertheless, [9] describes an implementation of a dominance constraint solver which runs efficiently on practical examples from scope underspecification and

discourse. This solver is implemented based on finite set constraints in the Mozart System [22], the most recent implementation of the concurrent constraint programming language Oz [36].

Context unification (CU) was introduced in rewriting and unification theory [7, 30]. CU can be considered as second-order linear unification [16], which is a restriction of higher-order unification, or as an extension of string unification [32]. The decidability question for CU is a prominent open problem [29]. A decidable fragment of CU called *stratified unification* has been used to show the decidability of distributive unification [31] and for solving one-step rewriting constraints [24, 26]. It is shown in [33] that context unification with two context variables – each of which may occur an arbitrary number of times – is decidable. The proof is by reduction to string unification, which is decidable according to Makanin’s famous result [17, 34].

Tree Descriptions in Semantic Underspecification. Recently, tree descriptions based on dominance constraints and context unification have been proposed for the same application to natural-language semantics [10, 24, 14]. There, the goal was to find a uniform language providing underspecified representations for the semantics of scope, parallelism, anaphora, and their interactions (for a survey of semantic underspecification, see e.g. [8]). The common characteristic of both approaches is that they view the formulae of the semantic representation as trees and describe these trees. The role of dominance constraints in this context is to describe scope ambiguities; they are extended with constructions for describing parallelism and anaphoric and variable binding to obtain the Constraint Language over Lambda Structures (CLLS).

Contribution. If CU and CLLS are used for the same application, an immediate question is if there is a formal relationship between the two languages that says something about their relative expressive power.

In this paper, we show that the fragment of CLLS which provides dominance and parallelism constraints is equal in expressive power to context unification. We do this by giving satisfiability preserving, polynomial time encodings in both directions. The most interesting (and non-obvious) part of the construction is to encode dominance constraints in context unification. Once we know how to do that, the rest of this direction is easy. The inverse encoding can be deduced from a result in [24].

Plan of the Paper. In Section 2 we illustrate why encoding dominance constraints into context unification is nontrivial. In Section 3, we recall the fundamental definitions of trees and contexts. These definitions are used in Section 4, where we present dominance and parallelism constraints and briefly review a linguistic example. They are also used in Section 5, where we recall context unification, discuss first results on its expressive power, and give a linguistic example, too. Section 6 contains the encoding of dominance and parallelism constraints in CU, and Section 7 the inverse encodings. We conclude in Section 8.

2 What Is the Problem?

It is not obvious to encode dominance constraints in context unification. The problem is that both languages describe trees from different perspectives. We now illustrate the difference by an example.

Two Perspectives on Trees. Dominance constraints (and CLLS as a whole) describe relations between the *nodes* of the same tree (or a more general λ -structure). In contrast, the language of CU models relations between different *trees* and *contexts*. In CU, one cannot speak directly about the nodes of a tree; but we shall use contexts to speak about occurrences of subtrees later in this paper.

The perspective taken when speaking about the nodes of the same tree is called *internal* in [4], in contrast to the *external* perspective where one relates several trees. Both views have a long tradition in logics. The internal view is taken in modal logic and in second-order monadic logic (*SnS*) [27], whereas the external view is popular in unification theory [19,6,2] and for set constraints [12,1,37]. In feature logics [13,35], both perspectives have been employed and compared [3, 21].

Dominance versus Subtree Constraints. Dominance constraints contain node valued variables that we write as capital letters X, Y, Z . An atomic dominance constraint $X \triangleleft^* Y$ holds in a tree (structure) if the node denoted by X is above (strictly or not) the node denoted by Y .

A first idea for encoding dominance constraints in CU is to replace each atomic dominance constraint by a *subtree constraint* [38], which can be expressed in CU in a very simple way. Subtree constraints have tree valued variables for which we use lower case letters x, y, z . A subtree constraint $x \gg y$ says that the denotation of y is a subtree of the denotation of x .

Although they look very similar, there is an important difference between dominance and subtree constraints: Dominance constraints can speak about *occurrences* of subtrees by specifying their root nodes, whereas subtree constraints can't.

An Example. Because of this difference, the naive encoding of dominance as subtree constraints does not preserve satisfiability. As an example, we consider the dominance constraint in (1) and the “corresponding” subtree constraint (2).

$$\begin{array}{ll}
 (1) & X : f(X_1, X_2) \wedge X_1 \triangleleft^* Y \wedge X_2 \triangleleft^* Y \quad = \\
 (2) & x = f(x_1, x_2) \wedge x_1 \gg y \wedge x_2 \gg y \quad \neq
 \end{array}$$

The dominance constraint (1) is depicted by the graph to the right. It describes trees in which the node denoted by X is labeled with a binary function symbol f and has two (distinct) children denoted by X_1 and X_2 . Furthermore, it requires that there is a node, denoted by Y , which is below X_1 and X_2 . This is impossible in a tree. Thus, (1) is unsatisfiable.

The subtree constraint (2) requires that x , x_1 , x_2 , and y denote trees. The tree for x has two direct subtrees denoted by x_1 and x_2 , which in turn have a common subtree y (not necessarily at the same position). The subtree constraint (2) is satisfiable; one solution is obtained by mapping y , x_1 , and x_2 to the tree a , and x to the tree $f(a, a)$. The two occurrences of y in the subtree constraint (2) refer to different occurrences the tree a in $f(a, a)$.

3 Trees and Contexts

Understanding the notions of *trees* and *contexts* is essential for this paper. We next define both notions and explain the views on them we will adopt.

We assume a signature Σ of *function symbols* ranged over by f, g , each of which is equipped with a fixed arity $\text{ar}(f) \geq 0$. Constants, ranged over by a, b , are function symbols with arity 0. We assume that Σ contains at least two function symbols, one of which is not constant. Note that we do not restrict our signature to be finite.

Trees. A (finite constructor) *tree* τ is a ground term constructed from function symbols in Σ . For instance, $f(f(a, b), c)$ is a tree whose root node is labeled with f and which has three leaves labeled by a, b, c .

An equivalent definition of trees, which makes the nodes and node labels of the tree explicit, is based on *tree domains*. Let \mathbb{N} be the set of natural numbers $n \geq 1$ and \mathbb{N}^* the set of words over natural numbers. ϵ is the *empty word*, and the *concatenation* of two words π and π' is written by juxtaposition $\pi\pi'$. A path π' is a *prefix* of π if there is a π'' such that $\pi'\pi'' = \pi$.

A *tree domain* D is a nonempty prefixed-closed subset of \mathbb{N}^* . That is, D contains *paths* which are words of positive integers; they can intuitively be identified with the nodes of the tree. A *labeling function* is a function $L : D \rightarrow \Sigma$ defined on a tree domain D which satisfies for all $\pi \in D$ and $k \in \mathbb{N}$: $\pi k \in D$ iff $1 \leq k \leq \text{ar}(L(\pi))$. A tree, then, is a pair (D, L) of a tree domain and a labeling function.

The two definitions of trees can be connected by associating with each tree τ a tree domain D_τ and a labeling function $L_\tau : D_\tau \rightarrow \Sigma$ as follows:

$$\begin{aligned} D_{f(\tau_1, \dots, \tau_n)} &= \{\epsilon\} \cup \{k\pi \mid 1 \leq k \leq n, \pi \in D_{\tau_k}\} \\ L_{f(\tau_1, \dots, \tau_n)}(\pi) &= \begin{cases} f & \text{if } \pi = \epsilon \\ L_{\tau_k}(\pi') & \text{if } \pi = k\pi', 1 \leq k \leq n, \pi' \in D_{\tau_k} \end{cases} \end{aligned}$$

For instance, the tree $\tau = f(g(a), b)$ has the tree domain $D_\tau = \{\epsilon, 1, 11, 2\}$ and the labeling function L_τ with $L_\tau(\epsilon) = f$, $L_\tau(1) = g$, $L_\tau(11) = a$, and $L_\tau(2) = b$.

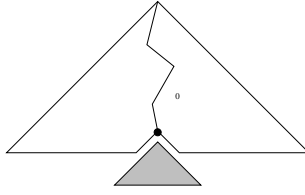


Fig. 1. A context γ with hole π_0 .

Lemma 1. *For every finite tree domain D and labeling function $L : D \rightarrow \Sigma$ there exists a unique tree τ such that $D_\tau = D$ and $L_\tau = L$.*

Whenever τ is a tree and π a path in D_τ , we define the *subtree* $\tau.\pi$ of τ at π as the unique tree with the following properties (otherwise $\tau.\pi$ is undefined):

$$\begin{aligned} D_{\tau.\pi} &= \{\pi' \mid \pi\pi' \in D_\tau\} \\ L_{\tau.\pi}(\pi') &= L_\tau(\pi\pi') \quad \text{for all } \pi\pi' \in D_\tau \end{aligned}$$

Lemma 2. *For all trees τ and paths $\pi \in D_\tau$ if $f = L_\tau(\pi)$ and $\text{ar}(f) = n$ then $\tau.\pi = f(\tau.(\pi 1), \dots, \tau.(\pi n))$.*

Contexts. Intuitively, a context is a tree with a hole. More formally, we introduce a special symbol \bullet that we call *hole marker* and assign it the arity $\text{ar}(\bullet) = 0$. A *context* γ is a ground term over $\Sigma \cup \{\bullet\}$ which contains exactly one occurrence of the hole marker. For instance, $f(a, f(\bullet, b))$ is a context, but $f(\bullet, f(\bullet, b))$ isn't. We shall use the letter τ for trees over Σ and the letter γ for contexts (i.e. special trees over $\Sigma \cup \{\bullet\}$).

The *hole* of a context γ is the occurrence of the hole marker in γ . More precisely, the hole is the unique path $\pi_0 \in D_\gamma$ such that $L_\gamma(\pi_0) = \bullet$. Fig. 1 shows a context with hole π_0 .

We will freely consider contexts as functions that map trees to trees. Application $\gamma[\tau]$ of a context γ to a tree τ is defined by

$$\gamma[\tau] = \gamma[\tau/\bullet]$$

That is, $\gamma[\tau]$ is the result of substituting the hole marker \bullet in γ by τ . The context \bullet corresponds to the identity function on trees. This illustrates that the hole marker can be seen as a λ -bound variable (rather than a constant or a free variable). Concatenation $\gamma \circ \gamma'$ of contexts seen as functions can be defined as $\gamma[\gamma'/\bullet]$.

Lemma 3. *For a context γ with hole π and all trees τ , it holds that $\gamma[\tau].\pi = \tau$.*

Contexts in Trees. Since contexts are ground terms over a special signature, we have already defined subtree selection for contexts. If γ is a context and $\pi \in D_\gamma$ then $\gamma.\pi$ is either a tree over Σ or a context. It is a context iff π is a prefix (proper or not) of the hole of γ .

Given a tree τ and a path $\pi \in D_\tau$, we write the context obtained by replacing the subtree of τ at π with the symbol \bullet as $\tau^\bullet\pi$. More precisely, $\tau^\bullet\pi$ is defined as the context with domain $D_{\tau^\bullet\pi} = \{\pi' \mid \pi \text{ not a proper prefix of } \pi'\}$ and the labeling function which assigns $L_{\tau^\bullet\pi}(\pi') = L_\tau(\pi')$ for all $\pi' \in D_{\tau^\bullet\pi} \setminus \{\pi\}$ and $L_{\tau^\bullet\pi}(\pi) = \bullet$.

Lemma 4. *For all τ and $\pi \in D_\tau$ it holds that $\tau^\bullet\pi[\tau.\pi] = \tau$.*

Given a prefix π_1 of π_2 and a tree τ with $\pi_2 \in D_\tau$, we define $\tau_{\pi_2}^{\pi_1}$ to be the context of τ between π_1 and π_2 :

$$\tau_{\pi_2}^{\pi_1} = (\tau^\bullet\pi_2).\pi_1 = (\tau.\pi_1)^\bullet\pi \quad \text{where } \pi_1\pi = \pi_2.$$

4 Dominance and Parallelism Constraints

We now present the language of dominance and parallelism constraints which is a fragment of the constraint language over λ -structures CLLS [10]. CLLS also has constructs for dealing with variable binding and anaphora, but we ignore these for the purpose of this paper.

Our notion of dominance constraints differs slightly from the one used e.g. by Vijay-Shanker [39]; these languages are mostly based on feature trees as common in computational linguistics, whereas our trees are *constructor trees*.

4.1 Tree Structures

We first define *tree structures*, logical structures representing trees. Tree structures fix the interpretation of a set of predicate symbols. Based on tree structures, we will define the syntax and semantics of our constraint language in the usual Tarskian style.

We associate with every tree τ a logical structure \mathcal{M}^τ , the *tree structure of τ* . The domain of the *tree structure* \mathcal{M}^τ coincides with the tree domain of τ . Furthermore, \mathcal{M}^τ provides interpretations for the binary relation symbol \triangleleft^* , a 4-ary relation symbol $./.\sim./.$, and a relation symbol $:f$ of arity $\text{ar}(f) + 1$ for every function symbol $f \in \Sigma$. We use the same symbols for relations and relation symbols; there shouldn't be any danger of confusion. For instance, we write $\pi \triangleleft^* \pi'$ in order to say that the relation \triangleleft^* holds for the pair (π, π') , whereas $X \triangleleft^* X'$ is an atomic constraint built from the relation symbol \triangleleft^* and variables X, X' . A relation symbol is generally interpreted by the relation of the same name.

If $f \in \Sigma$ and $\text{ar}(f) = n$, then the *labeling relation* $\pi : f(\pi_1, \dots, \pi_n)$ is true in \mathcal{M}^τ iff $L_\tau(\pi) = f$ and $\pi_i = \pi_i$ for all $1 \leq i \leq n$. The *dominance relation* $\pi \triangleleft^* \pi'$ is

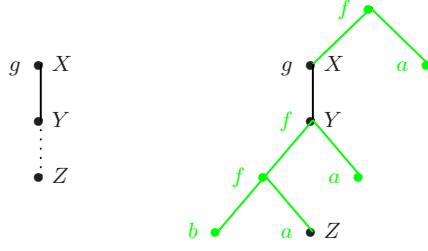


Fig. 2. The dominance constraint $X:g(Y) \wedge Y \triangleleft^* Z$ and of one of its solutions.

true in \mathcal{M}^τ iff $\pi, \pi' \in D_\tau$ and π is a prefix of π' . Finally, the *parallelism relation* $\pi_1/\pi'_1 \sim \pi_2/\pi'_2$ holds if the contexts $\tau_{\pi_1}^{\pi'_1}$ and $\tau_{\pi_2}^{\pi'_2}$ exist and coincide:

$$\pi_1/\pi'_1 \sim \pi_2/\pi'_2 \text{ holds in } \mathcal{M}^\tau \text{ iff } \pi_1 \triangleleft^* \pi'_1, \pi_2 \triangleleft^* \pi'_2, \text{ and } \tau_{\pi_1}^{\pi'_1} = \tau_{\pi_2}^{\pi'_2}.$$

Intuitively, this means the subtrees of τ below π_1 and π_2 have the same structure, except for the subtrees below π'_1 and π'_2 , which may be different.

4.2 The Constraint Language

We assume an infinite set of *node variables* X, Y, Z . A *parallelism constraint* φ is given by the following abstract syntax:

$$\varphi ::= X \triangleleft^* Y \mid X:f(X_1, \dots, X_n) \mid X/X' \sim Y/Y' \mid \varphi \wedge \varphi'$$

A parallelism constraint is a conjunction of atomic constraints for the dominance, labeling, and parallelism relations. A *dominance constraint* is parallelism constraint without atomic constraints $X/X' \sim Y/Y'$ for parallelism.

The semantics of parallelism constraints is given by interpretation over arbitrary tree structures \mathcal{M}^τ . A *solution* of a parallelism constraint φ consists of a tree structure \mathcal{M}^τ and a variable assignment α into its domain that satisfies all atomic constraints in φ . We write $(\mathcal{M}^\tau, \alpha) \models \varphi$ if $(\mathcal{M}^\tau, \alpha)$ is a solution of φ .

Note that the constraint $X:a \wedge Y:a$ has solutions where X and Y denote distinct nodes both of which are labeled with a .

We often display a dominance constraint and its solutions graphically. For instance, the constraint $X:g(Y) \wedge Y \triangleleft^* Z$ and one of its solutions are displayed in Fig. 2. Note that additional material (printed in light gray) has been filled into the space between the nodes denoted by Y and Z and above X . The dominance constraint does not say anything about these regions.

The careful reader might have noticed that dominance constraints can be expressed by parallelism constraints without atomic constraints for dominance.

Lemma 5. *The equivalence $X \triangleleft^* Y \leftrightarrow X/Y \sim X/Y$ is valid in all tree structures.*

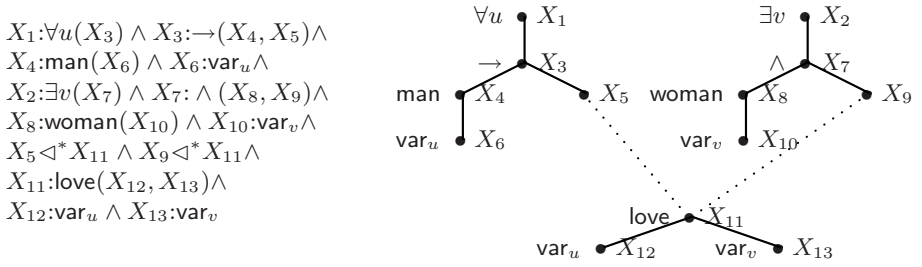


Fig. 3. An underspecified representation of the meaning of Example 3.

4.3 Application to Semantic Underspecification

As examples for the linguistic application of dominance and parallelism constraints, we briefly review a scope ambiguity and a very simple VP ellipsis. For the first example, consider the sentence (3), which is a classical scope ambiguity.

(3) Every man likes a woman.

The readings of this sentence can be represented by the predicate logic formulae in (4) and (5).

- (4) $\forall u.(man(u) \rightarrow \exists v.(woman(v) \wedge love(u, v)))$
 (5) $\exists v.(woman(v) \wedge \forall u.(man(u) \rightarrow love(u, v)))$

A compact underspecified representation of both readings is given by the dominance constraint in Fig. 3. The semantic representation of the sentence is considered as a tree, which is then described by a dominance constraint.

Ellipses can be modeled with parallelism constraints expressing that the trees corresponding to the semantics of source and target sentences must be the same except for the respective parallel elements. For instance, the semantics of (6) can be described by (7).

- (6) John sleeps. Mary does too.
 (7) $X: \text{sleep}(X') \wedge X': \text{john} \wedge Y': \text{mary} \wedge X/X' \sim Y/Y'$

We cannot go into this in more detail here and refer the reader to [10] for an in-depth discussion (in particular on the interaction of scope and ellipses).

5 Context Unification

Context unification is the problem of solving equations between tree valued terms in the two-sorted algebra \mathcal{TC} of trees and contexts. We first introduce equations between tree-valued terms and then show that they can also express equations between context-valued terms. Finally, we sketch an application to semantic underspecification.

$$\begin{array}{ll}
 x = g(y) & x \mapsto g(f(f(b, a), a)), \\
 \wedge y = C(z) & y \mapsto f(f(b, a), a), \\
 & C \mapsto f(f(b, \bullet), a), \\
 & z \mapsto a.
 \end{array}$$

Fig. 4. The equation system $x=g(y) \wedge y=C(z)$ and one of its solutions.

5.1 Syntax and Semantics of CU

The *algebra of trees and contexts* \mathcal{TC} over Σ is a two-sorted algebra whose domains are the set of trees and the set of contexts over Σ . The operations provided by \mathcal{TC} are tree construction and functional application of contexts to trees. Each function symbol $f \in \Sigma$ is interpreted as an $\text{ar}(f)$ -ary tree constructor, which maps a tuple (τ_1, \dots, τ_n) of trees to the tree $f(\tau_1, \dots, \tau_n)$. The application $\gamma[\tau]$ of a context γ to a tree τ has already been defined.

For both sorts of \mathcal{TC} , we assume an infinite set of variables: *tree variables* x, y, z and *context variables* C . A *tree-valued term* t is built from tree variables, applications of function symbols in Σ , and application of context variables.

$$t ::= x \mid f(t_1, \dots, t_n) \mid C(t) \quad (\text{ar}(f) = n)$$

In particular, every tree is a tree-valued term.

A *variable assignment* into \mathcal{TC} is a function β that assign trees to tree variables and contexts to context variables. Variable assignments can be lifted homomorphically to tree-valued terms:

$$\begin{aligned}
 \beta(f(t_1, \dots, t_n)) &= f(\beta(t_1), \dots, \beta(t_n)) \\
 \beta(C(t)) &= \beta(C)[\beta(t)].
 \end{aligned}$$

A variable assignment β into \mathcal{TC} is a *solution* of an equation system (i.e. a conjunction of equations between terms) if $\beta(t) = \beta(t')$ holds for all equations $t = t'$ in this system. *Context unification* is the problem of solving such equation systems over \mathcal{TC} . An example for a solution of the equation system $x=g(y) \wedge y=C(z)$ is given in Fig. 4. The similarity between Figures 2 and 4 is intended.

5.2 Properties of Contexts

The following three lemmas are quite simple, but will facilitate a lot of later work.

Lemma 6. *Two contexts γ and γ' are equal iff their holes are the same and there is a tree τ such that $\gamma[\tau] = \gamma'[\tau]$.*

Note that the existence of a single tree τ such that $\gamma[\tau] = \gamma'[\tau]$ is sufficient.

Proof. The “ \Rightarrow ” direction is trivial. For the other direction, all we have to prove is that the domains of the contexts γ and γ' are equal; this immediately implies equality of the labeling functions, since for every $\pi \in D_\gamma$ except for the (common) hole, $L_\gamma(\pi) = L_{\gamma[\tau]}(\pi) = L_{\gamma'[\tau]}(\pi) = L_{\gamma'}(\pi)$.

Let's say that the common hole of γ and γ' is π_0 . Then $\gamma[\tau] = \gamma'[\tau]$ implies the following equalities:

$$D_\gamma \cup \pi_0 \ D_\tau = D_{\gamma[\tau]} = D_{\gamma'[\tau]} = D_{\gamma'} \cup \pi_0 \ D_\tau.$$

As the unions on both sides are between sets whose respective intersection is $\{\pi_0\}$, it follows that $D_\gamma = D_{\gamma'}$. \square

We next express a correspondence between nodes and their contexts.

Lemma 7. *Let τ be a tree and π_1 a prefix of π_2 with $\pi_1, \pi_2 \in D_\tau$. Then $\tau_{\pi_2}^{\pi_1}$ is the unique context such that $\tau.\pi_1 = \tau_{\pi_2}^{\pi_1}[\tau.\pi_2]$.*

Proof. From Lemma 4 it follows that $\tau.\pi_1 = \tau_{\pi_2}^{\pi_1}[\tau.\pi_2]$. The uniqueness of $\tau_{\pi_2}^{\pi_1}$ follows from Lemma 6. \square

Lemma 8. *Let π_1 be a prefix of π_2 , π_2 a prefix of π_3 , and τ a tree whose domain contains π_1 , π_2 , and π_3 . Then $\tau_{\pi_2}^{\pi_1} \circ \tau_{\pi_3}^{\pi_2} = \tau_{\pi_3}^{\pi_1}$.*

Proof. Straightforward. \square

5.3 Equations between Context-Valued Terms

In the construction in the next section, it will be convenient to use equations between context-valued terms, such as $C = C_1 \circ C_2$. This notation emphasizes the functional character of contexts. In this section, we show that these equations can in fact be expressed by equations between tree-valued terms. A *context-valued term* u has the following abstract syntax:

$$u ::= C \mid \bullet \mid f(t_1, \dots, t_i, u, t_{i+1}, \dots, t_n) \mid u \circ u'$$

We conservatively extend \mathcal{TC} by concatenation $\gamma \circ \gamma'$ of contexts and lift variable assignments β to context-valued terms as follows. As above, we define that β is a *solution* of an equation $u = u'$ iff it maps u and u' to the same context.

$$\begin{aligned} \beta(\bullet) &= \bullet \\ \beta(f(t_1, \dots, u, \dots, t_n)) &= f(\beta(t_1), \dots, \beta(u), \dots, \beta(t_n)) \\ \beta(u \circ u') &= \beta(u) \circ \beta(u') \end{aligned}$$

Now we can define syntactic insertion $u[t]$ of tree-valued into context-valued terms in the obvious way. This produces tree-valued terms with the property $\beta(u[t]) = \beta(u)[\beta(t)]$. With this operation, we can express each equation between context-valued terms as a conjunction of equations between tree-valued terms.

Proposition 1 (Equations between context valued terms). *Let τ_1 and τ_2 be two different trees. Then the following equivalence holds:*

$$u=u' \quad \leftrightarrow \quad u[\tau_1]=u'[\tau_1] \wedge u[\tau_2]=u'[\tau_2].$$

Note that our restriction on the signature in Section 3 implies that two different trees really exist.

Proof. The direction from left to right is trivial. For the right-to-left direction, we show that the contexts γ and γ' denoted by u and u' must be equal. To this end, we only need to show that γ and γ' have the same hole; then their equality follows from Lemma 6.

Let's say that π and π' are the holes of γ and γ' , respectively. The path π cannot be a proper prefix of π' or vice versa. Otherwise, $\gamma[\tau_1] = \gamma'[\tau_1]$ would not be satisfied. Since $\pi' \in D_{\gamma'[\tau_1]}$, either $\pi' \in D_\gamma$, or π is a proper prefix of π' . But π is no prefix (proper or not) of π' , so $\pi' \in D_\gamma$. As π' and π are not a prefix of each other, it follows that $\gamma[\tau_1].\pi' = \gamma[\tau_2].\pi'$. Hence, by Lemma 3,

$$\begin{aligned} \gamma[\tau_1].\pi' &= \gamma'[\tau_1].\pi' = \tau_1 \\ \gamma[\tau_2].\pi' &= \gamma'[\tau_2].\pi' = \tau_2. \end{aligned}$$

So in contradiction to our assumptions, we have derived that $\tau_1 = \tau_2$. \square

5.4 Application to Semantic Underspecification

It is quite simple to express a scope ambiguity by using equations between context-valued terms. An underspecified representation of the meaning of Example 3 is given below.

$$\begin{aligned} x_\top &= C_1(\text{love}(\text{var}_u, \text{var}_v)) \\ C_1 &= C_2(\forall u(\rightarrow(\text{man}(\text{var}_u), C_3))) \\ C_1 &= C_4(\exists v(\wedge(\text{woman}(\text{var}_v), C_5))) \end{aligned}$$

The semantics of the whole sentence is represented by the tree denoted by x_\top in solutions of the above equations. The first equation states that the semantic description contains a description of the semantics of the verb *love*. The context of the verb semantics is denoted by C_1 . The second equation requires that a quantifier *every man* is placed within the context denoted by C_1 , i.e. above the verb. The third equation states that another quantifier *a woman* has also be placed above the verb.

6 Parallelism Constraints into Context Unification

In this section, we encode parallelism constraints (and thus dominance constraints) into context unification. More precisely, we show that for every parallelism constraint φ , there is an equation system $\llbracket \varphi \rrbracket$ in the language of context unification with the same solutions (up to a simple correspondence). We freely

$\llbracket X \triangleleft^* Y \rrbracket_p = \exists C (C_X \circ C = C_Y) \quad (C \text{ fresh})$
$\llbracket X : f(X_1, \dots, X_n) \rrbracket_p = \bigwedge_{1 \leq i \leq n} C_{X_i} = C_X \circ f(x_1, \dots, \bullet, \dots, x_n) \quad \text{if } (n \geq 1)$
$\llbracket X : a \rrbracket_p = x = a$
$\llbracket X / X' \sim Y / Y' \rrbracket_p = \exists C (C_{X'} = C_X \circ C \wedge C_{Y'} = C_Y \circ C) \quad (C \text{ fresh})$
$\llbracket \varphi_1 \wedge \varphi_2 \rrbracket_p = \llbracket \varphi_1 \rrbracket_p \wedge \llbracket \varphi_2 \rrbracket_p$

Fig. 5. Pre-encoding of dominance and parallelism constraints.

use equations between context-valued terms, which is safe according to Proposition 1.

We will proceed as follows: First, we define the encoding and consider some examples. Second, we lift the encoding to the first-order theory of parallelism constraints and prove its correctness.

For the proof, we will relate every solution $(\mathcal{M}^\tau, \alpha)$ of a parallelism constraint to a variable assignment $\llbracket \mathcal{M}^\tau, \alpha \rrbracket$ into \mathcal{TC} which solves the encoded constraint. With this terminology, the key result (Proposition 3) of our correctness proof (which makes the term “have the same solutions” precise) can be stated like this: For an arbitrary dominance constraint φ and its encoding $\llbracket \varphi \rrbracket$ as a CU equation system, the following equivalence holds.

$$(\mathcal{M}^\tau, \alpha) \models \varphi \Leftrightarrow \mathcal{TC}, \llbracket \mathcal{M}^\tau, \alpha \rrbracket \models \llbracket \varphi \rrbracket$$

As illustrated in Section 2, the main obstacle that we must overcome in our encoding of dominance constraints is to provide the power to talk about *occurrences* of subtrees. The central idea is to talk about nodes (occurrences of subtrees) by talking about their contexts. For instance, the two occurrences of a in the term $f(a, a)$ can be specified by the contexts represented by $f(a, \bullet)$ and $f(\bullet, a)$ respectively.

6.1 The Encoding

Let us define the encoding of a parallelism constraint φ . We associate with every variable X appearing in a φ a context variable C_X (whose purpose it is to denote the context starting at the root of the tree and whose hole is the node denoted by X) and a tree variable x (whose purpose it is to denote the tree below X). In addition, we introduce a new tree variable x_\top that we want to denote the entire tree. To ensure that these new variables interact correctly, we impose the following constraint, $\text{Root}(\varphi)$, where $\mathcal{FV}(\varphi)$ are the free variables of φ :

$$\text{Root}(\varphi) = \bigwedge_{X \in \mathcal{FV}(\varphi)} x_\top = C_X(x)$$

In addition, we define a pre-encoding $\llbracket \cdot \rrbracket_p$ as in Figure 5. The complete encoding $\llbracket \cdot \rrbracket$ is obtained as

$$\llbracket \varphi \rrbracket = \llbracket \varphi \rrbracket_p \wedge \text{Root}(\varphi).$$

An atomic dominance constraint $X \triangleleft^* X'$ is pre-encoded by $\exists C (C_{X'} = C_X \circ C)$, which expresses that the context of the node X can be enlarged by adding more material below its hole to obtain the context X' . An atomic parallelism constraint $X/X' \sim Y/Y'$ is pre-encoded by $\exists C (C_X \circ C = C_{X'} \wedge C_Y \circ C = C_{Y'})$, which expresses that the context of the node X can be enlarged to the context X' by adding the same material as for enlarging the context of Y to that of Y' . The pre-encoding of $X:f(X_1, \dots, X_n)$ requires for all $1 \leq i \leq n$ that the context above X_i is the context above X , enlarged with $f(x_1, \dots, \bullet, \dots, x_n)$, where the hole is at position i . For a nullary labeling constraint $X:a$, the pre-encoding requires $x = a$.

Proposition 2 (Encoding Parallelism Constraints). *A parallelism constraint φ is satisfiable iff its encoding $\text{Root}(\varphi) \wedge \llbracket \varphi \rrbracket_p$ is a satisfiable equation system of context unification.*

Proof. The proposition will be a simple consequence of Theorem 1, the analogous result for first-order formulae. \square

6.2 Examples

Before we turn to the first-order case, let us consider some examples. First, we reconsider Example (1) from Section 2. When we tried to encode this dominance constraint as a subtree constraint (2), we lost unsatisfiability. However, our new encoding works just fine. (8) shows the pre-encoding of the example; we have left the Root formula away, as it is not necessary for the unsatisfiability in this case.

- (1) $X : f(X_1, X_2) \wedge X_1 \triangleleft^* Y \wedge X_2 \triangleleft^* Y$
 (8) $C_{X_1} = C_X \circ f(\bullet, x_2) \wedge C_{X_2} = C_X \circ f(x_1, \bullet) \wedge C_{X_1} \circ C = C_Y \wedge C_{X_2} \circ C' = C_Y$

We can see that (8) is unsatisfiable in the following way. As $C_{X_1} \circ C = C_Y$ and $C_{X_2} \circ C' = C_Y$, $C_{X_1} \circ C = C_{X_2} \circ C'$. In this equation, we can substitute C_{X_1} by $C_X \circ f(\bullet, x_2)$ and C_{X_2} by $C_X \circ f(x_1, \bullet)$ and obtain $f(\bullet, x_2) \circ C = f(x_1, \bullet) \circ C'$, which is clearly unsatisfiable because the holes are different on both sides.

Another example will serve to show that the Root formula is really necessary to obtain the correct results. (10) is the (complete) encoding of the (unsatisfiable) dominance constraint (9) (a and b are different constants):

- (9) $X:a \wedge Y:b \wedge X \triangleleft^* Y$
 (10) $x_{\top} = C_X(x) \wedge x_{\top} = C_Y(y) \quad \wedge \quad x=a \wedge y=b \wedge C_X \circ C = C_Y$

The pre-encoding alone (i.e. the last three conjuncts) is satisfiable; together with the Root formula, it isn't. $x_{\top} = C_X(x) \wedge x_{\top} = C_Y(y)$ implies $C_X(x) = C_Y(y)$, which, when combined with $C_X \circ C = C_Y$, yields $x = C(y)$. When using $x=a \wedge y=b$ as a substitution, we obtain $a = C(b)$, which is not satisfiable.

$\llbracket \varphi \rrbracket$	$=$	$\llbracket \varphi \rrbracket_p$	$(\varphi \text{ atomic})$
$\llbracket \Phi_1 \wedge \Phi_2 \rrbracket$	$=$	$\llbracket \Phi_1 \rrbracket \wedge \llbracket \Phi_2 \rrbracket$	
$\llbracket \neg \Phi \rrbracket$	$=$	$\neg \llbracket \Phi \rrbracket$	
$\llbracket \exists X. \Phi \rrbracket$	$=$	$\exists C_X \exists x. (x \top = C_X(x) \wedge \llbracket \Phi \rrbracket)$	

Fig. 6. Encoding closed first-order formulas over parallelism constraints.

6.3 Encoding First-Order Formulae

In Fig. 6, the encoding of parallelism constraints is lifted to first-order formulae Φ . If we restrict ourselves to closed first-order formulae, an explicit Root formula is no longer needed; its components are distributed among the encodings of existential quantifiers. If we write $\exists\Phi$ for the existential closure of a formula Φ , then it holds for all dominance constraints φ that:

$$\exists (\text{Root}(\varphi) \wedge \llbracket \varphi \rrbracket_p) = \llbracket \exists\varphi \rrbracket$$

Hence, the correctness of the encoding $\text{Root}(\varphi) \wedge \llbracket \varphi \rrbracket_p$ claimed in Prop. 2 follows from the correctness of the encoding of first-order sentences.

Now let us turn to the proof of the first-order case. First, we formulate the correspondence $\llbracket \cdot, \cdot \rrbracket_V$ we announced above. This function maps pairs of tree structures \mathcal{M}^τ and variable assignments mapping the variables in V to the domain of τ to variable assignments into \mathcal{TC} . The goal is that if the arguments satisfy a given dominance constraint, the result will satisfy its encoding.

$$\begin{aligned} \llbracket \mathcal{M}^\tau, \alpha \rrbracket_V(x \top) &= \tau \\ \llbracket \mathcal{M}^\tau, \alpha \rrbracket_V(x) &= \tau.\alpha(X) \quad \text{for all } x \text{ such that } X \in V \\ \llbracket \mathcal{M}^\tau, \alpha \rrbracket_V(C_X) &= \tau_{\alpha(X)}^e \quad \text{for all } C_X \text{ such that } X \in V. \end{aligned}$$

With this definition, the following proposition holds.

Proposition 3. *Let \mathcal{M}^τ be a tree structure, α a variable assignment, and Φ a first-order formula over the parallelism constraints. Then Φ is satisfied by $(\mathcal{M}^\tau, \alpha)$ iff $\llbracket \Phi \rrbracket$ is satisfied by $\llbracket \mathcal{M}^\tau, \alpha \rrbracket_{\mathcal{FV}(\Phi)}$.*

Proof. We prove the proposition by structural induction. First, we show that it is true for the atomic constraints; towards the end of the proof, we conduct the induction steps. Throughout the proof, we write $\beta = \llbracket \mathcal{M}^\tau, \alpha \rrbracket_{\mathcal{FV}(\Phi)}$ for brevity.

– $X \triangleleft^* Y$. The treatment of $X/X' \sim Y/Y'$ is analogous.

“ \Rightarrow ” Assume that $(\mathcal{M}^\tau, \alpha)$ satisfies $X \triangleleft^* Y$; we show that β satisfies the encoding $\exists C(C_X \circ C = C_Y)$. Our assumption means that $\alpha(X)$ is a prefix of $\alpha(Y)$. Hence, we can construct a variable assignment β' that is like β , but assigns $\tau_{\alpha(Y)}^{\alpha(X)}$ to C . By Lemma 8, β' solves $C_X \circ C = C_Y$ and, thus, β is a solution of $\llbracket X \triangleleft^* Y \rrbracket$.

- “ \Leftarrow ” Assume that $\exists C(C_X \circ C = C_Y)$ is satisfied by β . Then there must be a context γ such that $\beta(C_X) \circ \gamma = \beta(C_Y)$; hence, $\alpha(X)$ must be a prefix of $\alpha(Y)$, and $(\mathcal{M}^\tau, \alpha)$ satisfies $X \triangleleft^* Y$.
- $X:f(X_1, \dots, X_n)$, where $n \geq 1$
- “ \Rightarrow ” Assume that $(\mathcal{M}^\tau, \alpha)$ satisfies $X:f(X_1, \dots, X_n)$; we assume $1 \leq i \leq n$ and conclude that β satisfies all equations $C_{X_i} = C_X \circ f(x_1, \dots, \bullet, \dots, x_n)$ where the hole marker \bullet is at position i .
- Let u be the context-valued term $C_X \circ f(x_1, \dots, \bullet, \dots, x_n)$. We first show that the holes of $\beta(u)$ and $\beta(C_{X_i})$ are the same, and then that their values on $\beta(x_i)$ are equal. (Here we need $n \geq 1$, as x_i would not exist otherwise.) From Lemma 6, we can then conclude $\beta(u) = \beta(C_{X_i})$. The hole of $\beta(C_{X_i}) = \tau_{\alpha(X_i)}^\epsilon$ is $\alpha(X_i)$, and that of $\beta(u)$ is $\alpha(X)i$. Since $(\mathcal{M}^\tau, \alpha)$ is a solution of $X:f(X_1, \dots, X_n)$, we have $\alpha(X)i = \alpha(X_i)$, and hence the holes are equal.
- We already noticed that $\alpha(X)i = \alpha(X_i)$ for all $1 \leq i \leq n$. Lemma 2 implies that

$$\begin{aligned} \beta(x) &= \tau.\alpha(X) = f(\tau.\alpha(X)1, \dots, \tau.\alpha(X)n) \\ &= f(\tau.\alpha(X_1), \dots, \tau.\alpha(X_n)) \\ &= f(\beta(x_1), \dots, \beta(x_n)) \end{aligned}$$

Based on this equation and Lemma 7, we are now in the position to prove $\beta(u)(\beta(x_i)) = \beta(C_{X_i})(\beta(x_i))$ (and thus $\beta(u) = \beta(C_{X_i})$ as required):

$$\begin{aligned} \beta(u)(\beta(x_i)) &= \tau_{\alpha(X)}^\epsilon[f(\beta(x_1), \dots, \beta(x_n))] = \tau_{\alpha(X)}^\epsilon[\beta(x)] = \tau \\ \beta(C_{X_i})(\beta(x_i)) &= \tau_{\alpha(X_i)}^\epsilon[\beta(x_i)] = \tau \end{aligned}$$

- “ \Leftarrow ” Assume that β solves the equation $C_X = C_{X_i} \circ f(x_1, \dots, \bullet, \dots, x_n)$ for some $1 \leq i \leq n$, where the hole \bullet is at position i . Lemma 7 yields

$$\begin{aligned} \tau &= \tau_{\alpha(X)}^\epsilon[\tau.\alpha(X)] = \beta(C_X)[\beta(x)] \\ \tau &= \tau_{\alpha(X_i)}^\epsilon[\tau.\alpha(X_i)] = \beta(C_{X_i})[\beta(x_i)] = \beta(C_X)[f(\beta(x_1), \dots, \beta(x_n))] \end{aligned}$$

Since context functions are one-to-one and $\beta(C_X)$ is a context function, these equations imply $\beta(X) = f(\beta(x_1), \dots, \beta(x_n))$. This is equivalent to

$$\tau.\alpha(X) = f(\tau.\alpha(X_1), \dots, \tau.\alpha(X_n)),$$

which in turn means that $(\mathcal{M}^\tau, \alpha)$ solves $X:f(X_1, \dots, X_n)$.

- $X:a$
- “ \Rightarrow ” Assume that $(\mathcal{M}^\tau, \alpha)$ satisfies $X:a$. Since $\text{ar}(a) = 0$, it follows that $\tau.\alpha(X) = a$, so β solves $x=a$.
- “ \Leftarrow ” Assume that β satisfies $x=a$; then $\beta(x) = \tau.\alpha(X) = a$ and, hence, $(\mathcal{M}^\tau, \alpha)$ solves $X:a$.

Of the complex cases, negation and conjunction are trivial. Existential quantification is more interesting:

– $\exists X.\Phi$

“ \Rightarrow ” We assume that $(\mathcal{M}^\tau, \alpha)$ satisfies $\exists X.\Phi$; so there is a path π such that $(\mathcal{M}^\tau, \alpha[\pi/X])$ solves Φ . By induction hypothesis, $\llbracket \mathcal{M}^\tau, \alpha[\pi/X] \rrbracket_{\mathcal{FV}(\Phi)}$ satisfies $\llbracket \Phi \rrbracket$. On the free variables of Φ , this variable assignment agrees with $\llbracket \mathcal{M}^\tau, \alpha \rrbracket_{\mathcal{FV}(\exists X\Phi)}[\tau.\pi/X, \tau_\pi^\epsilon/C_X]$, and the latter variable assignment satisfies $x_\top = C_X(x)$ as well. Thus $\llbracket \mathcal{M}^\tau, \alpha \rrbracket_{\mathcal{FV}(\exists X\Phi)}$ solves $\llbracket \exists X\Phi \rrbracket$.

“ \Leftarrow ” We assume that $\beta = \llbracket \mathcal{M}^\tau, \alpha \rrbracket_{\mathcal{FV}(\exists X\Phi)}$ solves $\exists C_X \exists x (\llbracket \Phi \rrbracket \wedge x_\top = C_X(x))$. There is a π such that $\beta[\tau.\pi/x, \tau_\pi^\epsilon]$ solves $\llbracket \Phi \rrbracket$. Since $\beta[\tau.\pi/x, \tau_\pi^\epsilon/C_X]$ is equal to $\llbracket \mathcal{M}^\tau, \alpha[\pi/X] \rrbracket_{\mathcal{FV}(\Phi)}$ on all free variables of $\llbracket \Phi \rrbracket$, it follows from the induction hypothesis that $(\mathcal{M}^\tau, \alpha[\pi/X])$ solves Φ . Hence $(\mathcal{M}^\tau, \alpha)$ solves $\exists X\Phi$. \square

Corollary 1 (Encoding First-Order Formulae). *A closed first-order formula Φ over dominance and parallelism constraints is satisfied by a pair $(\mathcal{M}^\tau, \alpha)$ iff there is a variable assignment β into \mathcal{TC} that solves $\llbracket \Phi \rrbracket$ such that $\beta(x_\top) = \tau$.*

7 Context Unification into Parallelism Constraints

We finally show how to express equations of context unification by parallelism constraints. This is not obvious but it follows from a result of [24] which shows that CU has the same expressive power as *equality up-to constraints*. Equality up-to constraints can be translated to parallelism constraints plus *similarity constraints*. Finally, one can get rid of similarity constraints by a neat trick.

An equality up-to constraint is a conjunction of atomic constraints of the following form, which are interpreted in the algebra \mathcal{TC} .

$$\psi ::= x/x' = y/y' \mid x = f(x_1, \dots, x_n) \mid \psi \wedge \psi'$$

An atomic equality up-to constraint $x/x' = y/y'$ is satisfied by a variable assignment β into \mathcal{TC} if there is a context γ such that $\beta(x) = \gamma[\beta(x')]$ and $\beta(y) = \gamma[\beta(y')]$. Intuitively, this is the case iff the trees denoted by x and y are *equal, up to* an occurrence of x' in x and of y' in y respectively. Equality up-to constraints are equivalent to context unification:

Proposition 4 (Equality up-to Constraints and CU [24]). *For every equation system of context unification, there is a satisfaction equivalent equality up-to constraint, and vice versa.*

With this result, it remains to encode equality up-to constraints into parallelism constraints. This would be simple if parallelism constraints could express similarity constraints. A similarity constraint has the form $X \sim Y$ and is interpreted by the *similarity relation*. A similarity relationship $\pi \sim \pi'$ holds for two nodes if $\tau.\pi = \tau.\pi'$ (i.e. if the subtrees below π and π' are the same).

Lemma 9. *If the signature Σ contains a single constant, then parallelism constraints can express similarity constraints.*

$\llbracket x/x'=y/y' \rrbracket^{-1} = X/X'' \sim Y/Y'' \wedge X' \sim X'' \wedge Y' \sim Y'' \quad (X'', Y'' \text{ fresh})$
$\llbracket x=f(x_1, \dots, x_n) \rrbracket^{-1} = X:f(X'_1, \dots, X'_n) \wedge \bigwedge_{i=1}^n X_i \sim X'_i \quad (X'_1, \dots, X'_n \text{ fresh})$

Fig. 7. Encoding equality up-to into parallelism and similarity constraints.

Proof. Let a be the unique constant of Σ . Every finite tree must contain a node labeled with a ; so the following equivalence holds for all tree models:

$$X \sim Y \leftrightarrow \exists Z \exists Z' (Z:a \wedge Z':a \wedge X/Z \sim Y/Z') \quad \square$$

If the number of constants in Σ is finite, we can express $X \sim Y$ by a finite disjunction; but this would not lead to a polynomial time transformation. But there is a neat trick to work around which even applies for infinitely many constants.

Lemma 10. *For every signature Σ , there exists a signature Σ' with a single constant such that parallelism and similarity constraints over Σ can be translated in linear time into satisfiability equivalent constraints of the same kind over Σ' .*

Proof. For any signature Σ , let Σ' be the signature consisting of all non-constant symbols of Σ , plus the constants of Σ considered as unary function symbols, plus a new constant a . We transform each parallelism constraint φ into a constraint φ' by replacing every constraint $X:b$ by $\exists Y (X:b(Y) \wedge Y:a)$. Now it is easy to see that φ is satisfiable over Σ iff φ' is satisfiable over Σ' . \square

Theorem 1 (Parallelism Constraints = Context Unification). *For every parallelism constraint φ , there is a satisfiability equivalent equation system of context unification, and vice versa.*

Proof. The correctness of an encoding of parallelism constraints into CU is stated in Proposition 2.

For the converse, we first express CU by equality up-to constraints according to Proposition 4. Second, we encode equality up-to constraints by parallelism and similarity constraints. This is quite easy; an encoding $\llbracket \psi \rrbracket^{-1}$ is defined in Figure 7. In order to encode ψ , we assume a node variable X for every tree variable x occurring in ψ . The variable X is supposed to denote the root node of an occurrence of x in the solution of the encoding of φ . It is obvious that $\llbracket \cdot \rrbracket^{-1}$ preserves satisfiability. The encoding of $x/x'=y/y'$ expresses that somewhere below the nodes X and Y , there are nodes X'' and Y'' the trees below which look just like the trees below the nodes X' and Y' , and the contexts between X and X'' and Y and Y'' are equal. (Note that this is a weaker condition than parallelism itself; it does not say anything about the locations of the nodes denoted by X' and Y' .) The encoding of equation $x=f(x_1, \dots, x_n)$ works similarly: It expresses that X is labeled with f and that its subtrees look just like the subtrees below the X_1, \dots, X_n .

Third, we switch to a signature with a single constant which we can do according to Lemma 10. We can now express all similarity constraints by parallelism constraints (Lemma 9) which completes the proof. \square

8 Conclusion

The main result of this paper is that context unification has the same expressive power as parallelism constraints. Parallelism constraints subsume dominance constraints. The most involved part was to embed dominance constraints into CU. The inverse direction from CU to parallelism constraints proceeds via a deviation through equality up-to constraints, which have the same expressiveness as CU as well.

The correspondence between CU and CLLS has two important consequences. For one, it allows us to transfer complexity and decidability results. For the time being, however, the decidability of either language is unknown. Conversely, the satisfiability problem of dominance constraints is shown NP-complete in [15]. Of course, NP-hardness for several fragments of CU was well known before.

The other consequence is that CU can be easily expressed by parallelism constraints in CLLS [10] which explains why the linguistic application given for CU in [25] carries over to CLLS. Furthermore, this application of CU is clarified. In earlier papers, scope ambiguities could be described in CU but only in a somewhat intransparent fashion. In the light of the results presented, it becomes clear that the equations used previously were really just encodings of dominance and parallelism constraints.

Acknowledgments. We are deeply indebted to Peter Ruhrberg, a former colleague of ours who conjectured the presented relationship long before CLLS was found. It is a pleasure to thank all members (student or not) of the CHORUS project. The research reported here was supported by the SFB 378 at the Universität des Saarlandes and the Esprit Working Group CCL II (EP 22457).

References

1. Aiken, A., and E. L. Wimmers. Solving Systems of Set Constraints. In *International Conference on Logic in Computer Science*, pages 329–340, June 1992.
2. Baader, F., and J. Siekmann. Unification theory. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford University Press, 1993.
3. Backofen, R., and G. Smolka. A complete and recursive feature theory. *Journal of Theoretical Computer Science*, 146(1–2):243–268, July 1995.
4. Blackburn, P., C. Gardent, and W. Meyer-Viol. Talking about trees. In *European Chapter of the Association of Comp. Linguistics*, 1993.
5. Bos, J. Predicate logic unplugged. In *Proc. of the 10th Amsterdam Colloquium*, pages 133–143, 1996.
6. Colmerauer, A. Equations and inequations on finite and infinite trees. In *Proceedings of the 2nd International Conference on Fifth Generation Computer Systems*, pages 85–99, 1984.

7. Comon, H. Completion of rewrite systems with membership constraints. In *Coll. on Automata, Languages and Programming*, volume 623 of *LNCS*, 1992.
8. Deemter, K. van, and S. Peters. *Semantic Ambiguity and Underspecification*. CSLI, Stanford, 1996.
9. Duchier, D. and C. Gardent. A constraint-based treatment of descriptions. In *Proceedings of IWCS-3*, Tilburg, 1999.
10. Egg, M., J. Niehren, P. Ruhrberg, and F. Xu. Constraints over lambda-structures in semantic underspecification. In *Proc. of COLING/ACL*, pages 253–359, 1998.
11. Gardent, C., and B. Webber. Describing discourse semantics. In *Proc. of the TAG+ Workshop*, Philadelphia, 1998.
12. Heintze, N., and J. Jaffar. A decision procedure for a class of set constraints. In *International Conference on Logic in Computer Science*, pages 42–51, 1990.
13. Kasper, R. T., and W. C. Rounds. A logical semantics for feature structures. In *Annual Meeting of the Association of Comp. Linguistics*, pages 257–265, 1986.
14. Koller, A. *Constraint languages for semantic underspecification*. Master's thesis, Universität des Saarlandes, Saarbrücken, 1999.
<http://www.coli.uni-sb.de/~koller/papers/da.html>.
15. Koller, A., J. Niehren, and R. Treinen. Dominance constraints: Algorithms and complexity. In *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics*, Grenoble, 1998.
16. Lévy, J. Linear second order unification. In *International Conference on Rewriting Techniques and Applications*, volume 1103 of *LNCS*, pages 332–346, 1996.
17. Makanin, G. S. The problem of solvability of equations in a free semigroup. *Soviet Akad. Nauk SSSR*, 223(2), 1977.
18. Marcus, M. P., D. Hindle, and M. M. Fleck. D-theory: Talking about talking about trees. In *Proc. of the Annual Meeting of the ACL*, 1983.
19. Martelli, A. and U. Montanari. An efficient unification algorithm. *ACM TOPLAS*, 4(2):258–282, 1982.
20. Meyer-Viol, W., and R. Kempson. Sequential construction of logical forms. In *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics*, Grenoble, France, 1998.
21. Müller, M., and J. Niehren. Ordering constraints over feature trees expressed in second-order monadic logic. *Information and Computation*, 2000. Special Issue on RTA, Tsukuba, Japan, March 1998. To appear.
22. Mozart Consortium: Universität des Saarlandes, DFKI, SFB 378, SICS, Université de Louvain. The Mozart System of Oz. Freely available at www.mozart-oz.org, January 1999.
23. Muskens, R. Underspecified semantics. CLAUS Report 95, Univ. des Saarlandes, Saarbrücken, 1998.
24. Niehren, J., M. Pinkal, and P. Ruhrberg. On equality up-to constraints over finite trees, context unification and one-step rewriting. In *Proc. of the CADE*, volume 1249 of *LNCS*, pages 34–48, 1997.
25. Niehren, J., M. Pinkal, and P. Ruhrberg. A uniform approach to underspecification and parallelism. In *Annual Meeting of the Association of Comp. Linguistics*, pages 410–417, 1997.
26. Niehren, J., R. Treinen, and S. Tison. On rewrite constraints and context unification. Technical report, Universität des Saarlandes, Programming Systems Lab, 1999. Submitted. Available at
<http://www.ps.uni-sb.de/Papers/abstracts/rewrite-context>.
27. Rabin, M. O. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

28. Reyle, U. Dealing with ambiguities by underspecification: construction, representation, and deduction. *Journal of Semantics*, 10:123–179, 1993.
29. Decidability of context unification. The RTA list of open problems, number 90, <http://www.lri.fr/~rtaloop/>, 1998.
30. Schmidt-Schauß, M. Unification of stratified second-order terms. Technical Report 12/94, J. W. Goethe Universität, Frankfurt, 1994.
31. Schmidt-Schauß, M. A unification algorithm for distributivity and a multiplicative unit. *J. of Symbolic Computation*, 22(3):315–344, 1997.
32. Schmidt-Schauß, M., and K. Schulz. On the exponent of periodicity of minimal solutions of context equations. In *International Conference on Rewriting Techniques and Applications*, volume 1379 of *LNCS*, 1998.
33. Schmidt-Schauß, M., and K. Schulz. Solvability of context equations with two context variables is decidable. In *Proc. of the CADE*, LNCS, 1999.
34. Schulz, K. U. Word unification and transformation of generalized equations. *Information and Computation*, 11:149–184, 1993.
35. Smolka, G. Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.
36. Smolka, G. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000, pages 324–343. Springer-Verlag, 1995.
37. Talbot, J.-M., Ph. Devienne, and S. Tison. Generalized definite set constraints. *Constraints, an International Journal, Special Issue on CP'97*, January 2000.
38. Venkataraman, K. N. Decidability of the purely existential fragment of the theory of term algebra. *Journal of the ACM*, 34(2):492–510, 1987.
39. Vijay-Shanker, K. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18(4):481–517, 1992.

A Revision System of Circular Objects and Its Applications to Dynamic Semantics of Dialogues

Norihiro Ogata*

Osaka University
Faculty of Language and Culture
ogata@lisa.lang.osaka-u.ac.jp,

Abstract. Since Peter Aczel's theory[1] of *hypersets*, many applications to formalizations of a *circular object* such as a mutual belief has been proposed [3,2,5]. This paper will propose *Membership Description Systems* (MDSs), a partial revision system of circular objects and their applications to a dynamic semantics of a dialogue in the sense that a dialogue can be considered as a revision process of mutual beliefs between its agents. Although usual dynamic semantics [10] updates a variable assignment or a set of information states, our proposal of a semantics of dialogues directly updates *situations*, which is specified by MDSs, as dynamic semantics of *circular propositions* [11] directly updates situations. Furthermore, using MDSs as updated objects in the semantics makes a partial and direct update of circular situations themselves possible. As a result, a dynamic semantics of a language with the \downarrow -operator, which is introduced by [3] to describe circular propositions, can be provided.

1 Introduction

This paper will propose *Membership Description Systems* (MDSs) a formal system of revision of some types of an infinitely structured object, called a *circular object*, which can be a formal model of *liar sentences* [3], mutual beliefs, shared information, and common knowledge [2]. They can be a basis of dynamic semantics of a dialogue in the sense that a dialogue can be considered as a revision process of mutual beliefs between its agents.

MDSs are theoretically based on the framework of circular objects proposed in [3,2,5], which exploits *hyperset theory* [1,5] in which we can define circular sets as a unique solution to *equational system* of sets. Although usual dynamic semantics [10,4] updates variable assignments or states as a set of possible variable assignments, a semantics based on MDSs updates *situations* directly as dynamic semantics [11], and furthermore *circular situations* are also revised directly in the semantics.

Section 2 is a brief introduction of some basic concepts about circular objects and hyperset theory. Section 3 is the definition of a language for description

* This research is supported by the Grant of the Japan Society of Promotion for science. I am grateful to Makoto Kanazawa, Nigel Collier and the reviewer for their helpful advice and comments on the previous research of this paper.

of mutual beliefs and its two types of dynamic semantics which cannot treat direct update of circular situations. Section 4 is the definition of MDSs, their relevant concepts and a semantics based on MDSs which can treat direct update of circular situations.

2 Theory of Circular Objects Based on Hyperset Theory

2.1 Circular Objects and Their Revisions

We can find some types of infinite objects as follows.

- (1) a. $f(f(f(f(\dots))))$
 b. $f(a, f(a, f(a, f(a, \dots))))$
 c. $f(a, f(f(a), f(f(f(a))), f(f(f(f(a))), \dots)))$
 d. $f(a, f(f(f(a)), f(f(f(g(a)), f(a)), \dots)))$

(1a)-(1c) can be recognized as *regular*, since we can find some regularity to predict further detailed structures of them, while (1d) can not be recognized as regular from this sequence. Furthermore, (1a-1b) are simple repetitions, while (1c) is not a simple repetition. We say that an object is a *circular object*¹ if it has simple repetitive structure as (1b). For example, (2), namely a *mutual belief* between Max and Claire, are circular objects which we can find in the real world.

- (2) Max believes that Max has the ace of hearts,
 Claire believes that Max has the ace of hearts,
 Max believes that Max believes that Max has the ace of hearts and that
 Claire believes that Max has the ace of hearts,
 Claire believes that Max believes that Max has the ace of hearts and that
 Claire believes that Max has the ace of hearts, and ...

We mean by *revisions of circular objects* such a change from (1a) to (1b), or from (1b) to (1a). Therefore, mutual belief revisions by dialogues are also considered as revisions of circular objects.

2.2 Hyperset Theory

The term ‘hyperset theory’ is informal. We mean by the term a set theory which has the *Anti-Foundation Axiom (AFA)* instead of the Foundation Axiom. *AFA* has many formalizations [1,5], for example:

- (i) Every flat systems of equations (X, A, e) has a unique solution;
- (ii) Every graph G over A has a unique decoration;
- (iii) For every proper substitution e there is a unique proper substitution s such that $s = s \star e$.

The most remarkable characteristic of hyperset theory is that hyperset theory admits *circular sets* such as $x \in x$.

This paper concentrates on the formalization using *equational system of sets* which will be introduced in the next subsection, in detail.

¹ [6] call it as a *regular tree*.

2.3 Equational Systems of Sets

A circular set such as $x = \{x, a\}$ is specified by the notion of *equational systems of sets*, defined as follows.

Definition 1 (Barwise & Moss [5]).

1. A flat equational system of sets is a triple $\mathcal{E} = (X, A, e)$, where X and A are sets of urelements such that $X \cap A = \emptyset$, and a function $e : X \rightarrow \text{pow}(X \cup A)$.
2. X is called the set of indeterminates of \mathcal{E} , and A is called the set of atoms of \mathcal{E} .
3. A solution to \mathcal{E} is a function θ with domain X satisfying $\theta(x) = \{\theta(y) | y \in e(x) \cap X\} \cup (e(x) \cap A)$, for each $x \in X$.

If e is a function from indeterminates X to $\text{pow}(A \cup X)$, then the system is called *flat*, while if e is a function from indeterminates X to any set constructed from A and X basically, the system is called *general*. For example, $\{x = (a, x)\}$ is general, but one of its equivalents $\{x = \{y, z\}, y = \{a\}, z = \{a, x\}\}$ is flat.

Using the concept of flat equational systems of sets, we can state a form of Anti-Foundation Axiom as follows:

ANTI-FOUNDATION AXIOM: Every flat equational system of sets has a unique solution θ .

Let *solution – set*(\mathcal{E}) be the set $\{\theta(x) | x \in X\}$ where $\mathcal{E} = (X, A, e)$. We can define the *hyperuniverse* $V_{afa}[\mathcal{U}]$ as follows.

$V_{afa}[\mathcal{U}] = \bigcup \{\text{solution – set}(\mathcal{E}) | \mathcal{E} \text{ is a flat equational system of sets with atoms } A \subseteq \mathcal{U}\}.$

3 A Language of Mutual Belief and Its Dynamic Semantics

Now we define a minimal language \mathfrak{L} in order to describe mutual beliefs, consisting of each sentence φ defined as follows.

$$\varphi ::= \text{has}(\mathbf{a}, \mathbf{c}) | \text{Bel}(\mathbf{a}, \varphi) | \text{Bel}(\mathbf{a}, v \wedge \varphi) | \varphi_1 \wedge \varphi_2 | \downarrow v \varphi,$$

where $c \in C = \{2\clubsuit, \dots, \mathbf{A}\spadesuit\}$ (a set of card symbols), $v \in \text{Var}$ (a set of sentence variables), and $a \in \text{AG} = \{\text{max}, \text{claire}\}$ (a set of agent symbols). $\downarrow v \varphi^2$ means that it is a unique solution to $v = \varphi$, i.e., the \downarrow -operator indicates the scope of its circularity.

Example 1. (2) amounts to sentence:

$(*) \downarrow v. \text{Bel}(\mathbf{Max}, v \wedge \text{has}(\mathbf{Max}, \mathbf{A}\heartsuit) \wedge \text{Bel}(\mathbf{Claire}, v \wedge \text{has}(\mathbf{Max}, \mathbf{A}\heartsuit))).$

² This is basically the same with the scope indicator of self-referential terms in the language of [3].

As a model of \mathfrak{L} , we define class *SOA* of states of affairs and class *SIT* of situations.

Definition 2. Assume the Anti-Foundation Axiom. Let $H = \{((Has, a, c), 1) | a \in A = \{Claire, Max\}, c \in \{2\clubsuit, \dots, A\spadesuit\} \text{ be a set of states of affaires, and } \Phi \text{ be a functor such that } \Phi(X) = \{((Bel, a, s), 1) | a \in A, s \subseteq X\} \cup H. SOA \text{ is the greatest fixed point of } \Phi. \text{ If } s \subseteq SOA, \text{ then } s \in SIT.$

Φ has its greatest fixed point, since Φ is obviously monotone and the Knaster-Tarski Theorem assures the existence of the greatest fixed point and the least fixed point of monotone operators taking a complete lattice to a complete lattice.

Theorem 1 (the Knaster-Tarski Theorem [7,12]). Let $\mathfrak{L} = (L, \subseteq)$ be a complete lattice and $F : L \rightarrow L$ be a monotone function. Then F has the least fixed point $lfg(F)$ and the greatest fixed point $gfp(F)$. Furthermore, $lfp(F) = \bigcap \{X | F(X) \subseteq X\}$ and $gfp(F) = \bigcup \{X | X \subseteq F(X)\}$.

We can model a mutual belief between Max and Claire (*) as $b = \{p, q\}$, where $p = (Bel, Claire, b \cup \{(Has, Max, A\heartsuit; 1)\}; 1)$ and $q = (Bel, Max, b \cup \{(Has, Max, A\heartsuit; 1)\}; 1)$, according to Fagin et al.[8] and Barwise [2]³. We will write $(\sigma; 1)$ for $((\sigma), 1)$.

Proposition 1. The mutual belief b defined in the above is a member of *SIT*.

Proof. We have only to show $p, q \in SOA$. Let $X = SOA \cup \{p, q\}$. Then $\Phi(X) = \{(Bel, a, s; 1) | a \in A, s \subseteq SOA \cup \{p, q\}\} \cup H$ which includes $\{p, q\}$ i.e., $\{(Bel, Claire, \{p, q, (Has, Max, A\heartsuit; 1)\}; 1), (Bel, Max, \{p, q, (Has, Max, A\heartsuit; 1)\}; 1)\}$. That is,

$$\{p, q\} \subseteq \Phi(X) \quad (1)$$

Since SOA is a fixed point of Φ , $\Phi(SOA) = SOA$. Then by monotonicity, $\Phi(SOA) \subseteq \Phi(X)$. So $SOA \subseteq \Phi(X)$. Hence, by (1), $SOA \cup \{p, q\} \subseteq \Phi(SOA \cup \{p, q\})$. But then by the property of greatest fixed points, for any X , if $X \subseteq \Phi(X)$, then $X \subseteq SOA$, from which it follows that $p, q \in SOA$. \square

3.1 Dynamic Scope-Taking

As the basic property of dynamic semantics, it admits *dynamic scope-taking* of an operator of the object language. In *Dynamic Predicate Logic* (DPL) [9], the following equivalence is hold:

$$(\exists v. \varphi_1) \wedge \varphi_2 \equiv (\exists v. \varphi_1 \wedge \varphi_2),$$

since DPL's semantics is defined as follows: for a set of assignments s ,

- $\llbracket \pi(t_1, \dots, t_n) \rrbracket(s) = s \cap \{g | \mathcal{M} \models \pi(t_1, \dots, t_n)[g]\},$
- $\llbracket \varphi_1 \wedge \varphi_2 \rrbracket = \llbracket \varphi_1 \rrbracket \circ \llbracket \varphi_2 \rrbracket,$

³ This modeling is classified to the Fixedpoint approach in Barwise's term.

– $\llbracket \exists v \varphi \rrbracket = [\sim_x] \circ \llbracket \varphi \rrbracket$, where $[\sim_x](s) = \{h | g \sim_x h, g \in s\}$.

We will define the notion of dynamic scope-taking as follows.

Definition 3. *An unary operator O takes its scope dynamically w.r.t. a binary operator P if $P(O(x_1), x_2) = O(P(x_1, x_2))$.*

Corollary 1. *In DPL, $\exists v$ takes its scope dynamically w.r.t. \wedge .*

3.2 \mathfrak{S}_1 : A Semantics of \mathfrak{L} Based on \cup -Based Updates

Exploiting the basic idea of [11]⁴, we tentatively propose a dynamic semantics of \mathfrak{L} based on \cup -based update of information states.

Definition 4. *For each $\varphi \in \mathfrak{L}$, an update function $\llbracket \varphi \rrbracket : SIT \rightarrow SIT$ is assigned in \mathfrak{S}_1 , where $\llbracket \varphi \rrbracket$ is defined by induction of the complexity of φ as follows.*

- $\llbracket has(\mathbf{a}, \mathbf{c}) \rrbracket(s) = s \cup \{(Has, a, c; 1)\}$,
- $\llbracket Bel(\mathbf{a}, \varphi) \rrbracket(s) = s \cup \{(Bel, a, \llbracket \varphi \rrbracket; 1)\}$;
- $\llbracket Bel(\mathbf{a}, v \wedge \varphi) \rrbracket(s) = s \cup \{(Bel, a, \mathbf{v} \cup \llbracket \varphi \rrbracket; 1)\}$;
- $\llbracket \varphi_1 \wedge \varphi_2 \rrbracket = \llbracket \varphi_1 \rrbracket \circ \llbracket \varphi_2 \rrbracket$ (sequential composition of functions);
- $\llbracket \downarrow v \varphi \rrbracket(s) = \mathbf{v}$ where $\mathbf{v} = \llbracket \varphi \rrbracket(s)$ and \mathbf{v} is an indeterminate.

Example 2. To sentence 1, an update function G is assigned in \mathfrak{S}_1 , where for some $s \in SIT$ and $G(s)$ is defined as follows:

$$\begin{aligned} G(s) &= \llbracket \downarrow v. Bel(\mathbf{Max}, v \wedge has(\mathbf{Max}, \mathbf{A} \heartsuit) \wedge Bel(\mathbf{Claire}, v \wedge has(\mathbf{Max}, \mathbf{A} \heartsuit))) \rrbracket(s) \\ &= \mathbf{v}, \end{aligned}$$

where

$$\begin{aligned} \mathbf{v} &= \llbracket Bel(\mathbf{Max}, v \wedge has(\mathbf{Max}, \mathbf{A} \heartsuit) \wedge Bel(\mathbf{Claire}, v \wedge has(\mathbf{Max}, \mathbf{A} \heartsuit))) \rrbracket(s) \\ &= \llbracket Bel(\mathbf{Claire}, v \wedge has(\mathbf{Max}, \mathbf{A} \heartsuit)) \rrbracket(\llbracket Bel(\mathbf{Max}, v \wedge has(\mathbf{Max}, \mathbf{A} \heartsuit)) \rrbracket(s)) \\ &= \llbracket Bel(\mathbf{Claire}, v \wedge has(\mathbf{Max}, \mathbf{A} \heartsuit)) \rrbracket(\{(Bel, Max, \mathbf{v} \cup \{(Has, Max, \mathbf{A} \heartsuit; 1)\}; 1)\} \cup s) \\ &= \{(Bel, Claire, \mathbf{v} \cup \{(Has, Max, \mathbf{A} \heartsuit; 1)\}; 1), (Bel, Max, \mathbf{v} \cup \{(Has, Max, \mathbf{A} \heartsuit; 1)\}; 1)\} \cup s. \end{aligned}$$

We call such a semantics \cup -based, defined as follows.

Definition 5. *A (dynamic) semantics S of a language L is a \cup -based w.r.t. X iff for any (non-negative) sentence $\varphi \in L$, S assigns it a function $F : x \mapsto x \cup y$, for some $x, y, x \cup y \in X$.*

Lemma 1. *\mathfrak{S}_1 is \cup -based w.r.t. SIT .*

⁴ Gerbrandy and Groeneveld (1997) “Reasoning about Information change,” ms. proposes “Dynamic Epistemic Semantics” (*DES*). Although this framework is concerned with the change of common knowledge, this framework is not concerned about updates of circular objects directly.

Proof. (By induction of the complexity of sentences of \mathfrak{L} .) We have only to show that each sentence φ of form of $\varphi_1 \wedge \varphi_2$ and $\downarrow v.\varphi_1$ is assigned an update function such that for each $s \in SIT$, $s \mapsto s \cup t$ for some $t \in SIT$. Assume $\llbracket \varphi_1 \rrbracket(s) = t$ and $\llbracket \varphi_2 \rrbracket(s) = u$.

$$\begin{aligned}\llbracket \varphi_1 \wedge \varphi_2 \rrbracket(s) &= \llbracket \varphi_1 \rrbracket \circ \llbracket \varphi_2 \rrbracket(s) \\ &= \llbracket \varphi_1 \rrbracket \circ \llbracket \varphi_2 \rrbracket(s) \\ &= \llbracket \varphi_2 \rrbracket(\llbracket \varphi_1 \rrbracket(s)) \\ &= \llbracket \varphi_2 \rrbracket(s \cup t) \\ &= s \cup (t \cup u),\end{aligned}$$

$$\begin{aligned}\llbracket \downarrow v.\varphi_1 \rrbracket(s) &= \mathbf{v} \\ \mathbf{v} &= \llbracket \varphi_1 \rrbracket(s) \\ \mathbf{v} &= t \cup s.\end{aligned}$$

□

However we can not deal with dialogues (3) as shared belief revision by such a \cup -based update.

- (3) a. Claire: You have the ace of hearts; Max: Yes.(That's right.) [Agreement]
b. Max: I have the ace of hearts; Claire: Uh-huh. [Acknowledgement]

(3a-3b) can be considered as the formulas in (4), respectively, where $h = \text{has}(\mathbf{Max}, \mathbf{A\heartsuit})$ and $\downarrow v.\text{Bel}(\mathbf{Claire}, v \wedge h)$ means Claire's introspective belief about h .

- (4) a. $(\downarrow v.\text{Bel}(\mathbf{Claire}, v \wedge h)) \wedge \text{Bel}(\mathbf{Max}, v \wedge h)$
 $\equiv (\downarrow v.\text{Bel}(\mathbf{Claire}, v \wedge h) \wedge \text{Bel}(\mathbf{Max}, v \wedge h))$
 b. $(\downarrow v.\text{Bel}(\mathbf{Max}, v \wedge h)) \wedge \text{Bel}(\mathbf{Claire}, v \wedge h)$
 $\equiv (\downarrow v.\text{Bel}(\mathbf{Max}, v \wedge h) \wedge \text{Bel}(\mathbf{Claire}, v \wedge h))$

Namely, in \mathfrak{S}_1 the \downarrow -operator cannot take its scope dynamically w.r.t. \wedge , since such dynamic scope-taking can not be dealt with by a simple \cup -based update as explained in (5), where $h' = (\text{Has}, \text{Max}, \mathbf{A\heartsuit}; 1)$.

- (5) $s \cup \{(Bel, Claire, \mathbf{v} \cup \{h'\}; 1), (Bel, Max, \mathbf{v} \cup \{h'\}; 1)\} \neq s \cup \{(Bel, Claire, \mathbf{v} \cup \{h'\}; 1), (Bel, Max, \mathbf{v} \cup \{h'\}; 1)\}$, since in the left $\mathbf{v} = \{(Bel, Max, \mathbf{v} \cup \{h'\}; 1)\}$ but in the right $\mathbf{v} = \{(Bel, Max, \mathbf{v} \cup \{h'\}; 1), (Bel, Claire, \mathbf{v} \cup \{h'\}; 1)\}$.

From the viewpoint of equational systems, such a dynamism is considered as revision of the following revision of the equational systems.

- (6) a. $\{\mathbf{v} = \{(Bel, Claire, \mathbf{v} \cup \{h'\}; 1)\}\} \mapsto \{\mathbf{v} = \{(Bel, Max, \mathbf{v} \cup \{h'\}), (Bel, Claire, \mathbf{v} \cup \{h'\})\}\}$

- b. $\{\mathbf{v} = \{(Bel, Max, \mathbf{v} \cup \{h'\}; 1)\}\} \mapsto \{\mathbf{v} = \{(Bel, Max, \mathbf{v} \cup \{h'\}), (Bel, Claire, \mathbf{v} \cup \{h'\})\}\}$

From this example, we can conclude the following proposition.

Proposition 2.

1. A semantics \cup -based w.r.t. SIT cannot assign a function which can revise circular situation to a sentence of its object language.
2. A semantics \cup -based w.r.t. SIT cannot admit the \downarrow -operator's dynamic scope-taking w.r.t. \wedge .

3.3 \mathfrak{S}_2 : A Substitution-Based Semantics of \mathfrak{L}

According to [5], a revision of circularity itself can be defined by a *substitution* which can be defined by *corecursion*.

Definition 6 (Barwise & Moss [5]). A substitution is a function θ whose domain is a set of urelements. A substitution operation is an operation sub whose domain consists of a class of pairs (θ, b) where θ is a substitution and $b \in \mathcal{U} \cup V_{afa}[\mathcal{U}]$, such that the following conditions are met.

1. If $x \in \text{dom}(\theta)$, then $sub(\theta, x) = \theta(x)$.
2. If $x \in \mathcal{U} \setminus \text{dom}(\theta)$, then $sub(\theta, x) = x$.
3. For all sets b , $sub(\theta, b) = \{sub(\theta, a) \mid a \in b\}$.

[5] has shown the existence and uniqueness of sub [5] (Theorem 8.1).

Example 3. A substitution θ_b as the revision function required in (7) are defined by corecursion as follows:

$$\theta_b(u) = \theta_b(u) \cup \{b\},$$

for all indeterminate u .

$$(7) \quad \{x = \{a, x\}\} \mapsto \{x = \{a, b, x\}\}$$

That is, $\theta_b(x) = \theta_b(x) \cup \{b\} = \{\theta_b(x), a\} \cup \{b\} = \{\theta_b(x), a, b\}$.

The existence and uniqueness of the solution to the equation $\theta_b(x) = \{\theta_b(x), a, b\}$ are guaranteed by the Anti-Foundation Axiom.

Similarly, the required revision functions in (6) is defined by corecursion as follows.

$$(8) \quad F_{a,p}(x) = F_{a,p}(x) \cup \{(Bel, a, F_{a,p}(x) \cup \{p\}; 1)\}$$

Dialogues (3) are interpreted as follows:

$$\begin{aligned} & F_{Max, h'}(\llbracket \downarrow v. Bel(Claire, v \wedge h) \rrbracket(s)) \\ &= F_{Max, h'}(\mathbf{v}) \cup \{(Bel, Max, F_{Max, h'}(\mathbf{v}) \cup \{h'\}; 1)\} \\ &= F_{Max, h'}(\{(Bel, Claire, F_{Max, h'}(\mathbf{v}) \cup \{h'\}; 1) \cup s\} \cup \{(Bel, Max, F_{Max, h'}(\mathbf{v}) \cup \{h'\}; 1)\}) \\ &= \{(Bel, Claire, F_{Max, h'}(\mathbf{v}) \cup \{h'\}; 1) \cup \{F_{Max, h'}(p) \mid p \in s\} \cup \{(Bel, Max, F_{Max, h'}(\mathbf{v}) \cup \{h'\}; 1)\} \\ &= \{(Bel, Claire, F_{Max, h'}(\mathbf{v}) \cup \{h'\}; 1), (Bel, Max, F_{Max, h'}(\mathbf{v}) \cup \{h'\}; 1)\} \cup \{F_{Max, h'}(p) \mid p \in s\} \\ &= \{(Bel, Claire, F_{Max, h'}(\mathbf{v}) \cup \{h'\}; 1), (Bel, Max, F_{Max, h'}(\mathbf{v}) \cup \{h'\}; 1)\} \cup \{F_{Max, h'}(p) \mid p \in s\} \\ &= F_{Max, h'}(\mathbf{v}) \end{aligned}$$

That is, $F_{Max,h'}(\mathbf{v}) = \{(Bel, Claire, F_{Max,h'}(\mathbf{v}) \cup \{h'\}; 1), (Bel, Max, F_{Max,h'}(\mathbf{v}) \cup \{h'\}; 1)\} \cup \{F_{Max,h'}(p) | p \in s\}$.

$$\begin{aligned}
& F_{Claire,h'}(\llbracket \downarrow v.Bel(Max, v \wedge h) \rrbracket(s)) \\
&= F_{Claire,h'}(\mathbf{v}) \cup \{(Bel, Claire, F_{Claire,h'}(\mathbf{v}) \cup \{h'\}; 1)\} \\
&= F_{Claire,h'}(\{(Bel, Max, F_{Claire,h'}(\mathbf{v}) \cup \{h'\}; 1) \cup s\} \cup \{(Bel, Claire, F_{Claire,h'}(\mathbf{v}) \cup \{h'\}; 1)\}) \\
&= \{(Bel, Max, F_{Claire,h'}(\mathbf{v}) \cup \{h'\}; 1)\} \cup \{F_{Claire,h'}(p) | p \in s\} \cup \{(Bel, Claire, F_{Claire,h'}(\mathbf{v}) \cup \{h'\}; 1)\} \\
&= \{(Bel, Max, F_{Claire,h'}(\mathbf{v}) \cup \{h'\}; 1), (Bel, Claire, F_{Claire,h'}(\mathbf{v}) \cup \{h'\}; 1)\} \cup \{F_{Claire,h'}(p) | p \in s\} \\
&= F_{Claire,h'}(\mathbf{v})
\end{aligned}$$

That is, $F_{Claire,h'}(\mathbf{v}) = \{(Bel, Max, F_{Claire,h'}(\mathbf{v}) \cup \{h'\}; 1), (Bel, Claire, F_{Claire,h'}(\mathbf{v}) \cup \{h'\}; 1)\} \cup \{F_{Claire,h'}(p) | p \in s\}$.

Although this result achieves the aim of providing the formal semantics of dialogues as mutual belief revisions, it can not grasp the dynamism of \downarrow and \wedge in (4), since the semantics gives substitution $F_{a,p}$ to utterances like ‘Yes’ and ‘Uh-huh’ directly and not to formulas. As the result, we must revise language \mathcal{L} itself by adding a new special formula, say, $\mathbf{F}(\mathbf{a}, \mathbf{it})$, where \mathbf{a} means the utterer of agreements or acknowledgements and \mathbf{it} refers the agreed or acknowledged proposition. For example, dialogues (3) are expressed and the \downarrow -operators in the sentences take their scope dynamically w.r.t. \wedge as follows in this expanded language.

$$\begin{aligned}
(9) \quad & \text{a. } (\downarrow v.Bel(\mathbf{Claire}, v \wedge h)) \wedge \mathbf{F}(\mathbf{Max}, \mathbf{it}) \\
& \quad \equiv (\downarrow v.Bel(\mathbf{Claire}, v \wedge h) \wedge \mathbf{F}(\mathbf{Max}, \mathbf{it})) \\
& \quad \text{b. } (\downarrow v.Bel(\mathbf{Max}, v \wedge h)) \wedge \mathbf{F}(\mathbf{Claire}, \mathbf{it}) \\
& \quad \equiv (\downarrow v.Bel(\mathbf{Max}, v \wedge h) \wedge \mathbf{F}(\mathbf{Claire}, \mathbf{it}))
\end{aligned}$$

But this dynamic scope-taking is special only for the predicate \mathbf{F} , and generally, dynamic scope-taking as in (4) is not hold in this semantics.

We can summarize this discussion as follows.

Proposition 3.

1. In \mathfrak{S}_2 , $\downarrow v$ doesn't take its scope dynamically w.r.t. \wedge .
2. In \mathfrak{S}_2 with predicate \mathbf{F} , for some variable v , $\downarrow v$ doesn't take its scope dynamically w.r.t. \wedge .

4 Revision Systems of Circular Objects Based on Membership Description Systems

We investigate an alternative approach to circular objects which can grasp the dynamism of \downarrow and \wedge , here. Our main idea is based on the fact that any set can be specified by membership relation as well as equational systems of sets, as follows:

$$\{x = \{x\}\} \Leftrightarrow (x \in x) \wedge \forall y (y \neq x \rightarrow y \notin x).$$

Namely, we can construct the specification of a set based on membership relation, called a *membership description system* (MDS), from any equational system of sets as follows.

Definition 7.

1. An MDS \mathcal{D} is a quadruple:

$$(X, A, \epsilon^+, \epsilon^-),$$

where

- $\epsilon^+, \epsilon^- \subseteq \text{pow}(A \cup X) \times X$, and
- $\epsilon^+ \cap \epsilon^- = \emptyset$.

2. A solution to an MDS \mathcal{D} is a function θ with domain X satisfying $\theta(x) \supseteq \{\theta(y) | y\epsilon^+ x, y \in X\} \cup \{a | a\epsilon^+ x, a \notin A\} - \{\theta(y) | y\epsilon^- x, y \in X\}$, for each $x \in X$.

We will also write an MDS as a set of propositions on membership, e.g., $\{x\epsilon^+ x\}$ for $(\{x\}, \emptyset, (x, x), \emptyset)$.

Furthermore, we will define some concepts on MDSs as follows.

Definition 8.

1. An MDS $\mathcal{D} = (X, A, \epsilon^+, \epsilon^-)$ is a subMDS of $\mathcal{D}' = (X', A, \epsilon'^+, \epsilon'^-)$, written $\mathcal{D} \sqsubseteq \mathcal{D}'$, if $X \subseteq X'$, $\epsilon^+ \subseteq \epsilon'^+$ and $\epsilon^- \subseteq \epsilon'^-$. \mathcal{D}' is an expansion of \mathcal{D} . If $X = X'$ and $\mathcal{D} \sqsubseteq \mathcal{D}'$, \mathcal{D}' is an extension of \mathcal{D} , written $\mathcal{D} \subseteq \mathcal{D}'$.
2. An MDS $\mathcal{D} = (X, A, \epsilon^+, \epsilon^-)$ is partial if $\epsilon^+ \cup \epsilon^- \subset \text{pow}(A \cup X) \times X$. \mathcal{D} is complete if $\epsilon^+ \cup \epsilon^- = \text{pow}(A \cup X) \times X$.
3. Let $\mathcal{E} = (X, A, e)$ be an equational system of sets. Then the MDS $\mathcal{D}^{\mathcal{E}}$ constructed from \mathcal{E} is an MDS:

$$(X, A, \epsilon^+, \epsilon^-),$$

where

- $x\epsilon^+ y$ iff $x \in e(y)$, and
 - $x\epsilon^- y$ iff $x \notin e(y)$,
- for all $y \in X$.

Proposition 4.

1. If an MDS \mathcal{D} is complete, there is a unique solution to \mathcal{D} .
2. An MDS $\mathcal{D}^{\mathcal{E}}$ constructed from an equational system of sets \mathcal{E} is complete.
3. For an MDS \mathcal{D} , there may be more than one solution to \mathcal{D} .

Proof. (1) Let $\mathcal{D} = (X, A, \epsilon^+, \epsilon^-)$ be a complete MDS. Let \mathbf{e} be a function constructed by the following conditions:

- $x\epsilon^+ y$ iff $x \in \mathbf{e}(y)$, and
- $x\epsilon^- y$ iff $x \notin \mathbf{e}(y)$,

for all $y \in X$. That is, $\mathbf{e}(y) = \{x | x\epsilon^+ y\}$. Then (X, A, \mathbf{e}) is obviously an equational system of sets. By AFA, there is a unique solution to it.

(2) Directly from definition of an MDS constructed from an equational system of sets and (1).

(3) Let us consider an MDS $\{y\epsilon^+ x\}$, and functions $\theta_0 = \{(x, \{\theta_0(y)\}), (y, \{\theta(y)\})\}$, $\theta_1 = \{(x, \{\theta_1(x), \theta_1(y)\}), (y, \{\theta_1(y)\})\}$, and $\theta_2 = \{(x, \{\theta_2(x), \theta_2(y)\}), (y, \{\theta_2(x), \theta_2(y)\})\}$. θ_0 , θ_1 and θ_2 are solutions to the MDS. \square

Therefore, an MDS has many solutions, since it can be an MDS of any of its expansions which are constructed from equational systems of sets. We call θ a *minimal solution* to an MDS $\mathcal{D} = (X, A, \epsilon^+, \epsilon^-)$ if θ can be defined as a unique solution of the equational system of sets $\mathcal{E} = (X, A, e)$, where $e(y) = \{x \in X \cup A \mid x\epsilon^+y\}$ for each $y \in X$. We call θ a *maximal solution* to an MDS $\mathcal{D} = (X, A, \epsilon^+, \epsilon^-)$ if θ can be defined as a unique solution of the equational system of sets $\mathcal{E} = (X, A, e)$, where $e(y) = \{x \in X \cup A \mid \text{not } x\epsilon^-y\}$ for each $y \in X$. We choose a minimal solution to an MDS as its intended interpretation. If an MDS has no minimal solution, we call it *undefined*. Then we can consider a sequence of subMDSs $\mathcal{D}_0 \subseteq \dots \subseteq \mathcal{D}_n$ as a revision process of circular objects.

Example 4. The revision in (7) is specified as a simple adding operation $\{b\epsilon^+x\}$ to $\{x\epsilon^+x, a\epsilon^+x\}$.

4.1 \mathfrak{S}_3 : A Semantics Based on MDSs

Now we will propose \mathfrak{S}_3 , a semantics of \mathfrak{L} based on MDSs. In \mathfrak{S}_3 , revisions of circular objects are considered as transitions of MDSs such that each of them are related by subMDS or expansion relations. For example, revisions in (3) are defined by the following expanding operations:

- (10) a. $\{(Bel, Claire, s; 1)\epsilon^+v, h'\epsilon^+s, v\epsilon^+s\} \cup \{(Bel, Claire, s; 1)\epsilon^+v\}.$
 b. $\{(Bel, Max, s; 1)\epsilon^+v, h'\epsilon^+s, v\epsilon^+s\} \cup \{(Bel, Claire, s; 1)\epsilon^+v\}.$

Therefore, we can provide a natural semantics of (4) based on MDSs as follows.

Definition 9. In \mathfrak{S}_3 , for each sentence $\varphi \in \mathfrak{L}$, φ is assigned an update function $\llbracket \varphi \rrbracket : \text{Context} \times \text{Root} \times \text{pow}(\text{MDS}) \rightarrow \text{Context} \times \text{Root} \times \text{pow}(\text{MDS})$, where *Context* is a set of set of assignments, *Root* is an indeterminate (if r is an indeterminate and $a \in AG$, then $a(r)$ is an indeterminate such that $r \neq a(r)$), and $\llbracket \varphi \rrbracket(C, r, D)$ is defined by induction of the complexity of φ as follows:

- $\llbracket has(\mathbf{a}, \mathbf{c}) \rrbracket(C, r, D) = (C, r, D \cap \{(Has, a, c; 1)\epsilon^+r\}),$
- $\llbracket Bel(\mathbf{a}, \varphi) \rrbracket(C, r, D) = \llbracket \varphi \rrbracket(C, r, D \cap \{(Bel, a, a(r); 1)\epsilon^+r\}) \cup \llbracket \varphi \rrbracket(C, a(r)),$
- $\llbracket \downarrow v\varphi \rrbracket(C, r, D) = \llbracket \varphi \rrbracket([\sim_v](C), r, D),$ where $[\sim_v](C) = \{h \mid g \sim_v h, g \in C\},$
- $\llbracket \varphi_1 \wedge \varphi_2 \rrbracket = \llbracket \varphi_1 \rrbracket \circ \llbracket \varphi_2 \rrbracket,$
- $\llbracket has(\mathbf{a}, \mathbf{c}) \rrbracket(C, r) = \{(Has, a, c; 1)\epsilon^+r\},$
- $\llbracket Bel(\mathbf{a}, \varphi) \rrbracket(C, r) = \{(Bel, a, a(r); 1)\epsilon^+r\} \cup \llbracket \varphi \rrbracket(C, a(r)),$
- $\llbracket v \rrbracket(C, r) = \{g(v)\epsilon^+r \mid g \in C\},$
- $\llbracket \varphi_1 \wedge \varphi_2 \rrbracket(C, r) = \llbracket \varphi_1 \rrbracket(C, r) \cup \llbracket \varphi_2 \rrbracket(C, r).$

Proposition 5. $(\downarrow v.Bel(\mathbf{Claire}, v \wedge h)) \wedge Bel(\mathbf{Max}, v \wedge h) \equiv (\downarrow v.Bel(\mathbf{Claire}, v \wedge h) \wedge Bel(\mathbf{Max}, v \wedge h))$ holds in \mathfrak{S}_3 .

Proof. Let c be a set of assignments, r be a root, and D is a set of MDSs.

$$\begin{aligned}
& \llbracket (\downarrow v. \text{Bel}(\mathbf{Claire}, v \wedge h)) \wedge \text{Bel}(\mathbf{Max}, v \wedge h) \rrbracket (C, r, D) \\
&= \llbracket (\downarrow v. \text{Bel}(\mathbf{Claire}, v \wedge h)) \rrbracket \circ \llbracket \text{Bel}(\mathbf{Max}, v \wedge h) \rrbracket (C, r, D) \\
&= \llbracket \text{Bel}(\mathbf{Max}, v \wedge h) \rrbracket (\llbracket \downarrow v. \text{Bel}(\mathbf{Claire}, v \wedge h) \rrbracket (C, r, D)) \\
&= \llbracket \text{Bel}(\mathbf{Max}, v \wedge h) \rrbracket (\llbracket \text{Bel}(\mathbf{Claire}, v \wedge h) \rrbracket ([\sim_v](C), r, D)) \\
&= \llbracket \text{Bel}(\mathbf{Claire}, v \wedge h) \wedge \text{Bel}(\mathbf{Max}, v \wedge h) \rrbracket ([\sim_v](C), r, D) \\
&= \llbracket (\downarrow v. \text{Bel}(\mathbf{Claire}, v \wedge h)) \wedge \text{Bel}(\mathbf{Max}, v \wedge h) \rrbracket (C, r, D)
\end{aligned}$$

□

Corollary 2. In \mathfrak{S}_3 , $\downarrow v$ takes its scope dynamically w.r.t. \wedge .

5 Conclusion

We have seen three types of dynamic semantics of a language of mutual beliefs as an example of the revision of circular objects. Semantics which is \cup -based w.r.t. *SIT* cannot treat the dynamic scope-taking \downarrow -operator w.r.t. \wedge . However, semantics based on MDSs, which are proposed as revision systems of circular objects here, admit the dynamic scope-taking \downarrow -operator w.r.t. \wedge .

References

1. Aczel, P. *Non-well-founded Sets*. CSLI, Stanford, 1987.
2. Barwise, J. *Situation in Logic*. CSLI, Stanford, 1989.
3. Barwise, J., and John Etchemendy. *The Liar*. Oxford University Press, Oxford, 1987.
4. Benthem, J. van. *Exploring Logical Dynamics*. CLSI Publications, Stanford, 1996.
5. Barwise, J., and L. Moss. *Vicious Circles*. CSLI, Stanford, 1996.
6. Colmerauer, A. Prolog and infinite trees. In Clark, K. L., and S. A. Tärnlund, editors, *Logic Programming*, pages 231–252. Academic Press, London, 1982.
7. Davey, B. A., and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, 1990.
8. Fagin, R., J. Y. Halpern, Y. Moses, and V. Y. Vardi. *Reasoning about Knowledge*. The MIT Press, Cambridge, 1995.
9. Groenendijk, J., and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100, 1991.
10. Groenendijk, J., and M. Stokhof. Two theories of dynamic semantics. In Jan van Eijck, editor, *Logics in AI: European Workshop JELIA '90*, pages 55–64. Springer Verlag, Berlin, 1991.
11. Groeneveld, W. Dynamic semantics and circular propositions. *Journal of Philosophical Logic*, 23:267–306, 1994.
12. Lloyd, W. J. *Foundations of Logic Programming*. Springer-Verlag, Berlin, second, extended edition, 1987.

Lexicalized Proof-Nets and TAGs

Sylvain Pogodalla

Xerox Research Center Europe

6 chemin de Maupertuis

F-38240 Meylan, France

`Sylvain.Pogodalla@xrce.xerox.com`

Introduction

First introduced by [12], pomset linear logic can deal with linguistic aspects by inducing a partial order on words. [9] uses this property: it defines modules (or partial proof-nets) which consist in entries for words, describing both the category of the word and its behavior when interacting with other words. Then the natural question of comparing the generative power of such grammars with Tree Adjoining Grammars [7], as [6] pointed some links out, arises.

To answer this question, we propose a logical formalization of TAGs in the framework of linear logic proof-nets. We aim to model trees and operations on these trees with a restricted part of proof-nets (included in the intuitionistic ones), and we show how this kind of proof-nets expresses equivalently TAG-trees.

The first section presents all the definitions. Then, in the second section, we propose a fragment of proof-nets allowing the tree encoding and the third section defines the way we model operations on proof-nets. As replying to the second section, the fourth one allows us to come back from proof-nets to trees. Finally, section 5 shows examples of how the definitions and properties work.

1 Definitions

1.1 TAG

First, extending the original definition of TAG [7] with the substitution operation as in [15,3], we get:

Definition 1. *A TAG is a 5-uple (V_N, V_T, S, I, A) where:*

1. V_N is a finite set of non-terminal symbols,
2. V_T is a finite set of terminal symbols,
3. S is a distinguished non-terminal symbol, the start symbol,
4. I is a set of initial trees,
5. A is a set of auxiliary trees.

Initial trees represent basic sentential structures or basic categories. They have non-terminal nodes to be substituted for and serve as arguments to themselves or to auxiliary trees. A leaf (marked with a $*$) with the same label as the root node characterizes the auxiliary trees. An elementary tree is either an initial tree or an auxiliary tree.

The TAGs we are considering here will always be such that every elementary tree has at least a (terminal) node labeled by a terminal symbol, so that the TAGs are *lexicalized*.

Second, for referring trees and nodes in these trees, we use the notations [7] defined for trees on the finite alphabet V ($V = V_N \cup V_T$):

Definition 2. γ is a tree over V iff it is a function from D_γ into V where the domain D_γ is a finite subset of J^* such that:

1. if $q \in D_\gamma, p < q$, then $p \in D_\gamma$;
2. if $p \cdot j \in D_\gamma, j \in J$, then $p \cdot 1, p \cdot 2, \dots, p \cdot (j - 1) \in D_\gamma$

where J^* is the free monoid generated by J the set of all natural numbers, \cdot is the binary operation, 0 is the identity and for $q \in J^*, p \leq q$ iff there is a $r \in J^*$ such that $q = p \cdot r$, and $p < q$ iff $b \leq q$ and $p \neq q$.

We call elements in D_γ addresses of γ . If $(p, X) \in \gamma$, then we say that X is the label of the node at the address p in γ . We write it $\gamma(p) = X$.

Third, we require another property:

Property 1 (ϖ). A tree γ satisfies the ϖ property iff $\forall p \in D_\gamma$ such that $\gamma(p) \in V_T$ then $p = q \cdot 1$ and $q \cdot 2 \notin D_\gamma$.

It means that for a tree, if a node is terminal, labeled by a terminal symbol, then it is the unique daughter of its mother-node. Performing the two operations (substitution and adjunction) preserves this property.


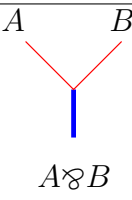
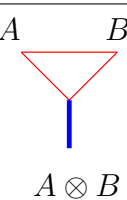
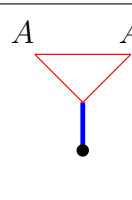
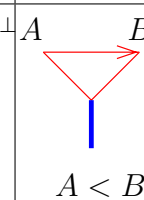
But considering TAGs whose elementary trees have the ϖ property does not restrict the generated language. Indeed, if G is a TAG whose elementary trees do not have the ϖ property, $T(G)$ is the set of all the trees that the two operations produce in the TAG G , $L(G)$ is the language that G generates (the set of strings as sequences of terminal symbol-labeled leaves of trees in $T(G)$) and if G_1 is the TAG made from G in order to get the elementary trees have the ϖ property, then we have no special relation between $T(G)$ and $T(G_1)$. Nevertheless, we have $L(G) \subset L(G_1)$.

Fourth, another restriction, similar to the restriction from [3], is to avoid the use of trees γ such that:

$$\exists p \in D_\gamma, \gamma(p) = \gamma(p \cdot 1) \text{ and } p \cdot 2 \notin D_\gamma$$

It means there is no tree that have an X -labeled node whose unique leaf is also an X -labeled node.

Table 1. Definitions of the links

Name	axiom	\wp	\otimes	Cut	$<$
Premises	none	A and B	A and B	A and A^\perp	A and B
R&B-graph					
Conclusions	A and A^\perp	$A \wp B$	$A \otimes B$	none	$A < B$

1.2 Lexicalized Proof-Nets

Proof-nets in linear logic have become familiar [4,12,2]. In this paper, we refer to [13]’s notations of proof-nets, extended to the ordered calculus [14]. It defines proof-nets as bicolored (Red and Blue, or Regular and Bold) graphs with the five links corresponding to the axiom, the tensor (\otimes), the before ($<$), the par (\wp) and the cut (Cut). This calculus enjoys cut-elimination [12], a crucial property for our modeling.

Let us remind the main definitions:

Definition 3 (RB-graphs). *A RB-graph is a graph with coloured edges (blue and red, or bold and regular). B-edges are undirected. The R-edges may be undirected or directed, in which case we call them R-arcs.*

Definition 4 (Links). *There are five sorts of links, defined as RB-graphs (see table 1).*

Definition 5 (Proof-structure). *A proof structure is a RB-graph such that any B-edge is the conclusion of exactly one link and the premise of at most one link (the B-edges which are not a premise of any link are called conclusions of the proof-structure, they contain all the cuts), provided with a set of R-arcs between conclusions which defines a strict partial order.*

Definition 6 (Proof-net). *An ordered proof-net is a proof-structure which contains no alternate elementary circuit¹.*

We speak about correctness criterion, or correctness checking to speak about the absence of any alternate elementary circuit in a proof-structure, so that we know whether a proof-structure is a proof-net or not.

¹ a path of even length, starting and ending on the same vertex, using only once every other vertex and with alternating blue and red edges.

Table 2. Rewriting rules on proof-nets for cut-elimination

$A \bullet A^\perp$	$(A < B) \bullet (A^\perp < B^\perp)$	$(A \wp B) \bullet (A^\perp \otimes B^\perp)$

Proposition 1 (Cut-elimination). *Cuts can be eliminated. More precisely: let Π be a proof-net whose conclusions are $F_1, F_2, \dots, F_k, \bullet_1, \dots, \bullet_p$ ordered by \Re (where F_1, \dots, F_k are formulas and all the \bullet_i are cuts) it is possible to rewrite Π as Π' with conclusions F_1, \dots, F_k ordered by the restriction of \Re to these formulas. Moreover, this rewriting enjoys strong normalisation and confluence [12].*

Table 2 shows the rewriting rules on proof-nets.

We do not consider all proof-nets, but only those taking their formulas in the \mathcal{C} language defined as follows: \mathcal{A} is an alphabet of atomic formulas (we shall take $\mathcal{A} = V_N$) and

$$\begin{aligned} \mathcal{B}_1 &::= \mathcal{A}^\perp | \mathcal{A}^\perp \wp \mathcal{B}_1 & \mathcal{B}_2 &::= \mathcal{A} | \mathcal{B}_2 < \mathcal{A} \\ \mathcal{C} &::= \mathcal{A} | \mathcal{A}^\perp | \mathcal{A} \otimes \mathcal{A}^\perp | \mathcal{A} \wp \mathcal{A}^\perp | \mathcal{B}_1 \wp (\mathcal{B}_2 \otimes \mathcal{A}^\perp) \end{aligned}$$

Moreover, we always set the \Re partial order relation to \emptyset .

In addition to logical formulas, we also decorate proof-nets with *labels* from a finite set of terminal symbols. Then, as a restriction of the lexicalized intuitionistic labeled proof-nets defined in [10], we define:


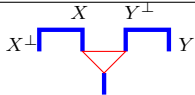
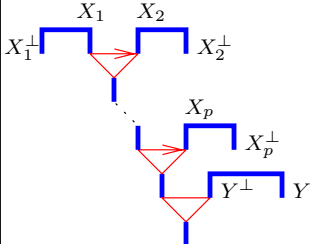
- Definition 7.**
- Output:** An output is either a B -edge that is labeled by a positive atom or the conclusion of a par-link between two atoms dual one from another (we call such a conclusion a par-gate).
 - Intuitionistic proof-net:** An intuitionistic proof-net (IPN) is a proof-net which contains one and only one output.
 - Simply lexicalized proof-net:** A simply lexicalized (SLIPN) is a lexicalized IPN the conclusions of which are of the form:
 - atoms or dual of atoms,
 - output,
 - $A_{j_1}^\perp \wp \dots \wp A_{j_n}^\perp \wp ((A_1 < \dots < A_m) \otimes Y^\perp)$ with $j_i \in [1, m]$ ($j_i \leq j_k$ iff $i \leq k$) and every $A_{j_i}^\perp$ is labeled by a string w_{j_i} ($A_i \in V_N$ and $w_j \in V_T$),
 - $X \otimes X^\perp$ with X atomic (we call such a conclusion a tensor-gate).

Note that neither the lexicalization nor the intuitionistic property contribute to the proof-structure correctness checking. The correctness criterion does not change (since it's (almost) the only one to handle the before-link, we would rather keep it). On the other hand, the intuitionistic feature allow the use of (a variant of) intuitionistic paths [8]. It is stable under the operations we are considering and paths enable the decoding from proof nets to trees (see section 4.2).

2 From Trees to Proof-Nets

This section defines for each elementary tree of a TAG a corresponding SLIPN with an induction on the height of the trees. The set of atoms for logical formulas comes from V_N , and labels come from V_T .

Table 3. Initial trees mapping

	Trees	Proof-nets
$h = 1$	$\begin{array}{c} X \\ \\ x \end{array}$	
	$\begin{array}{c} Y \\ \\ X \end{array}$	
	$\begin{array}{c} Y \\ / \quad \quad \backslash \\ X_1 \quad \dots \quad X_i \quad \dots \quad X_p \end{array}$	
$h = 2$	$\begin{array}{c} Y \\ / \quad \quad \backslash \\ X_1 \dots X_{i_1} \quad X_{i_2} \dots X_{i_p} \dots X_n \\ \quad \quad \\ x_{i_1} \quad x_{i_2} \quad x_{i_p} \end{array}$	(see figure 1(a))
general case	$\begin{array}{c} Y \\ / \quad \quad \backslash \\ X_1 \dots X_{i_1} \quad X_{j_1} \quad \dots \quad X_{j_m} \quad \dots \quad X_{i_l} \dots X_{i_p} \quad X_n \\ \quad \triangle \quad \triangle \quad \quad \\ x_{i_1} \quad \quad \quad x_{i_l} \quad x_{i_p} \end{array}$	(see figure 1(b))

2.1 Initial Trees

We first give the general idea of this encoding on the tree T_2 of table 5. Forgetting the lexicalized part, we can read this tree as a terminal node (N) preceding another terminal node (V) to produce their mother-node (P). We can express this idea with a formula of pomset logic: $(N < V) \multimap P$. But do not forget that this is a brick from which we want to derive S . Moreover, proof-nets correspond to one-sided sequent, so that, actually, we are more interested in the dual of such formula, namely: $(N < V) \otimes P^\perp$. Thus, we shall have a SLIPN with this latter sub-formula, other connectives dealing with the lexicalization.

Table 3 sums up the translation. Note that for $h = 1$, the two latter cases do not belong to the considered TAG. Nevertheless, we require the definition of their corresponding SLIPNs for the next steps of the induction. The case $h = 2$ only presents the case where lexicalized subtrees' height (at least one exists) is 1 and other subtrees' height is 0, for we deal with the cases where other subtrees' height can be 1 in the next general case. For this latter, we possibly have $\{i_1, \dots, i_p\} = \emptyset$ and in figure 1(b) the $\Pi_{j_1}, \dots, \Pi_{j_m}$ are the inductively built SLIPNs corresponding to the subtrees of γ at X_{j_1}, \dots, X_{j_m} .

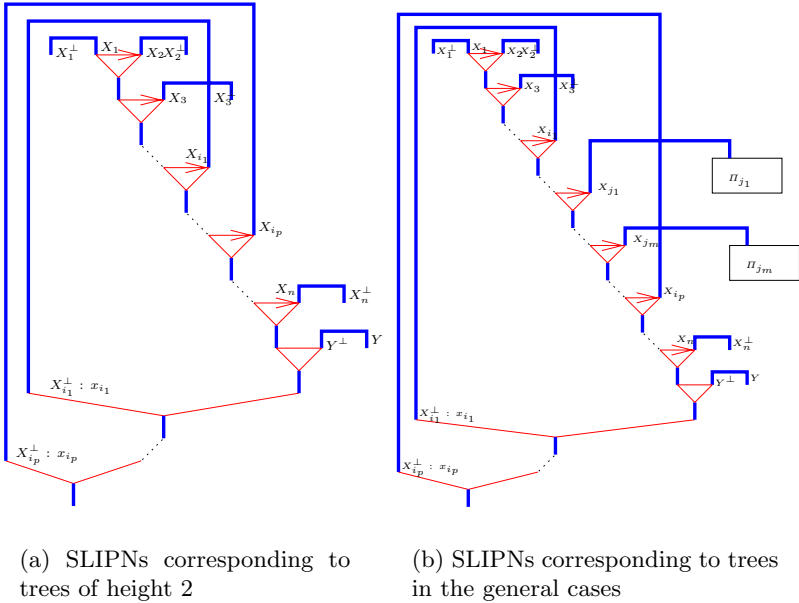


Fig. 1. SLIPNs for higher trees

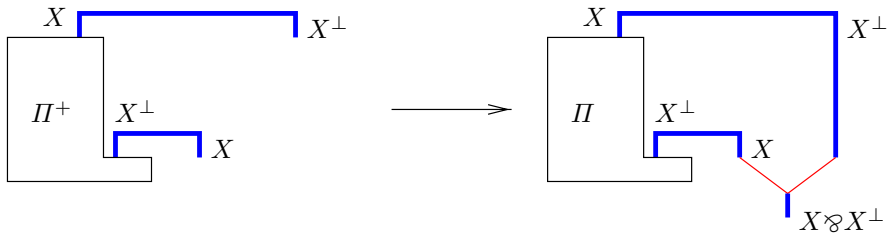


Fig. 2. Auxiliary trees

Definition 8. For every initial tree T , we call corresponding initial SLIPN, or T transformation, the SLIPN defined as in table 3 and figure 1.

- Remark 1.*
1. The unique output of every SLIPN corresponds to the root node of the tree, and every terminal node, labeled by a non-terminal symbol, of a tree corresponds to a conclusion of the corresponding SLIPN.
 2. There is a one-to-one mapping between the non-terminal symbol labeled nodes of a tree and the axiom links of the corresponding SLIPN.

2.2 Auxiliary Trees

Let γ be an auxiliary tree and let us define γ^+ as the same tree as γ except for its X^* node replaced with an X node. We call r the adress of N^* in γ so that $\gamma(r) = X^*$ and $\gamma^+(r) = X$. Then, following the definition in the previous section, we can define Π^+ the SLIPN corresponding to γ^+ . And, as γ is an auxiliary tree, $\gamma^+(r) = \gamma^+(0)$ and Π^+ has a conclusion X (corresponding to $\gamma^+(0)$) and a conclusion X^\perp (corresponding to $\gamma^+(r)$).

Thus we define Π the corresponding SLIPN to γ as the proof-net built from Π^+ in binding with a \wp -link its X and its X^\perp conclusions (see figure 2). Π is a (correct) SLIPN.

Definition 9. For every auxiliary tree γ , we call auxiliary corresponding SLIPN, or γ transformation, the SLIPN defined as above. Then, for every elementary tree, we call corresponding SLIPN the initial or auxiliary SLIPN corresponding to that tree.

3 Elementary Operations

This section deals with a particular case of the next section but focuses on the core operations which we can refer to during the generalisation.

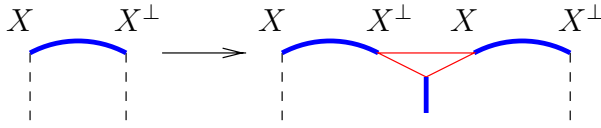


Fig. 3. Adding a tensor-gate

3.1 The Substitution Operation

Let γ_1 and γ_2 be two trees such that we can substitute γ_2 to a terminal node X (whose address is r) of γ_1 , and let Π_1 and Π_2 be their corresponding SLIPNs. Then $\gamma_1(r) = \gamma_2(0)$, and (cf. remark 1) Π_1 has a conclusion X^\perp , and the output of Π_2 is X .

Thus we can bind these conclusions with a cut-link and yield a new SLIPN from which we eliminate the cut and obtain a new SLIPN Π .

Definition 10. *For every tree γ resulting from the substitution of γ_2 to a node of γ_1 , we define its corresponding SLIPN Π as above.*

3.2 The Adjunction Operation

Preparing the Target Tree. In order to allow the adjunction of an auxiliary tree on a target tree, we need to modify a little bit this latter.

Let γ be the tree on which we want to perform an adjunction, r the address of the node where to perform the adjunction, and Π the corresponding SLIPN. As noted in remark 1, Π contains an axiom link $\gamma(r) \multimap \gamma(r)^\perp$ corresponding to the node $\gamma(r)$. So we can split this link into two axiom-links linked with a tensor-link and we obtain (with $X = \gamma(r)$) a new SLIPN Π' as shown in figure 3.

Proposition 2. *Adding a tensor-gate preserves the correctness of the proof-net.*

Actually, such a conclusion is an instance of cuts (as a cut is equivalent to a conclusion $(\exists X)(X \otimes X)$). Then if this tensor-gate remains unused, cut-elimination amounts to delete this tensor-gate and come back to a simple axiom-link. The second example of section 5 uses this feature at the very end of the derivation.

Performing the Operation. In this section, we define the SLIPN corresponding to the result of adjoining the tree γ_2 to γ_1 . In this preliminary case, both γ_1 and γ_2 are elementary trees (γ_2 is an auxiliary tree and γ_1 is the target tree). And Π_1 and Π_2 correspond to them.

We assume Π_1 already has a tensor-gate added on the axiom corresponding to the node where we want to adjoin γ_2 . So that if X labels the started-node of γ_2 , then X also labels the node of γ_1 receiving the adjunction and we have a

tensor-gate conclusion $X \otimes X^\perp$ for Π_1 and a par-gate $X \wp X^\perp$ (the output) for Π_2 (merely by construction).

Thus, we can bind Π_1 and Π_2 with a cut-link and eliminate this cut to obtain a new SLIPN (because there is no modification on the lexicalized parts, still no alternate elementary circuit and still only one output: γ_1 's one).

Definition 11. *For every tree γ resulting from the adjunction of the auxiliary tree γ_2 on γ_1 , we call the corresponding SLIPN the SLIPN built as above.*

3.3 Operations on Derived Trees

Up to now, we defined a way of modeling elementary trees in the framework of SLIPNs, and a way of combining these SLIPNs to model the adjunction and substitution operations on elementary trees. We now are about to extend this modeling on derived trees, so that a SLIPN will correspond to every tree (elementary or derived) of a TAG.

Definition 12. *For a tree γ and an elementary tree E_0 , we call derivation of γ the pair $< E_0, ((\diamond_1, E_1, \gamma_1), \dots, (\diamond_n, E_n, \gamma_n)) >$ such that $\gamma_n = \gamma$ and for every $i \in [1, n]$, γ_i results from the operation \diamond_i (adjunction or substitution) between γ_{i-1} and the elementary tree E_i .*

n is the length of the derivation.

Remark 2. For a derived tree, the derivation is not necessarily unique.

Definition 13. *Let γ be a derived tree from the derivation d . Then we can define the d-SLIPN corresponding to γ , built only with cut and cut-elimination (between unlabeled conclusions) from the SLIPNs corresponding to the elementary trees of the derivation.*

Actually, proving the existence of this SLIPN interests us more than the simple definition, as it also gives its construction's steps.

Proof. We prove the existence of $\Pi^{(d)}$ by induction. We also prove the property that if γ has a terminal node (except for the started node of an auxiliary tree), labeled by a non terminal symbol X , then $\Pi^{(d)}$ has a pendant conclusion X^\perp (corresponding to this node, hence not labeled neither).

1. if $l = 0$: γ is an elementary tree, and we already defined its transformation $\Pi^{(0)}$. And its construction also proves the property of the pendant conclusion.
2. if $l > 0$: Let $d_{l-1} = < \gamma_0, ((\diamond_1, E_1, \gamma_1), \dots, (\diamond_{l-1}, E_{l-1}, \gamma_{l-1})) >$ and $\Pi^{(d_{l-1})}$ be the SLIPN corresponding to γ_{l-1} in the d-derivation.

- a) If \diamond_l is the substitution of a leaf X of γ_{l-1} by E_l (whose transformation is π_l) then $\Pi^{(d_{l-1})}$ has an axiom-link $X \sqcup X^\perp$ in which X^\perp is a pendant conclusion (induction hypothesis), and π_l has an axiom-link $X \sqcup X^\perp$ in which X is a pendant conclusion ($E_l(0) = X$). Then we can link these two pendant conclusions with a cut-link, and eliminate it. This yields a new SLIPN. It also proves the property of pendant conclusion, as every terminal node of the new tree, labeled with a non-terminal symbol, already was terminal in one or another of the two trees so that (by induction hypothesis) they already had the property.
- b) if \diamond_l is the adjunction on the leave X of γ_{l-1} of the auxiliary tree E_l (whose transformation is π_l) then γ_{l-1} has an axiom-link $F \sqcup F^\perp$ we can replace (with respect to the SLIPNs class belonging) with two axiom-links linked together with a tensor-link (i.e. we add a tensor gate $X \otimes X^\perp$ as for adjunctions on elementary trees). π_l has a par-gate $X \wp X^\perp$ so that we can bind the two gates with a cut-link, and then eliminate this latter. We obtain a new SLIPN, and as above, the induction proves the property of the pendant conclusion.

□

Remark 3. This shows that cuts are only between atomic formulas or tensor and par-gate. Actually, the grammar given for the conclusions of SLIPNs indicates that no other cut can occur.

During this section, we made the assumption of allowing adjunctions at every time on every node. Of course, sometimes we do not want such possibilities. Allowing the tensor-gate addition only in the lexicon, and not during the derivation, brings a solution to this option.

Section 5 shows examples for both cases. In particular, the mildly-context sensitivity of TAGs, generating $\{a^n b^n c^n d^n\}$, illustrates the second case.

4 From SLIPNs to Trees

So far, we explained how, given a TAG and a derived tree in this TAG, we could obtain a SLIPN that we qualify as corresponding. But we now have to see how this SLIPN actually corresponds so that we shall henceforth be able to handle only proof-nets and translate the results on trees.

4.1 Polarities

Let us define a positive polarity ($^\circ$) and a negative one ($^\bullet$). Every formula is inductively polarized as follows: if α is an atom then α° and $\alpha^{\perp\bullet}$. Then, for each link we define the polarity of the conclusion from premises' ones as in table 4.

We call *input* the negative conclusions, and *output* the positive one (which is coherent with the previous definition of the output).

The grammar on SLIPNs' conclusions shows that SLIPNs are polarized.

Table 4. Polarities of the conclusions

\otimes	\circ	\bullet
\circ	\circ	\bullet
\bullet	\bullet	—

\wp	\circ	\bullet
\circ	—	\circ
\bullet	\circ	\bullet

$<$	\circ	\bullet
\circ	\circ	—
\bullet	—	—

4.2 Reading of SLIPNs

We give an algorithm for the reading of any cut-free SLIPN, based on formulas' polarities. It uses a very simple principle: following from a starting point (namely the output) the positive polarities, we define a path across the proof-net. And every time the path crosses an axiom-link, we add a node to the tree under construction. Actually, we build both the function γ and D_γ . Of course, the path can not cross twice the same axiom-link (such a possibility would occur only with par-gate).

As we shall see later, the path never cross a par-link (except par-gates, from the positive conclusion), always enter a tensor-link through a negative premise and always enter a before-link through the positive conclusion. So, because of the before-link, the path is not linear (both premises, positive ones, are likely to be the next on the path), and we define the *first branch* as the path from the positive premise of the before-link at the beginning of the arrow, and the *second branch* as the path from the other premise.

Then, when adding a new node on the tree, its mother-node is the the last node met on the same branch of the path (in a before-link, both premises are on the same branch as the conclusion, but they are themselves on different branches; other connectives do not create branches). So that if its mother-node's adress is p , then the new node's adress is $p \cdot j$ with $j \in \mathbb{N}^*$ and for all i such that $0 < i < j$, $p \cdot i \in D_\gamma$.

Then, we state the algorithm as follows:

1. Enter the net through the only output (so, if X is the output, $0 \in D_\gamma$, and $(0, X) \in \gamma$).
2. Follow the path defined by the positive polarities until reaching an atom (when a before-link is crossed, first choose the premise at the beginning of the red arc) and cross it. If its conclusions are X and X^\perp , we define its adress p as precised above (wrt the branches) and $p \in D_\gamma$ and $(p, X) \in \gamma$.
3. a) if the input is lexicalized, then lexicalize the last written node of the tree under construction (if the lexicalization is x , we then add $p \cdot 1 \in D_\gamma$ and $(p \cdot 1, x) \in \gamma$). Either there is no more link after, or this input is premise of a par-link whose other polarities are negative. In both cases, come back to the last before-link the path did not go through the two branches and make as in 2.
b) else just follow as in 2
4. Stop when the path joined all the atoms.

The next section will show that every SLIPN built from elementary SLIPNs can be read such a way and that the path cross every axiom-link.

- Remark 4.* 1. This reading provides a unique correspondance between any axiom link of the SLIPN and a node (labeled by a non terminal symbol). Moreover, if the negative conclusion of an axiom-link is pendant and not lexicalized, then it corresponds to a terminal node of the tree.
2. Two different SLIPNs can have the same reading. It underlines the importance of making precise a base of elementary SLIPNs (corresponding to the elementary trees of a given TAG).

4.3 From SLPINs, Back to Trees

We now have both a mapping from trees to SLIPNs, and a mapping from SLIPNs to trees. It remains us to see if the composition of these mappings gives the identity.

This consists in three steps:

1. check that the reading of a SLIPN corresponding to an elementary tree is the same as the elementary tree;
2. check that the reading of a SLIPN corresponding to the substitution between two trees is the resulting tree;
3. check that the reading of a SLIPN corresponding to the adjunction between two trees is the resulting tree.

Moreover, given a TAG, the basic bricks we consider are SLIPNs corresponding to elementary trees of this TAG. Then the only way to build new SLIPNs is binding them with cut between unlabeled conclusions.

Let us remind the definition of subtrees and supertrees as in [7]

Definition 14. *Let γ be a tree and $p \in D_\gamma$. Then*

$$\gamma/p = \{(q, X) | (p \cdot q, X) \in \gamma, q \in J^*\}$$

$$\gamma \backslash p = \{(q, X) | (q, X) \in \gamma, p \not\prec q\}$$

γ/p is called the subtree of γ at p and $\gamma \backslash p$ is called the supertree of γ at p . Further, for $p \in J^*$

$$p \cdot \gamma = \{(p \cdot q, X) | (q, X) \in \gamma\}$$

Property 2. $\gamma = \gamma \backslash p \cup p \cdot (\gamma/p)$ for every tree γ and $p \in D_\gamma$.

Remark 5. If the reading of a SLIPN Π gives γ , and if we make the path begin at any positive conclusion of an axiom link that corresponds to the node at adress p in γ , then the reading algorithm returns γ/p . And of course, the reading of Π , with a pruning at the same axiom link returns $\gamma \backslash p$.

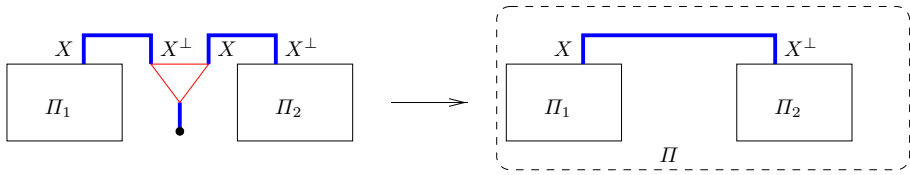


Fig. 4. Substitution

Elementary Reading. Let γ be an elementary tree. To prove the reading of the SLIPN corresponding to γ being γ itself, we simply proceed by induction, with the same steps as for the building of elementary SLIPNs.

Substitution. Let us consider two SLIPNs Π_1 and Π_2 , whose readings are respectively γ_1 and γ_2 . We assume Π_1 was built (with cuts) from elementary SLIPNs and Π_2 is an elementary SLIPN itself.

Proposition 3. *If a negative (not labeled) atomic conclusion of Π_1 and a positive atomic conclusion of Π_2 support a cut-linking, then the corresponding terminal node of γ_1 accepts a substitution by the root of γ_2 . And the reading of the new SLIPN, after cut-elimination, corresponds exactly to the resulting tree.*

Proof. Let p be the address of X in the reading γ_1 of Π_1 , where X corresponds to the axiom link of figure 4 (on the left). The address of X in the reading γ_2 of Π_2 is 0 (the root node) and the reading of γ_2 starts at this X axiom-link. On the other hand, the reading of γ_1 stops at X for its branch. After the cut and the cut-elimination, the reading of the new SLIPN (on the right of figure 4) starts at the output, which also was the output of Π_1 , and continues like for γ_1 until the new X axiom-link is reached. Its address in the new tree γ is also p . There, the reading of γ_2 takes place. So that, as defined in the algorithm, if γ is the reading of the new SLIPN,

$$\forall q \in D_{\gamma_2}, \gamma(p \cdot q) = \gamma_2(q)$$

and nothing changes for the remaining reading: it is the same as for γ_1 (because $\gamma_1 = \gamma \setminus p$). Then the reading γ of Π is such that

$$\gamma = \gamma_1 \cup p \cdot \gamma_2$$

which corresponds to the definition of the substitution of the $\gamma_1(p)$ node with γ_2 . \square

Adjunction. As above, let us consider two SLIPNs Π_1 and Π_2 , whose readings are respectively γ_1 and γ_2 . We assume Π_1 was built (with cuts) from elementary SLIPNs and Π_2 is an elementary SLIPN itself.

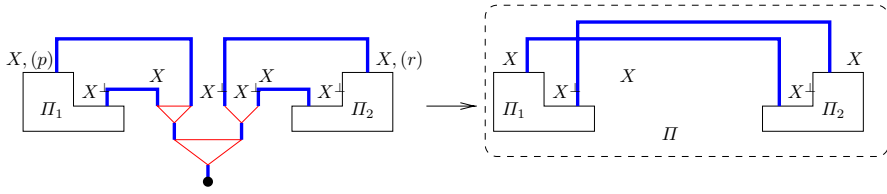


Fig. 5. Adjunction

Proposition 4. *If a tensor-gate of Π_1 and the unique positive conclusion of Π_2 support a cut-linking, then the node corresponding to the tensor-gate on γ_1 accepts an adjunction of γ_2 . And the reading of the new SLIPN, after cut-elimination, corresponds exactly to the resulting tree.*

Proof. In the following, when dealing with γ_1 , we speak about the reading of Π_1 without the tensor-gate.

Let us consider Π on figure 5 (the SLIPN on the right). The input of Π is the same as Π_1 . As the positive conclusion of an axiom-link always occurs before the negative conclusion in the path, then new tree γ from Π , after reaching X in Π_1 cross the axiom-link to Π_2 , so that $\gamma/p \neq \gamma_1/p$ but $\gamma \setminus p = \gamma_1 \setminus p$ (see remark 5). Yet, The X^\perp on Π_2 is the other conclusion of the starting axiom-link for γ_2 . So that at the p address, for γ , we read γ_2 . Then for every $q \in D_{\gamma_2}$, $\gamma(p \cdot q) = \gamma_2(q)$.

Moreover, reaching the X of Π_2 , the path does not stop but continue with the remaining part of γ_1 , namely γ_1/p . So that at the new address of X of Π_2 in γ we add γ_1/p . And the new address of X in γ is $p \cdot r$ (with r the address of X^* in γ_2).

Then

$$\gamma = \gamma_1 \setminus p \cup p \cdot \gamma_2 \cup p \cdot r \cdot \gamma_1 / p$$

which is the definition of the adjunction of γ_2 on γ_1 at X . \square


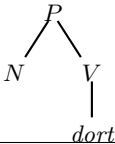
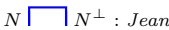
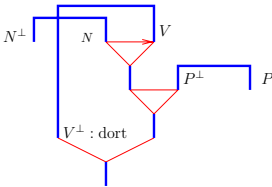
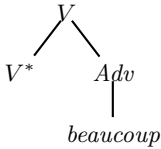
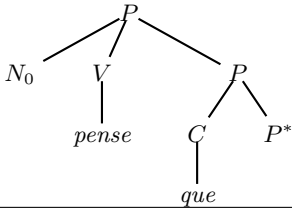
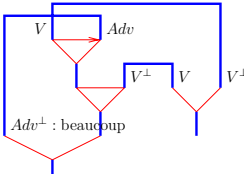
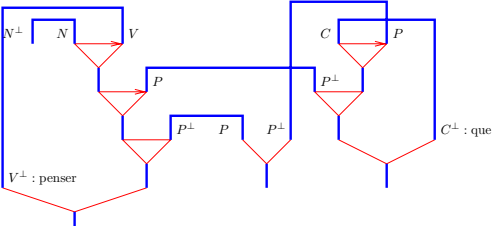
Eventually, we can state the next propositions:

Proposition 5. *Every derived tree (from an elementary tree lexicon) corresponds to the reading of a SLIPN, the latter resulting from Cut operations between SLIPNs corresponding to the elementary trees of the lexicon.*

Reciprocally, with a lexicon of elementary SLIPNs corresponding to trees, with the restriction of Cut operations on formulas that are not lexicalized, the reading of the resulting SLIPNs are the derived trees.

Proof. This is immediate after the previous propositions. \square

Table 5. Lexicon

	Jean(T_1)	Dort(T_2)
Trees		
SLIPNs		
	Beaucoup(T_3)	Pense que(T_4)
Trees		
SLIPNs		

5 Examples

5.1 Substitution and Adjunction

First let us define from the lexicon of the TAG the corresponding elementary SLIPNs. We assume the lexicon of table 5. This lexicon can yields the trees of figure 6 (for the first tree: substituting N in T_2 with T_1 , then adjoining T_3 on the result. For the second tree: continue with the adjunction of T_4). But we can also make this derivation on the SLIPNs as shown in the figures 7 and 8.

Let us see how to read the SLIPN of figure 7(e), and obtain the derived tree of figure 6(a): first the path enters the net through the atom P (the unique

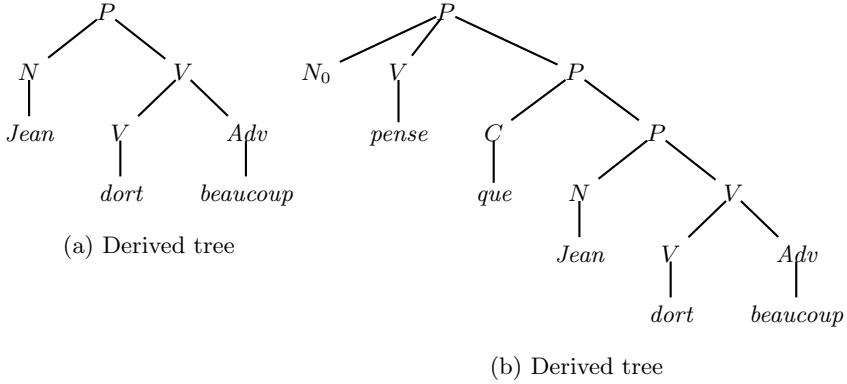


Fig. 6. Resulting trees

output) and marks P as a node. Then it follows the positive polarities and reaches a before-link with two branches. It first chooses the positive formula at the beginning of the red (regular) arc and crosses an axiom-link. There is a negative lexicalized atom. So on the tree we add a lexicalized (*Jean*) node N . Then doing the same with the other premise of the before-link we get a new atom V and the negative conclusion of the axiom-link is not lexicalized. So we have a junction which will produce new branches under the V node. They are two simple branches: one is a (lexicalized by *dort*) V , and the other is a (lexicalized by *beaucoup*) Adv .

To have a deeper adjunction, let us continue with the adjunction of T_4 . Figure 8 shows the different steps of the operation. But we leave the reader check that polarizing the resulting SLIPN of figure ?? and reading it leads to the tree of figure 6(b).

5.2 A Formal Language

As in the previous section, we first define the lexicon of table 6. Note that in this lexicon, the tensor gate belongs to the lexical item, so that we shall never use a tensor-gate addition during a derivation.

We only initiate the use of this lexicon with an adjunction of T_2 on another instance of T_2 (figure 9(a)), then an adjunction on T_1 , resulting in the SLPIN of figure 9(c). At every adjunction on T_2 , a new tensor-gate appear (provided by T_2), as the former (on the derived SLIPN) disappears with the adjunction operation (the cut between the par-gate and the tensor gate of T_2). As the reader can check, the reading actually corresponds to our expectations and generates the word *aabbccdd*.

Thus, without splitting any axiom-link and adding any tensor-gate during the derivation, we can generate the language $\{a^n b^n c^n d^n\}$.

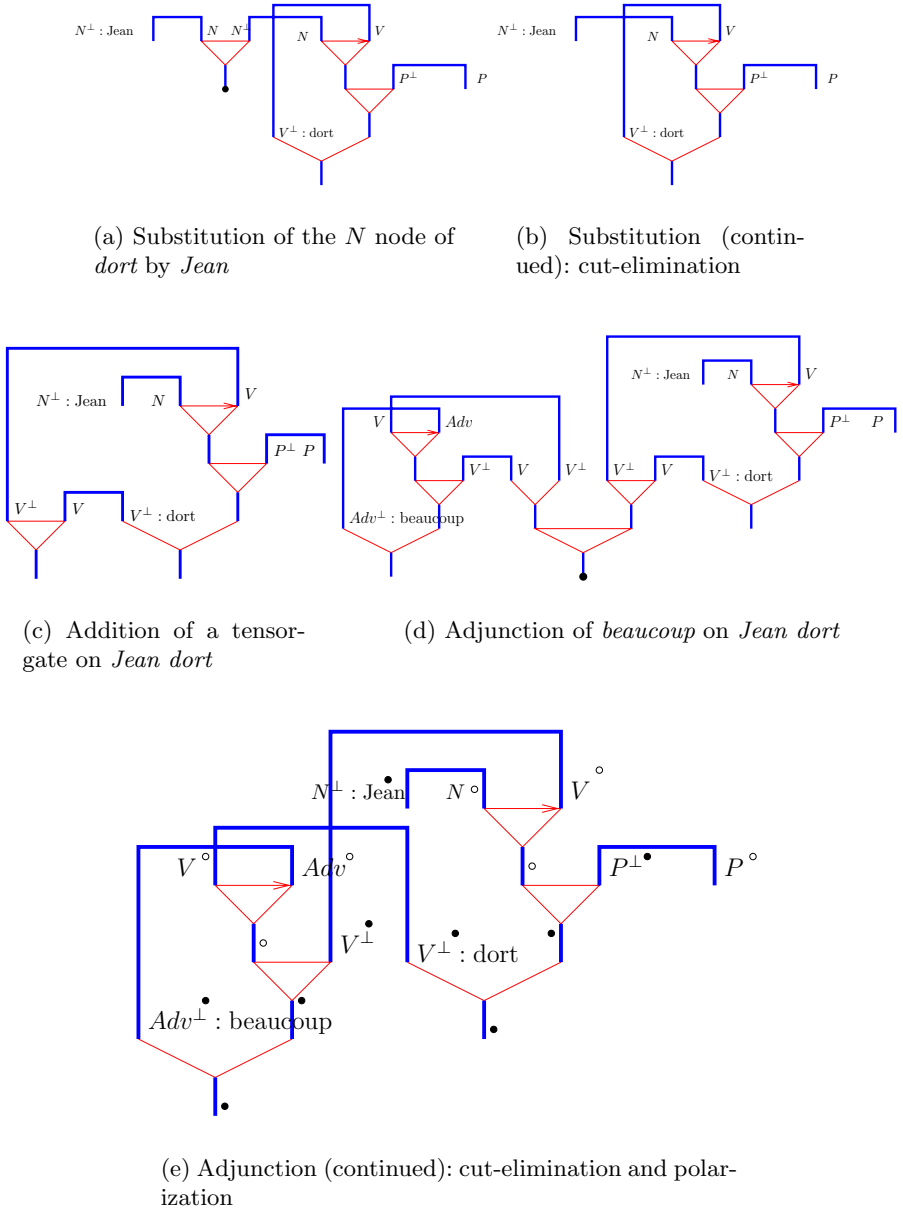
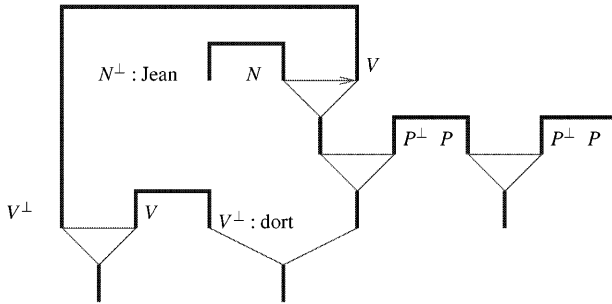
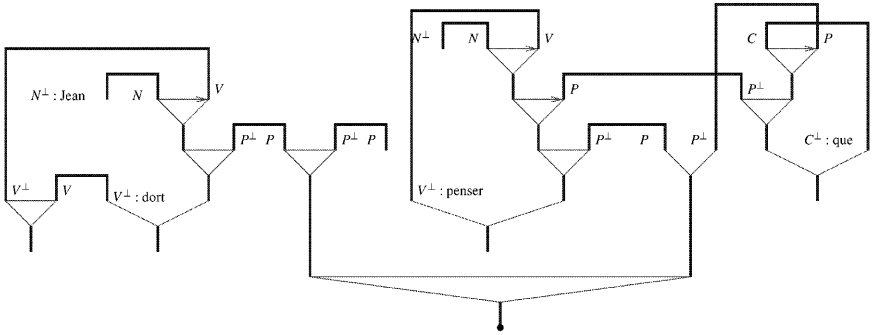
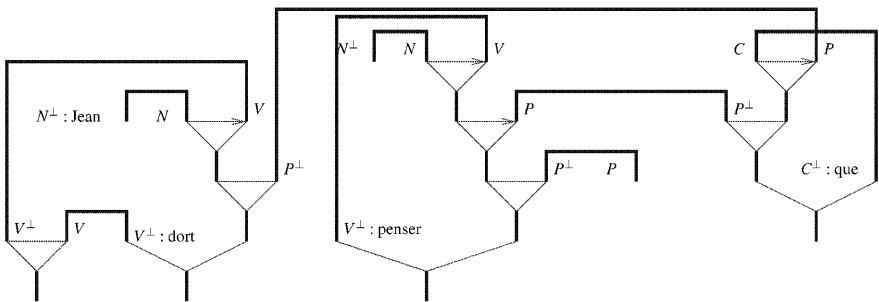


Fig. 7. Operating on SLIPNs

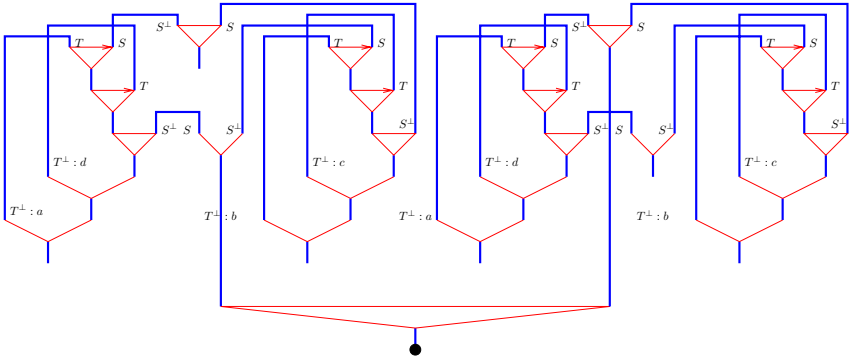


(a) Addition of a tensor-gate

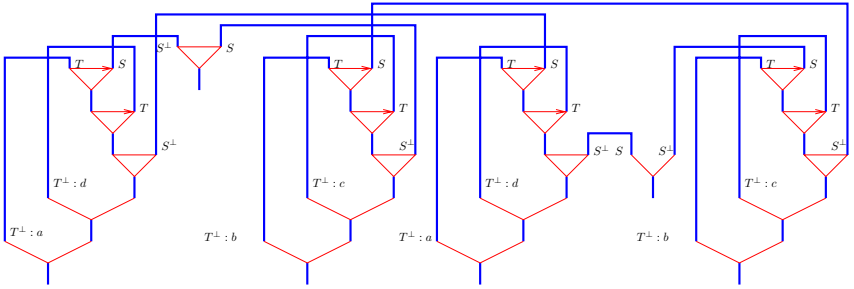
(b) Adjunction of *pense que* on *Jean dort beaucoup*

(c) Adjunction (continued): cut-elimination

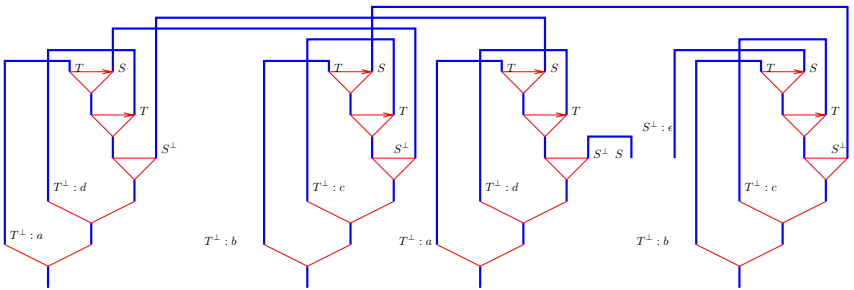
Fig. 8. Operating on SLIPNs (continued)



(a) Adjunction of T_2 on another instance of T_2



(b) Cut-elimination



(c) Adjunction on T_1 and cut-elimination

Fig. 9. Generating $aabbccdd$

Table 6. Lexicon

	Empty (T_1)	Main tree (T_2)
Trees	$\begin{array}{c} S \\ \\ \epsilon \end{array}$	
SLIPNs		

Conclusion

Using a restricted fragment of pomset intuitionistic proof-nets, we showed how to generate the same language as a TAG. This indicates how a more generic formalization (namely [11]’s one) allows both keeping generative power and dealing with some linguistic phenomena not by lexical rewriting rules on trees, but by lexical definitions. For instance, we can compare the modeling of clitics in [1] or in [11].

We also want to underline that we do not really use the partial order capabilities of pomset proof-nets: the before-links arrange totally the atoms in order. Of course, this results straightforwardly from the fact that the order in the trees is total, so that the same occurs in the SLIPNs with respect to the before-links.

Moreover, we use both commutative and non-commutative connectors, and the building of the path defines the order of the lexical items. The path performs the splitting of the sequent required in the rules (especially the adjunction rule) of [3].

Finally, to know how to express the semantics, at least two possibilities arise: to see it as for intuitionistic proof-nets [5], or as having an alternative expression like with the derivation trees (trees that track the operations performed during a derivation).

References

1. Abeillé, A. *Les nouvelles syntaxes*. Armand Colin Éditeur, Paris, Armand Colin Éditeur, 103, boulevard St-Michel - 75240 Paris, 1993.
2. Abrusci, V. M. Non-commutative proof nets. In Girard, J.-Y., Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 271–296. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.
3. Abrusci, V. M., C. Fouqueré, and J. Vauzeilles. Tree adjoining grammars in non-commutative linear logic. In Retoré, C., editor, *LACL'96*, volume 1328 of *LNCS/LNAI*, pages 96–117. Springer-Verlag Inc., New-York, USA, September23–25 1996.
4. Girard, J.-Y. Linear logic. In *Theoretical Computer Science*, 50:1–102, 1987.
5. de Groote, P. and C. Retoré. On the semantic readings of proof-nets. In Geert-Jan Kruijff, G. M. and D. Oehrlé, editors, *Formal Grammar*, pages 57–70. FoLLI, Prague, August 1996.
6. Joshi, A. K. and S. Kulick. Partial proof trees, resource sensitive logics and syntactic constraints. In Retoré, C., editor, *LACL'96*, volume 1328 of *LNCS/LNAI*, pages 21–42. Springer-Verlag Inc., New-York, USA, September23–25 1996.
7. Joshi, A. K., L. S. Levy, and M. Takahashi. Tree adjunct grammars. In *Journal of Computer and System Sciences*, 10(1):136–163, 1975.
8. Lamarche, F. Proof nets for intuitionistic linear logic: Essantial nets. Technical report, Imperial College, 1994.
9. Lecomte, A. and C. Retoré. Pomset logic as an alternative categorical grammar. In *Formal Grammar*. Barcelona, 1995.
10. Lecomte, A. and C. Retoré. Words as modules: a lexicalised grammar in the framework of linear logic proof nets. In *International Conference on Mathematical Linguistics*. John Benjamins, Tarragona, Spain, 1996.
11. Lecomte, A. and C. Retoré. Logique de ressources et réseaux syntaxiques. In Genthial, D., editor, *TALN'97*, pages 70–83. Pôle langage naturel, Grenoble, 12-13June 1997.
12. Retoré, C. *Réseaux et séquents ordonnés*. Ph.D. thesis, University of Paris VII, 1993.
13. Retoré, C. Perfect matchings and series-parallel graphs: multiplicatives proof nets as r&b-graphs. In Girard, J.-Y., M. Okada, and A. Scedrov, editors, *Linear Logic 96 Tokyo Meeting*, volume 3 of *Electronic Notes in Theoretical Computer Science*. april 1996.
14. Retoré, C. Pomset logic: a non-commutative extension of classical linear logic. In *TLCA'97*, volume 1210 of *LNCS*, pages 300–318. 1997.
15. Schabes, Y., A. Abeille, and A. K. Joshi. Parsing strategies with 'lexicalized' grammars. In *12th International Conference on Computational Linguistics (COLING'88)*, volume 2, pages 579–583. 1988.

Lambek Calculus Proofs and Tree Automata

Hans-Joerg Tiede*

Illinois Wesleyan University
Department of Mathematics and Computer Science
P.O. Box 2900
Bloomington, IL, 61702-2900, USA
`htiede@titan.iwu.edu`, <http://www.iwu.edu/~htiede>

Abstract. We investigate natural deduction proofs of the Lambek calculus from the point of view of tree automata. The main result is that the set of proofs of the Lambek calculus cannot be accepted by a finite tree automaton. The proof is extended to cover the proofs used by grammars based on the Lambek calculus, which typically use only a subset of the set of all proofs. While Lambek grammars can assign regular tree languages as structural descriptions, there exist Lambek grammars that assign non-regular structural descriptions, both when considering normal and non-normal proof trees. Combining the results of Pentus (1993) and Thatcher (1967), we can conclude that Lambek grammars, although generating only context-free languages, can extend the strong generative capacity of context-free grammars. Furthermore, we show that structural descriptions that disregard the use of introduction rules cannot be used for a compositional semantics following the Curry-Howard isomorphism.

1 Introduction

In this paper, we investigate the strong generative capacity of Lambek grammars, using natural deduction proof trees as the notion of structure assigned by Lambek grammars to strings they generate.

This approach differs from Buszkowski's (1997) definition of structure which he formalized in terms of functor/argument structures (f -structures) and phrase structures (p -structures). His definition led, in the case of Lambek grammars, to some unintuitive results, most notably the structural completeness theorem, which states that any binary tree that can be assigned to a string at all, can be assigned as a structural description of that string by any Lambek grammar that can generate that string.

The strong generative capacity of Lambek grammars fell into disrepute because of that result (cf. Steedman, 1993), which caused some researchers of proof theoretic grammars to turn their primary attention towards systems that do not

* The results of this paper are contained in my dissertation (Tiede, 1999). I would like to thank my thesis advisor Larry Moss, Johan van Benthem, Christian Retoré, and two anonymous referees for comments on this paper. All remaining errors are the author's.

suffer from this defect, like non-associative Lambek grammars (cf. Moortgat, 1997). However, Buszkowski had also shown that non-associative Lambek grammars cannot extend the strong generative capacity of context-free grammars (Buszkowski, 1997) using his definition of structure. Thus, we arrive at a problem for proof theoretic grammars: either they do not extend the strong generative capacity of context-free grammars, or they do so in a meaningless way.¹

The main result of this paper is that we can avoid the conclusion about the strong generative capacity of Lambek grammars if we consider natural deduction proof trees as the structure assigned by Lambek grammars, since they extend the strong generative capacity of context-free grammars but, in the case of normal form proofs, are not structurally complete. In addition, we show that the notions of p -structures and f -structures cannot be used for assigning meanings to strings using the Curry-Howard isomorphism.

In this paper, we assume a certain familiarity with proof theoretic methods in mathematical linguistics, which are covered in Buszkowski's (1997) survey article.

2 Proof Theoretic Grammars

Proof theoretic grammars can be described as formal grammars which assign to each element of a terminal vocabulary Σ a finite set of formulas built from a countable set of propositional variables using the connectives $/$ and \backslash . We will write

$$\mathbf{a} : A$$

if $\mathbf{a} \in \Sigma$ is assigned the formula A by the grammar. In addition, proof theoretic grammars have a distinguished atomic propositional variable, typically s , for sentence, and the following axioms and rules of inference:

$$A \quad [ID]$$

$$\frac{A/B \quad B}{A} /E \quad \frac{B \quad B \backslash A}{A} \backslash E$$

$$\frac{\begin{array}{c} [B] \\ \vdots \\ A \end{array}}{A/B} /I \quad \frac{\begin{array}{c} [B] \\ \vdots \\ A \end{array}}{B \backslash A} \backslash I$$

Basic categorial grammars, as formalized by Ajdukiewicz and Bar-Hillel, do not use the rules $[/I]$ and $[\backslash I]$. Lambek grammars use all of the above rules.

¹ However, the complexity of the strong generative capacity of modal Lambek grammars appears to be still open.

A string $a_1 \cdots a_n \in \Sigma^*$ is generated by a proof theoretic grammar if there are formulas A_1, \dots, A_n , such that $a_i : A_i, 1 \leq i \leq n$, and

$$A_1, \dots, A_n \vdash s$$

using the rules of inference allowed by the particular grammar.

Gaifman's, Buszkowski's, and Pentus' studies of the weak generative capacity of categorial grammars, i.e. the string languages generated by categorial grammars, can be summarized as follows: basic categorial grammars, non-associative Lambek grammars and associative Lambek grammars generate precisely the context-free languages (cf. Bar-Hillel, 1964; Buszkowski, 1997; Pentus, 1993). These results raise the question whether there are advantages to using categorial grammars rather than context-free grammars. For all practical purposes, context-free grammars certainly are easier to work with than Lambek grammars; for example the only known polynomial time algorithm for membership in the language generated by a Lambek grammar is to translate the Lambek grammar into its corresponding context-free grammar, using Pentus' construction, and checking membership using that context-free grammar (cf. Finkel and Tellier, 1996).

In defense of Lambek grammars van Benthem (1995) pointed out that they, even if their generative capacity would only equal that of context-free grammars, surpass context-free grammars in terms of their strong generative capacity, i.e. in terms of what structures they can assign to the strings they can generate. However, in 1986 Buszkowski proved a theorem about the strong generative capacity of Lambek grammars that, while affirming that the strong generative capacity of Lambek grammars extends that of context-free grammars, shows that they do so in a way that trivializes the notion of strong generative capacity. He showed that any binary tree that can be assigned to a string at all, can be assigned as a structural description of that string by any Lambek grammar that can generate that string. Buszkowski called this property of Lambek grammars "structural completeness."

However, upon close inspection of Buszkowski's proof, it becomes apparent that the structural completeness theorem depends on the particular way in which "structure assigned by a Lambek grammar to a string it generates" is defined. In particular, the definition employed by Buszkowski does not take into account the structure of proof trees, which should be at the center of this definition, since, when Lambek grammars are employed for syntactic descriptions, the proof trees form the basis of a compositional semantics of natural languages, based on the Curry-Howard isomorphism.

3 Strong Generative Capacity

In mathematical linguistics, the notion of strong generative capacity is used to distinguish between the mere ability of a grammar to generate a language and the structure that it assigns to the strings in its language. The strong generative capacity of a grammar is the set of structures (derivation trees, in the case of

context-free grammars) that are assigned by the grammar to the language it generates.

The notion of strong generative capacity provides a formal model of the linguistic notion of structure. In particular, the derivation trees of context-free grammars were intended to provide a formal model of the notion of constituent structure. Furthermore, the principle of compositionality (cf. Janssen, 1997) stipulates that semantic representations be defined on structures assigned to strings, rather than on the strings themselves.

Recent research in a wide variety of grammar formalisms has stressed the importance of their strong generative capacity. This has resulted in a renewed interest in tree formalisms, such as tree automata, tree grammars, and logical formalisms for the description of trees (see Rogers, 1998 and most of the contributions to this volume).

In the context of categorial grammar, Buszkowski has established a variety of results concerning strong generative capacity. However, he uses a definition of structure assigned by a grammar to strings that is different from the one used here. The main difference between the two definitions is that Buszkowski considers structures to be *determined* by proofs, whereas here the proofs *are* the structures. In the case of classical categorial grammar, the two definitions coincide. However, this is not the case for grammars that can use introduction rules, as Buszkowski's definition essentially only considers the elimination part of the proof tree.

In addition to Buszkowski's results, there have also been related investigation in the complexity of the λ -terms associated with proofs of the Lambek calculus (van Benthem, 1987; Moerbeek, 1991). However, these were concerned with studying the λ -terms as strings, rather than as structured objects like trees.

The investigation of the strong generative capacity of grammar generalizes the theory of formal (string) languages to tree or term languages. Terms represent derivation trees of grammars. We review some of the basic concepts of tree automata and tree languages, for a more complete reference see Gécseg and Steinby (1984).

Definition 1. *A tree is a term over a finite signature Σ containing function and constant symbols. The set of n -ary function symbols in Σ will be denoted by Σ_n . The set of all terms over Σ is denoted by T_Σ ; a subset of T_Σ is called a tree language or a forest. The yield of a tree t , denoted by $\text{yield}(t)$, is defined by*

$$\begin{aligned}\text{yield}(c) &= c, c \in \Sigma_0 \\ \text{yield}(f(t_1, \dots, t_n)) &= \text{yield}(t_1) \cdots \text{yield}(t_n), f \in \Sigma_n, n > 0.\end{aligned}$$

Most of the research on tree languages has been conducted with respect to regular tree languages. There are many equivalent definitions of regular tree languages; we will use regular tree grammars.

Definition 2. *A regular tree grammar is a system $\langle \Sigma, \Gamma, S, \Delta \rangle$, such that Σ is finite signature (the terminal vocabulary), Γ is a finite set of 0-ary non-terminals,*

$S \in \Gamma$ is the distinguished start symbol, Δ is finite set of rules of the form,

$$A \rightarrow t$$

with $A \in \Gamma$ and $t \in T_{\Sigma \cup \Gamma}$. These rules are interpreted as rewrite rules, just as in the case of string grammars.

Definition 3. A tree language is regular if it is generated by a regular tree grammar.

Regular tree languages have Myhill-Nerode characterization similar to that of string languages. The Myhill-Nerode theorem for tree languages will be our primary tool for proving tree languages non-regular.

Definition 4. A context is a term over $\Sigma \cup \{x\}$ containing the zero-ary term x exactly once.

Theorem 1. (Myhill-Nerode): A subset $S \subseteq T_\Sigma$ is regular iff the equivalence relation, $\sim_S \subseteq T_\Sigma \times T_\Sigma$, defined by

$$\tau_1 \sim_S \tau_2$$

iff for all contexts $\tau(x)$,

$$\tau[x \mapsto \tau_1] \in S \text{ iff } \tau[x \mapsto \tau_2] \in S$$

determines finitely many equivalence classes, where $[x \mapsto \tau_i]$ denotes substituting x with τ_i .

PROOF. See Kozen 1997. ■

The relation between regular tree grammars and the derivation trees of context-free string grammars was established by Thatcher:

Theorem 2. (Thatcher, 1967) If S is the set of derivation trees of some context-free grammar, then S is regular.

The converse of this theorem does not hold, as there are regular tree languages that are not derivation trees of context-free grammars. However, the yield of any regular tree language is a context-free (string) language.

4 Proof Tree Languages

In general, we will consider structures generated by categorial grammars as term languages over the signature $\Sigma = \{[/E], [\backslash E], [/I], [\backslash I], c\}$, where

- c is a zero-ary function symbol,
- $[\backslash I]$ and $[/I]$ are unary function symbols,
- $[\backslash E]$ and $[/E]$ are binary function symbols.

The terms over this signature represent proof trees that neither have information about the formulas for which they are a proof nor the strings that are generated by a grammar using this proof. The reason for using this, somewhat impoverished notion of structure is that it provides an abstract definition using a finite signature, making it possible to consider the proof trees of every Lambek grammar as a language over this signature.

It should be noted that these terms represent proofs unambiguously; we can decide solely on the basis of the unlabeled proof trees which leaf is withdrawn by an application of $[\backslash I]$ or $[/I]$ without annotating the proof, since they withdraw the leftmost or the rightmost uncanceled hypothesis, respectively. However, this is the case only for the Lambek calculus, because of its lack of structural rules. For other logics, it would be necessary to annotate the proof trees at least with which premise is withdrawn and how many instances of this premise are withdrawn, which would not be possible to do with a finite signature.

As a matter of fact, these proof trees can be used as a variable free formalization of the bi-directional λ -calculus (cf. Wansing, 1993). Although we are not concerned with the labelling of the leaves with formulas here, as that would necessitate the introduction of infinitely many labels for leaves, it should be noted that we can use a principal type algorithm to provide an unlabeled proof tree with the principal pair for which it is a proof, i.e. given an unlabelled proof tree t , we can compute a pair $\langle \Gamma, \alpha \rangle$, such that t is a proof tree of

$$\Gamma \vdash \alpha$$

and any other pair $\langle \Delta, \beta \rangle$ such that t is a proof tree of

$$\Delta \vdash \beta$$

is a substitution instance of $\langle \Gamma, \alpha \rangle$. This algorithm can be directly modeled after the Hindley-Milner algorithm for the simply typed λ -calculus (cf. Hindley, 1997).

In order to study this formalization of proof trees, we need to consider how many cancelled and uncanceled assumptions are present in a proof tree, which is defined inductively:

Definition 5. *c contains one uncanceled assumption and no cancelled assumptions. If τ_1 and τ_2 contain m and n uncanceled assumptions and k and l cancelled assumptions, respectively, then $[/E](\tau_1, \tau_2)$ and $[\backslash E](\tau_1, \tau_2)$ contain $m + n$ uncanceled and $k + l$ cancelled assumptions, and, for $m > 1$, $[/I](\tau_1)$ and $[\backslash I](\tau_1)$ contain $m - 1$ uncanceled and $k + 1$ cancelled assumptions.*

4.1 Structures Assigned by Categorical Grammars

The structures that are assigned by classical categorical grammars are defined with respect to the category that they assign to strings.

Definition 6. *If $a \in \Sigma$ has category A (i.e. $a : A$), then a is assigned the structure c . If $w, v \in \Sigma^*$ are assigned the categories A/B and B and structures t_1 and*

t_2 , respectively, then wv has category A and is assigned the structure $[/E](t_1, t_2)$. If they are assigned categories B and $B \setminus A$, respectively, then vw has category A and structure $[\setminus E](t_1, t_2)$.

It is easy to show that the structures assigned by categorial grammars to strings they generate are regular tree languages. In fact, this immediately follows from the construction of Gaifman's theorem. It has also been proved directly by Buszkowski using the Myhill-Nerode theorem.

An even easier proof can be given using regular tree grammars.

Proposition 1. *The set of derivation trees of a categorial grammar is regular.*

PROOF. Since the grammar assigns a finite set of category, the set of all subformulas of assigned categories is finite. We will use this set as the non-terminals, i.e. the non-terminals are

$$\Gamma = \{A \mid A \in \text{sub}(B), \exists a \in \Sigma, a : B\}$$

where $\text{sub}(B)$ denotes the set of subformulas of B . The rules of the grammar contains for all $A, B, A/B \in \Gamma$

$$A \rightarrow [/E](A/B, B)$$

and for all $A, B, B \setminus A \in \Gamma$

$$A \rightarrow [\setminus E](B, B \setminus A).$$

Furthermore, if $a : A$, for some $A \in \Gamma$, then

$$A \rightarrow c$$

is a rule. ■

4.2 Structures Determined by Proofs

As was pointed above, in the case of Lambek grammars Buszkowski introduced a notion of structure assigned by a Lambek grammar that differs from the one used here. In order to define his notion of structure, he notes that the Lambek calculus can be axiomatized using as axioms all the

$$A \vdash B$$

such that

$$A \vdash B$$

is provable in the Lambek calculus, and as rules of inference $[/E]$ and $[\setminus E]$. This is true, since if

$$A_1, \dots, A_n \vdash B$$

is derivable in the Lambek calculus, then so is

$$A_1 \vdash (\cdots ((B/A_n) \cdots) / A_2)$$

and

$$A_1, \dots, A_n \vdash B$$

is then derivable using $[/E]$,

$$A_i \vdash A_i,$$

and

$$A_1 \vdash (\cdots ((B/A_n) \cdots) / A_2).$$

Using this axiomatization, Buszkowski is able to assign to strings derivable by a Lambek grammar structures over the signature $[/E], [\backslash E]$ using the rules of how structures are assigned by categorial grammars with the additional rule: if $w \in \Sigma^*$ has category A , w is assigned structure t , and

$$A \vdash B$$

is derivable in the Lambek calculus, then w has category B and is assigned structure t .² The structures built from $[/E]$ and $[\backslash E]$ are called f -structures (for function/argument structures) by Buszkowski. Buszkowski also considers structures built from a single function symbol $[E]$, which are called p -structures (for phrase structures). p -structures are the homomorphic image of f -structures using the following mapping:

$$\begin{aligned} h(c) &= c \\ h([\backslash E](t_1, t_2)) &= h([/E](t_1, t_2)) = [E](h(t_1), h(t_2)). \end{aligned}$$

The first criticism concerning his definition is that it does not take into account the structure of the proof that actually establishes that a particular string is generated by a grammar. The second criticism is that his definition of structure leads to structural completeness. The last criticism of his definition is concerned with the question of whether it is possible to use Buszkowski's definition of structure as a basis for a compositional semantics based on the Curry-Howard isomorphism. I believe that this is an important point to consider, since the main purpose of the notion of structure in linguistics is to use it as a basis for semantics, i.e. to define a (compositional) function that assigns meanings on the basis of the structure assigned by some grammar. However, it is not possible to use Buszkowski's notion of structure for this purpose.

Proposition 2. *There is no function f from p -structures or f -structures to proof trees, such that every normal-form proof tree that is a proof of*

$$A_1, \dots, A_n \vdash B$$

is assigned to some p -structure or f -structure determined by

$$A_1, \dots, A_n \vdash B.$$

² To be precise, Buszkowski does not use the same notation as I do, however, both notations convey the same meaning.

PROOF. The easiest way to show this, is to demonstrate that already the base case fails, i.e. that there is a sequent

$$A \vdash A$$

that has two (or more) normal form proofs. If some symbol, $\mathbf{a} \in \Sigma$, is assigned A by a Lambek grammar, then there is only one p -structure or f -structure that can be assigned to \mathbf{a} .

Consider the following sequent which has two normal form proofs in the Lambek calculus:

$$(A / (A \backslash A)) \backslash A \vdash (A / (A \backslash A)) \backslash A.$$

The first proof is a simple application of ID :

$$(A / (A \backslash A)) \backslash A,$$

while the second is somewhat more complex:

$$\frac{\frac{\frac{[A] \quad [A \backslash A]}{A} \backslash E}{A / (A \backslash A)} / I \quad \frac{(A / (A \backslash A)) \backslash A}{A} \backslash E}{\frac{[A / (A \backslash A)] \quad \frac{A}{A \backslash A} \backslash I}{A} / E} \backslash I$$

Which completes the proof. ■

4.3 Proofs as Structures

If we want to consider proofs to be structures, we have two options to consider: either we consider every proof that establishes that a certain string is generated by a Lambek grammar to be the structures assigned to that string by that grammar, or we only consider those proofs that are in normal-form. It should be noted that in the second case neither the weak generative capacity nor the semantic expressibility of the grammar changes. We will consider both options, as they differ with respect to their formal properties.

We first consider the set of all proofs of the Lambek calculus, although no grammar could ever use the set of all proofs. Let us call the set of proofs of the Lambek calculus T_{Π} .

Theorem 3. *The set of well-formed proof trees of the Lambek calculus is not regular.*

PROOF. Consider the following context

$$\tau_0(x) = [\backslash I]([\backslash E](x, c)).$$

Any tree that can be substituted for x has to contain at least one uncanceled assumption. If it doesn't, then the term is not well-formed, as we cannot withdraw the premise to its left and arrive at a well-formed proof tree in the Lambek calculus. Let us generalize the above observation and define

$$\tau_{n+1}(x) = [\backslash I](\tau_n(x)).$$

Also, for all $n > 0$, let π_n be an arbitrary well-formed proof tree containing $n + 1$ uncanceled assumptions. Into any $\tau_n(x)$ we can only substitute a tree with at least $n + 1$ uncanceled assumptions. Therefore, for all k, l , such that $k \neq l$,

$$\pi_k \not\sim_{T_H} \pi_l,$$

because either

$$\tau_k[x \mapsto \pi_l] \notin T_H$$

or

$$\tau_l[x \mapsto \pi_k] \notin T_H.$$

This immediately implies that \sim_{T_H} determines infinitely many equivalence classes, since we have infinitely many pairwise non-equivalent terms. Therefore the set of proofs of the Lambek calculus is not regular. ■

Corollary 1. *The set of normal form proofs of the Lambek calculus is not regular.*

PROOF. In the above construction, choose π_n to be a proof with $n+1$ uncanceled assumptions and with no occurrence of either $[\backslash I]$ or $[/I]$. Substituting π_n for x in $\tau_n(x)$ gives a normal form proof tree, since there are no introductions followed by eliminations. Again we arrive at infinitely many equivalence classes, showing that the set is not regular. ■

However, in the above proofs we considered the set of all proof trees of the Lambek calculus. The complexity of proof trees of grammars based on the Lambek calculus, which only use a subset of the set of all proof trees, depends on whether we consider all the possible proof trees that a grammar can assign to the strings in its language or only those in normal form. The approach taken both in traditional linguistics and in formal language theory is to consider any two derivations of a string as different as long as they are not identical. However, we could also consider two derivations as identical if they are β - η -equivalent, mainly for semantic reasons.³ Such a restriction can change the complexity of the structures assigned by a grammar, as the following theorem illustrates.

Theorem 4. *There is a Lambek grammar G such that its set of non-normal proof trees is not regular, although its set of normal proof trees is regular.*

³ See Wansing (1993) for a definition of β - η -equivalence in the context of the Lambek calculus and van Benthem (1995) for a discussion of using normal proof trees as structural representations.

PROOF. Consider the grammar G which generates the regular (string) language a^+ :

$$\begin{aligned} a : S, \\ S/S. \end{aligned}$$

The normal trees of G are regular using the construction in proposition 1 and the fact that normal proof trees of this grammar do not use introduction rules (see below). To show that the set of all proof trees of G are not regular, we use the Myhill-Nerode theorem and construct a set of contexts

$$T = \{\tau_n(x) \mid n \in \mathbb{N}\}$$

and a set of trees

$$\Pi = \{\pi_n \mid n \in \mathbb{N}\},$$

such that for all n, m , such that $n \neq m$,

$$\pi_n \not\sim_G \pi_m,$$

where \sim_G denotes the Myhill-Nerode equivalence relation of the proof tree language of G .

We construct T by

$$\tau_n(x) = \tau(x)[x \mapsto [\backslash E]([ID], [\backslash E]([ID], \dots, [\backslash E]([ID], ([I]^n x)))],$$

with the number of $[\backslash E]$'s matching the number of $[\backslash I]$'s, and Π by

$$\begin{aligned} \pi_0 &= [/ E]([ID], [/ E]([ID], [ID])) \\ \pi_{n+1} &= [/ E]([ID], \pi_n). \end{aligned}$$

First, notice that, by the construction, for all n ,

$$\tau_n(x)[x \mapsto \pi_n]$$

is a proof of the grammar, but for all $k < n$,

$$\tau_n(x)[x \mapsto \pi_k]$$

is not, since π_k does not contain enough uncanceled assumptions. Therefore, for all k, l , such that $k \neq l$,

$$\pi_k \not\sim_G \pi_l,$$

which completes the proof. ■

It should be noted that the above construction does not preserve normal forms of proofs. Thus, it is a natural question to ask what the tree automata theoretic complexity of normal form proofs of a grammar is. This perspective brings with it an interesting approach to strong generative capacity, disregarding all derivations that can be counted as equal to some normal form proof tree.

The first observation is that there are Lambek grammars whose normal form proof trees are regular. The set of normal form proofs that can be used by a Lambek grammar that generates a finite language has to be finite, and therefore regular, by van Benthem's finiteness theorem (cf. van Benthem, 1995). Furthermore, any Lambek grammar that only assigns categories of the form $A, A/B, (A/B)/C$, with A, B, C atomic, has normal-form proof trees that do not use any introduction rules, which can be established by an easy induction on the height of the normal form proof tree. These proof tree languages are therefore equivalent to the proof tree languages of basic categorial grammars assigning the same categories. Since the proof tree languages of basic categorial languages are regular, this is another example of a regular subset of the set of all normal proof tree languages of Lambek grammars.

However, there are examples of normal form proof tree languages that are not regular.

Theorem 5. *There is a Lambek grammar such that its set of normal form proof trees is not regular.*

PROOF. Consider the following grammar that generates the regular (string) language a^+ :

$$\begin{aligned} a : S, \\ S / (A/A), \\ S / (S / (A/A)). \end{aligned}$$

The following proof trees are normal-form derivation trees for the strings a , aa , aaa , respectively:

$$\begin{array}{c} S \\ \hline \frac{S / (S / (A/A)) \quad S / (A/A)}{S} / E \end{array}$$

$$\begin{array}{c} \frac{S / (A/A) \quad \frac{\frac{[A/A] \quad \frac{[A]}{A} / E}{A/A} / I}{S} / E}{S / (S / (A/A)) \quad \frac{S}{S / (A/A)} / I} / E \end{array}$$

$$\frac{S / (S / (A/A)) \quad \frac{S}{S / (A/A)} / I}{S} / E$$

In general, normal-form proof trees of this grammar for strings of length $n > 2$ have the following form:

- they have a top part, consisting of $n - 1$ occurrences of A/A :

$$\frac{\frac{[A/A] \quad [A]}{A} /E}{A} /E$$

- a middle part:

$$\frac{S/(A/A) \quad \overline{A/A}}{S} /I$$

- and a bottom part, where the number of $[/I]$ matches the number of A/A in the top part:

$$\frac{\frac{S/(S/(A/A)) \quad \overline{S/(A/A)}}{S} /I}{S/(S/(A/A)) \quad S/(A/A)} /E$$

From this observation it is straightforward to conclude that the set of normal form proof trees is not regular. We can again construct an infinite set of contexts

$$\{\tau_n(x) \mid n \in \mathbb{N}\}$$

defined by

$$\begin{aligned} \tau_0(x) &= [/E](c, [/I]([/E](c, [/I]([/E](c, [/I](x)))))) \\ \tau_{n+1}(x) &= [/E](c, [/I](\tau_n(x))) \end{aligned}$$

and an infinite set of trees

$$\{\pi_n(x) \mid n \in \mathbb{N}\}$$

defined by

$$\begin{aligned} \pi_0 &= [/E](c, [/E](c, c)) \\ \pi_{n+1} &= [/E](c, \pi_n) \end{aligned}$$

such that

$$\tau_k(x)[x \mapsto \sigma]$$

is a normal-form proof tree that is used to establish that some string is generated by the grammar iff $\sigma = \pi_k$. It should be noted that these proofs are in normal-form, although they follow introductions by eliminations, however, the formula that has been introduced is the *minor* premise of the elimination rule. ■

Finally, we need to revisit the question of structural completeness. As was pointed out above, Lambek grammars that assign formulas of the form A , A/B , $(A/B)/C$, with A, B, C atomic, have regular proof tree languages. It is known that as a consequence of Gaifman's theorem every categorial grammar is equivalent to one that assigns formulas of the above form. Since basic categorial grammars are not structurally complete, we can conclude:

Theorem 6. *For every context-free language, there is a Lambek grammar that is not structurally complete which generates it, assuming that structures that are assigned are normal form proof trees.*

PROOF. This follows immediately from Pentus' theorem and the discussion in the preceding paragraph. ■

5 Conclusion

The main conclusion is that using normal form proof trees as structural descriptions that Lambek grammars assign to strings can extend the strong generative capacity of context-free grammars, but does not trivialize the notion of strong generative capacity. Furthermore, the use of proof trees as structural descriptions makes it possible to use structural descriptions for a compositional semantics, based on the Curry-Howard isomorphism, which is impossible if structures are used that disregard the use of introduction rules.

One of the topics for further research on strong generative capacity is to consider proof theoretic grammars based on extensions of the Lambek calculus, including those with additional additive or multiplicative connectives and modalized versions of the Lambek calculus. Some results in the case of modalized Lambek grammars with one residuation modality have been obtained by Jaeger (1998). In the case of modalized Lambek grammars with a variety of modalities and interaction postulates between them, there is one obstacle: Carpenter (1999) has shown that modal Lambek grammars can generate any recursively enumerable language. Thus, their normal form proof trees extend even the context-free tree languages, as the yield of any context-free tree language is an indexed language. Thus, it would be necessary to restrict modal Lambek grammars in such a way that they can only generate (at most) indexed languages or subclasses of the indexed languages. However, it appears to be an open problem how to restrict modal Lambek grammars in such a way.

Another question that would naturally fall into the study of the strong generative capacity of proof theoretic grammars is: given that Lambek grammars extend the strong generative capacity of context-free languages but generate only context-free string languages, what decidable properties of the strong generative capacity of context-free grammars are also decidable for Lambek grammars? For example the following properties of context-free grammars are decidable:

- strong equivalence of two context-free grammars: do two context-free grammars assign the same structures to the strings they generate?

- ambiguity: does a context-free grammar assign different structures to some string it generates?

Other questions that relate to this area are what kind of logic would be needed for a descriptive approach to proof trees of Lambek grammars, following Rogers (1998) work, and the relationship of proof trees to those of tree adjoining grammars, which is related to work by Joshi and associates (cf. the contribution by Joshi, Kulick, and Kurtonina to this volume).

References

1. Ajdukiewicz, K. Die syntaktische Konnexität. In *Studia Philosophica* 1:1-27, 1935.
2. Bar-Hillel, Y. *Language and Information*. Addison-Wesley, Reading, 1964
3. Benthem, J. van. Logical Syntax. In *Theoretical Linguistics* 14(2/3): 119-142, 1987.
4. Benthem, J. van *Language in Action*. MIT Press, Cambridge, MA, 1995.
5. Buszkowski, W. Proof Theory and Mathematical Linguistics. In Benthem, J. and A. ter Meulen (eds.) *Handbook of Logic & Language*. MIT Press: 683-736, Cambridge, 1997.
6. Carpenter, R. The Turing-completeness of multimodal categorial grammars. In *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*. Gerbrandy, J., M. Marx, M. de Rijke, and Y. Venema (eds.), Vossiuspers, Amsterdam University Press, 1999.
7. Finkel, A. and I. Tellier. A polynomial algorithm for the membership problem with categorial grammars. In *Theoretical Computer Science* 164: 207-221, 1996.
8. Gécseg, F. and M. Steinby. *Tree Automata*, Akadémiai Kiadó, Budapest, 1984.
9. Hindley, J. R. *Basic Simple Type Theory*. Cambridge University Press, Cambridge, 1997.
10. Jaeger, G. On the generative capacity of multi-modal Categorial Grammars. IRCS Report 98-25, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, 1998.
11. Janssen, T. Compositionality. In Benthem, J. van, and A. ter Meulen (eds.) *Handbook of Logic & Language*. MIT Press, Cambridge, 1997.
12. Kozen, D., *Automata and Computability*. Springer, Berlin, 1997.
13. Lambek, J. The Mathematics of Sentence Structure. In *American Mathematical Monthly* 65: 154-169, 1958.
14. Moerbeek, O. *Algorithmic and Formal Language Theoretic Aspects of Lambek Calculus*. Unpublished Master's Thesis, Dept. of Logic and Theoretical Computer Science, University of Amsterdam, 1991.
15. Moortgat, M. Categorial Type Logics. In Benthem, J. van, and A. ter Meulen (eds.) *Handbook of Logic & Language*. Cambridge: MIT Press: 93-178, 1997.
16. Pentus, M. Lambek grammars are context-free. In *Proc. 8th Annual IEEE Symposium on Logic in Computer Science.*, 1993.
17. Rogers, J. *A Descriptive Approach to Language Theoretic Complexity*. CSLI Publications, 1998.
18. Steedman, M. Categorial Grammar. In *Lingua* 90: 221-258, 1993.
19. Tiede, H.-J. *Deductive Systems and Grammars*. PhD Dissertation, Indiana University, 1999.
20. Thatcher, J. W. Characterizing Derivation Trees of Context-Free Grammars through a Generalization of Finite Automata Theory. In *J. Comput. System Sci.* 1, 317-322, 1967.
21. Wansing, H. *The Logic of Information Structures*. Springer Verlag, Berlin, 1993.

Grammars with Composite Storages

Christian Wartena

Universität Potsdam

Institut für Linguistik/Allgemeine Sprachwissenschaft

Postfach 601553, 14415 Potsdam, Germany

wartena@ling.uni-potsdam.de

<http://www.ling.uni-potsdam.de/~wartena>

Abstract. Linear indexed grammars (LIGs) can be used to describe nonlocal dependencies. The indexing mechanism, however, can only account for dependencies that are nested. In natural languages one can easily find examples to which this simple model cannot be applied straightforwardly. In this paper I will show that a formalism fitting better to linguistic structures can be obtained by using a sequence of pushdowns instead of one pushdown for the storage of the indices in a derivation. Crucially, we have to avoid unwanted interactions between the pushdowns that would make possible the simulation of a turing machine. [1] solves this problem for multi-pushdown automata by restricting reading to the first nonempty pushdown. I will argue that the corresponding restriction on writing is more natural from a linguistic point of view. I will show that, under each of both restrictions, grammars with a sequence of n pushdowns give rise to a subclass of the n th member of the hierarchy defined by [15,16], and therefore are mildly context sensitive.

1 Introduction

Nonlocal dependencies in natural languages can be described with linear indexed grammars (LIGs) ([4]) The indexing mechanism can only account for dependencies that are *nested*. This is due to the fact that the indices are stored as on a pushdown: the index introduced first is read last. In human languages one can easily find counterexamples to this simple model. Thus, the obvious step is to investigate similar formalisms using another way to store indices. Up to now these possibilities have scarcely been explored. [9] introduces *multiset valued* LIGs, that are not only an extension of LIGs w.r.t. the structure of the index stack. [17] gives extensions using pushdowns of pushdowns, which, however, seem to have no direct linguistic interpretation.

According to a number of recent linguistic theories we have to distinguish between different types of movement such as movement of heads, movement into argument positions and movement into other positions ([10]) or between types determined by the feature that licenses the movement (cf. [2,3]). Further evidence for distinguishing between different types of movement comes from psycholinguistic research. It can be observed that the cognitive difficulty to process a sentence does not directly depend on the number of dependencies that have to be

computed (at the same time) but rather on the number of similar dependencies (cf. [13,5]).

These ideas suggest a formalization as a kind of LIG with more than one pushdown of indices in a derivation. Such grammars would, however, generate all recursive enumerable sets, as is easy to verify. A restriction that makes a system with more than one pushdown less powerful is proposed in [1]. Here it is shown that automata with a sequence of pushdowns, restricted by the condition that a pushdown only can be read if all pushdowns to its left are empty, accept only context sensitive languages parsable in polynomial time. As I show below, this result can be carried over to grammars with similar storages for indices. I will define these grammar formalisms and give a characterization of the classes of generated languages in terms of *controlled* grammars. This results in an elegant new definition of the hierarchy established in [1]. It can be argued that the used restriction on the accessibility of the pushdowns is not plausible from a linguistic point of view. A similar condition restricting writing instead of reading seems much better suited. Using this restriction I will define a new hierarchy of languages. I will show that each class of languages of the hierarchy defined in [1] as well as each class of the new hierarchy is a subset of a class from the hierarchy defined in [16]. Since all languages of this hierarchy were shown to be mildly context sensitive, the languages generated by LIGs extended with a restricted sequence of pushdowns are mildly context sensitive as well.

2 Grammars with Storage

In order to study grammars with additional memory, it is useful to define storages independently of the grammars or automata using them.

Definition 1. A *storage type* is a 6-tuple $S = (C, C_0, C_F, P, F, m)$, where C is a set of configurations, $C_0 \subseteq C$ and $C_F \subseteq C$ the sets of initial and final configurations, respectively, P is a set of predicate symbols, F a set of instruction symbols. m is the meaning function, which associates every $p \in P$ with a mapping $m(p) : C \rightarrow \{\text{true}, \text{false}\}$ and every $f \in F$ with a partial function $m(f) : C \rightarrow C$.

The trivial storage type S_{triv} is defined by $S_{\text{triv}} = (\{c\}, \{c\}, \{c\}, \emptyset, \{\text{id}\}, m)$, where c is an arbitrary object and $m(\text{id})(c) = c$. An (ordinary) pushdown S_{pd} over some finite alphabet Γ is defined by $S_{\text{pd}} = (\Gamma^*, \{\epsilon\}, \{\epsilon\}, P, F, m)$ with $P = \{\text{top} = \gamma \mid \gamma \in \Gamma\}$ and $F = \{\text{push}(\gamma) \mid \gamma \in \Gamma\} \cup \{\text{pop}\} \cup \{\text{id}\}$ and for every $a \in \Gamma$ and $\beta \in \Gamma^*$,

$$\begin{aligned} m(\text{top} = \gamma)(a\beta) &= (a = \gamma) & m(\text{pop})(a\beta) &= \beta \\ m(\text{push}(\gamma))(a\beta) &= \gamma a\beta & m(\text{id})(a\beta) &= a\beta \end{aligned}$$

For the sake of convenience I will sometimes assume that there is an additional predicate **empty** which is true if the stack is empty and false otherwise. This predicate is in some sense superfluous since we could use a “bottom-of-stack” symbol and test whether this symbol is on top of the stack.

On the basis of this universal definition of storages it is possible to make a generalization of LIGs w.r.t. the organization of the indices. These generalized grammars, called context free linear S -grammars, were introduced in [17].

Definition 2. If $S = (C, C_0, C_F, P, F, m)$ is a storage type, then a *context free linear- S -grammar* (CFL- S -G) is a five-tuple $G = (N, \Sigma, R, A_{\text{in}}, c_0)$, where N, Σ denote the sets of nonterminal and terminal symbols, resp. $A_{\text{in}} \in N$ is a distinguished symbol, called the start symbol, $c_0 \in C_0$ is the initial configuration, and R is a set of production rules of one of the following two forms:

$$A \rightarrow \text{if } \pi \text{ then } \zeta_1 B f \zeta_2 \quad A \rightarrow \text{if } \pi \text{ then } w$$

where $A \in N, B \in N$ called the *distinguished symbol*, $\pi \in \text{BE}(P)$ and $\zeta_1, \zeta_2 \in (N \cup \Sigma)^*, f \in F, w \in \Sigma^*$.

A pair of symbols from $N \times C$ is called an *object*. A string $\sigma \in ((N \times C) \cup \Sigma)^*$ is called a *sentential form*. A sentential form σ is said to derive a sentential form τ , written $\sigma \Rightarrow \tau$, if either (1) or (2).

$$\begin{array}{ll} (1) \sigma = \alpha(A, c)\beta & (2) \sigma = \alpha(A, c)\beta \\ A \rightarrow \text{if } \pi \text{ then } \zeta_1 B f \zeta_2 \in R & A \rightarrow \text{if } \pi \text{ then } w \in R \\ m(\pi)(c) = \text{true} & m(\pi)(c) = \text{true} \\ m(f) \text{ is defined on } c & \tau = \alpha w \beta \\ \tau = \alpha \zeta'_1(B, c') \zeta'_2 \beta & \end{array}$$

where $A, B \in N, \alpha, \beta \in ((N \times C) \cup \Sigma)^*, w \in \Sigma^*, c \in C, c' = m(f)(c)$ and ζ'_1, ζ'_2 are obtained from ζ_1 and ζ_2 respectively by replacing every nonterminal D by (D, c_0) . The reflexive and transitive closure of \Rightarrow , denoted by \Rightarrow^* , is defined as usual. The language generated by G is defined as $L(G) = \{w \in \Sigma^* \mid (A_{\text{in}}, c_0) \Rightarrow^* w\}$. The class of languages generated by CFL- S -Gs is denoted by $\mathfrak{L}_{\text{CFL}(S)}$. Obviously, CFL- S_{pd} -Gs correspond to normal LIGs. With respect to the corresponding occurrences, the object (B, c') is called the *distinguished child* of (A, c) . Given a derivation let $s = (A_1, c_1) \dots (A_n, c_n)$ with $n \in \mathbb{N}$ be some sequence of objects such that each (A_{i+1}, c_{i+1}) is the distinguished child of (A_i, c_i) with $1 \leq i < n$. Then s is called the *spine* from (A_1, c_1) to (A_n, c_n) and (A_n, c_n) is called a *distinguished descendent* of (A_1, c_1) .

A language generated by an CFL- S -G can be characterized in terms of a controlled grammar and an S -automaton. This will become important when we investigate languages generated by grammars using more than one pushdown.

Definition 3. If $S = (C, C_0, C_F, P, F, m)$ is a storage type, then an *S -automaton* M is a tuple $(Q, \Sigma, \delta, q_0, c_0, Q_F)$, where Q is a finite set of states, Σ is the input alphabet, $q_0 \in Q$ the initial state, $c_0 \in C_0$ the initial configuration, $Q_F \subseteq Q$ the set of final states, and δ , the transition relation, a finite subset of $Q \times \Sigma_\epsilon \times \text{BE}(P) \times Q \times F$, where $\text{BE}(P)$ denotes the set of boolean expressions over P .¹

¹ The empty string is denoted by ϵ . For each set V the notation V_ϵ is used as an abbreviation for $V \cup \{\epsilon\}$ and $\#V$ denotes the number of elements of V .

The set $\text{ID}(M) = Q \times \Sigma^* \times C$ is called the set of instantaneous descriptions. For each $(q_1, xw, c_1), (q_2, w, c_2) \in \text{ID}(M)$ with $x \in \Sigma_\epsilon$ we write $(q_1, xw, c_1) \vdash_M (q_2, w, c_2)$ if there exists $(q_1, x, \pi, q_2, f) \in \delta$ such that $m(\pi)(c_1) = \text{true}$,² $m(f)$ is defined on c_1 and $m(f)(c_1) = c_2$. The transitive and reflexive closure \vdash_M^* of \vdash_M is defined as usual. The language accepted by M is defined $L(M) = \{w \mid (q_0, w, c_0) \vdash_M^* (q, \epsilon, c) \text{ for some } c \in C \text{ and } q \in Q_F\}$ if $Q_F \neq \emptyset$ and $L(M) = \{w \mid (q_0, w, c_0) \vdash_M (q_1, w', c_1) \vdash_M \dots (q_n, \epsilon, c_n) \text{ for some } c_n \in C_F, q_n \in Q, c_i \in C - C_F \text{ and } q_i \in Q \text{ with } 1 \leq i < n\}$ otherwise. In the first case we say that M accepts by final state. In the second case M accepts by final configuration. We denote by $\mathcal{L}_C(S)$ and $\mathcal{L}_Q(S)$ the classes of languages accepted by S -automata accepting by final configuration and S -automata accepting by final state, respectively. If $\mathcal{L}_Q(S) = \mathcal{L}_C(S)$ and $\#C_0 = \#C_1 = 1$, then we call S *well-formed*. In this case we can drop the subscripts and write $\mathcal{L}(S)$. If moreover $C_0 = C_F$, then we say that S is *concatenating*. This will turn out to be an important property if composition of storage types is considered. A typical example of an concatenating storage type is provided by S_{pd} .

Linear control of context free grammars (CFGs) is defined in [16]. The definition is twofold. The first part says which paths have to be controlled; the second part defines the control itself.

Definition 4. A *linear distinguished grammar* (LDG) is a quadruple $G = (N, \Sigma, R, A_{\text{in}})$, where N , Σ and A_{in} (the start symbol) are interpreted as in a normal CFG and where R is a finite set of production rules of the form: $A \rightarrow \beta_1 X! \beta_2$ with $A \in N$, $X \in N \cup \Sigma$, called the *distinguished symbol*, $\beta_1, \beta_2 \in (N \cup \Sigma)^*$, and $!$ a special symbol not in $(N \cup \Sigma)$. A *Linear Controlled Grammar* (LCG) is a pair $K = (G, H)$, where G is a LDG and H is a language over R , called the control language.

A string $\sigma \in ((N \cup \Sigma) \times R^*)^*$ is called a *sentential form*. A sentential form σ is said to derive a sentential form τ , written $\sigma \rightrightarrows \tau$, if

$$\begin{aligned} \sigma &= \gamma(A, \omega)\delta \\ r &= A \rightarrow \beta_1 X! \beta_2 \in R \\ \tau &= \gamma\beta'_1(X, \omega r)\beta'_2\delta \end{aligned}$$

where $A \in N$, $X \in N \cup \Sigma$, $\beta_1, \beta_2 \in (N \cup \Sigma)^*$, $\gamma, \delta \in ((N \cup \Sigma) \times R^*)^*$, $\omega \in R^*$, and β'_1 and β'_2 are obtained from β_1 and β_2 resp. by replacing every symbol $Y \in N \cup \Sigma$ by (Y, ϵ) . The reflexive and transitive closure of \rightrightarrows , denoted by \rightrightarrows^* , is defined as usual. The language generated by K is defined as $L(K) = \{a_1 a_2 \dots a_n \mid (S, \epsilon) \rightrightarrows^* (a_1, \omega_1) (a_2, \omega_2) \dots (a_n, \omega_n) \text{ and } a_i \in \Sigma, \omega_i \in H \text{ for } 1 \leq i \leq n\}$. Let \mathfrak{G}_{LD} be the class of all LDGs. For any class of grammars \mathfrak{G} for which control is defined let $\mathfrak{G}/\mathcal{L} = \{(G, H) \mid G \in \mathfrak{G} \text{ and } H \in \mathcal{L}\}$ and for any class of grammars \mathfrak{G} let $L(\mathfrak{G}) = \{L(K) \mid K \in \mathfrak{G}\}$. Regarding each element of $(N \cup \Sigma) \times R^*$ as an object, the definitions of spines and distinguished descendants carry over straightforwardly to LCGs.

² In fact only $m(\pi)$ for $\pi \in P$ has been defined so far. It is, however, straightforward to extend the domain of m to $\text{BE}(P)$.

We can think of a CFL- S -grammar as an LDG controlled by (the language accepted by) an S -automaton but in which the control word for each spine is computed on the fly. On the other hand, it is easy to imagine how a CFL- S -G can be split up into an LDG and an S -automaton generating the correct set of control words. The relation between S -grammars, S -automata and controlled grammars was established for linear grammars (i.e. grammars with only one nonterminal in the right-hand side of each rule) by [14]. The more general case of CFL- S -Gs and LDGs is captured in the following theorem:

Theorem 1. $L(\mathfrak{G}_{LD}/\mathfrak{L}_Q(S)) = \mathfrak{L}_{CFL(S)}$

Proof. “ \subseteq ”: Let $K = (G, L(M))$ be a LCG for some LDG $G = (N, \Sigma, R, A_{in})$ and some S -automaton $M = (Q, R, \delta, q_0, c_0, Q_F)$ with $S = (C, C_0, C_F, P, F, m)$ an arbitrary storage type. Construct a CFL- S -G $G' = (N', \Sigma, R', A'_{in}, c_0)$ as follows.

$$\begin{aligned} N' &= \{(X, q) \mid X \in N \cup \Sigma \text{ and } q \in Q\} \\ A'_{in} &= (A_{in}, q_0) \\ R' &= \{(A, q) \rightarrow \text{if } \pi \text{ then } \beta'_1(X, p)f\beta'_2 \mid \\ &\quad r = A \rightarrow \beta_1 X! \beta_2 \in R \text{ and } (q, r, \pi, p, f) \in \delta\} \\ &\cup \{(X, q) \rightarrow \text{if } \pi \text{ then } (X, p)f \mid \\ &\quad X \in N \cup \Sigma \text{ and } (q, \epsilon, \pi, p, f) \in \delta\} \\ &\cup \{(a, q) \rightarrow a \mid a \in \Sigma \text{ and } q \in Q_F\} \end{aligned}$$

where β'_1 and β'_2 are obtained from β_1 and β_2 respectively by replacing every symbol $Y \in N \cup \Sigma$ by (Y, q_0) .

It is easy to verify that the configurations of the storage of M during a computation correspond to the associated storage configurations of the nonterminals on one spine (in G), and that the word accepted by M is the the string of productions rules (from G') that are used to expand the nonterminals on the spine. Consequently, this sequence of rules has to be a word in $L(M)$. Now it can be shown by induction that

$$\begin{aligned} (A, \epsilon) \xRightarrow{*} v'(B, \omega)w' \text{ and } (q_1, \omega, c_1) \vdash_M^* (q_2, \epsilon, c_2) \\ \text{iff} \\ ((A, q_1), c_1) \xRightarrow{*}_{G'} v((B, q_2), c_2)w \end{aligned}$$

where (B, ω) and $((B, q_2), c_2)$ are distinguished children of (A, ϵ) and $((A, q_1), c_1)$ resp. and with $A, B \in N$, $q_1, q_2 \in Q$, $c_1, c_2 \in C$, $v, w \in \Sigma^*$, $\omega \in R^*$ and $v', w' \in \Sigma \times R^*$ the respective corresponding counterparts of v and w .

“ \supseteq ”: For some $S = (C, C_0, C_F, P, F, m)$ let $G = (N, \Sigma, R, A_{in}, c_0)$ be a CFL- S -G. Construct an LDG $G' = (N, \Sigma, R', A_{in})$, where $A \rightarrow \beta_1 B! \beta_2 \in R'$ if $A \text{ if } \pi \text{ then } \rightarrow \beta_1 Bf\beta_2 \in R$ and $A \rightarrow b!w \in R'$ if $A \text{ if } \pi \text{ then } \rightarrow bw \in R$ with $A, B \in N$, $f \in F$, $\beta_1, \beta_2 \in (N \cup \Sigma)^*$, $b \in \Sigma$, $f \in F$ and $w \in \Sigma^*$. Further construct an S -automaton $M = (N \cup \{q_F\}, R', \delta, A_{in}, c_0, \{q_F\})$, by setting

$$\begin{aligned} \delta &= \{(A, r, \pi, B, f) \mid r = (A \text{ if } \pi \text{ then } \rightarrow \beta_1 Bf\beta_2) \in R\} \\ &\cup \{(A, r, \pi, q_F, \text{id}) \mid r = (A \text{ if } \pi \text{ then } \rightarrow w) \in R\} \end{aligned}$$

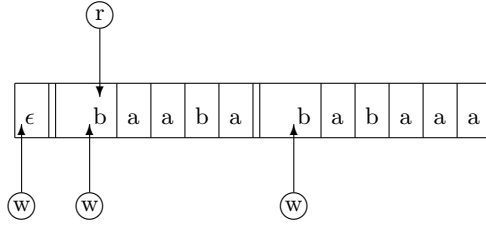


Fig. 1. Example of a possible configuration of a $(S_{\text{pd}} \circ_r S_{\text{pd}}) \circ_r S_{\text{pd}}$ storage with marking of possibilities for reading and writing.

where *id* denotes *identity*, the operation that does not change the configuration of the storage, assuming w.l.o.g. that the storage type S has an identity. It can be shown by induction that

$$\begin{aligned} (A, c_1) &\xRightarrow{\circ_r^*} v(B, c_2)w \\ &\text{iff} \\ (A, \epsilon) &\xRightarrow{\circ_r^*} v'(B, \omega)w' \text{ and } (A, \omega, c_1) \vdash_{\text{M}}^* (B, \epsilon, c_2) \end{aligned}$$

where (B, c_2) and (B, ω) are distinguished children of (A, c_1) and (A, ϵ) resp. and with $A, B \in N$, $q_1, q_2 \in Q$, $c_1, c_2 \in C$, $v, w \in \Sigma^*$, $\omega \in R^*$ and $v', w' \in \Sigma \times R^*$ the respective corresponding counterparts of v and w . \square

3 Sequences of Pushdowns

As noted above, grammars or automata using two pushdowns seem useful for the description of natural languages but are too powerful. Automata with a kind of storage that is in some respects similar to a sequence of pushdowns but less powerful are defined in [1]. These automata are called *multi-pushdown automata*. In fact the storage a multi-pushdown automaton uses can be considered as a tuple of pushdowns if writing is concerned, but with respect to reading the storage behaves like a single pushdown. A possible configuration of such a storage type is shown in Figure 1. In order to give a simple definition of this kind of storage types independently of the automata using them, I will define a concatenation operator on storage types. Applied to pushdowns it yields the storage types actually used in [1].

Definition 5. For any storage type $S^1 = (C^1, C_0^1, C_F^1, P^1, F^1, m^1)$ and $S^2 = (C^2, C_0^2, C_F^2, P^2, F^2, m^2)$ with $F^2 = F_r^2 \cup F_w^2$, where F_r^2 denotes the set of reading instructions and F_w^2 the set of writing instructions, the *concatenation w.r.t. reading* of S^1 and S^2 is defined as $S^1 \circ_r S^2 = (C, C_0, C_F, P, F, m)$ where

$$\begin{aligned} C &= C^1 \times C^2 & C_0 &= C_0^1 \times C_0^2 & P &= P^1 \cup \{\text{test}(p) \mid p \in P^2\} \\ C_F &= C_F^1 \times C_F^2 & F &= F^1 \cup \{\text{do}(f) \mid f \in F^2\} \end{aligned}$$

and for every $c^1 \in C^1$, $c^2 \in C^2$

$$\begin{aligned}
m(p)((c^1, c^2)) &= m^1(p)(c^1) \\
m(f)((c^1, c^2)) &= (m^1(f)(c^1), c^2) \\
m(\text{test}(p))((c^1, c^2)) &= m^2(p)(c^2) \\
m(\text{do}(f))((c^1, c^2)) &= (c^1, m^2(f)(c^2)) \text{ if } f \in F_w^2 \\
m(\text{do}(f))((c^1, c^2)) &= (c^1, m^2(f)(c^2)) \text{ if } f \in F_r^2 \text{ and } c^1 \in C_F^1
\end{aligned}$$

For the pushdown type let the instruction *push* be in F_w and all other instructions in F_r . Note that $m(\text{do}(f))((c^1, c^2))$ is undefined for $f \in F_r^2$ if $c^1 \notin C_F^1$. If S^1 is a pushdown this means that reading and popping from the second component is only possible if S^1 contains only the empty string. Thus a concatenation of pushdowns defined in this way corresponds to the storage type used in [1]. An important property of concatenated storage types that is easy to verify is that $S^1 \circ_r S^2$ is well-formed and non-concatenating if S^1 and S^2 are.

The languages that are accepted by an $S \circ_r S_{\text{pd}}$ -automaton can be characterized by linear control of an LDG restricted by a condition defined for LIGs in [6], namely that the distinguished symbol in a rule is always the leftmost nonterminal daughter.

Definition 6. Let $G = (N, \Sigma, R, A_{\text{in}})$ be an LDG. G is called an *extended left LDG* (ELLDG) if each production is of one of the following two forms: $A \rightarrow aB!C$ or $A \rightarrow a!$, where $A, B \in N$, $C \in N_\epsilon$ and $a \in \Sigma_\epsilon$. If each production is of the form $A \rightarrow BC!a$ or $A \rightarrow a!$, G is called an *extended right LDG* (ERLDG). The class of ELLDGs (ERLDGs) is denoted by $\mathfrak{G}_{\text{ELLD}}$ ($\mathfrak{G}_{\text{ERLD}}$).

The same restrictions can be applied to CFL- S -Gs. We say that a CFL- S -G $G = (N, \Sigma, R, A_{\text{in}}, c_0)$ is an *extended left linear S Grammar* (ELL- S -G) if each production is of one of the following two forms: $A \rightarrow aBfC$ or $A \rightarrow a$, where $A, B \in N$, $C \in N_\epsilon$, $a \in \Sigma_\epsilon$ and f an instruction symbol from S . If each production is of the form $A \rightarrow BCfa$ or $A \rightarrow a$, G is called an *extended right linear S Grammar* (ERL- S -G). By applying the construction from the proof of Theorem 1 to the grammars with the restricted rule format it is easy to verify that the classes of languages generated by ELL- S -Gs (ERL- S -Gs) and ELLDGs (ERLDGs) controlled by S -automata are equivalent.

Now I will show that the class of languages generated by ELLDGs controlled by languages accepted by an S -automaton is identical to the class of languages accepted by $(S \circ_r S_{\text{pd}})$ -automata. Since controlled ELLDGs and ELL- S -Gs are equivalent, we can use the latter just as well for the proof that will be a bit shorter in this case. The idea for the simulation of the controlled grammar is to use the second component of the storage of the automaton to store the grammar symbols simulating a leftmost derivation. The first component can correspond to the storage of the grammar. Each time a symbol of the control word, i.e. a rule of the grammar, is produced, this rule is carried out: the initial terminal is read from the input, the first nonterminal is coded in a new state and all other symbols are pushed to the second component of the store. As long as the expansion of distinguished symbols is controlled, reading from the second

component is not possible and not necessary since the controlled path leads from leftmost nonterminal to leftmost nonterminal daughter. Since each control word has to be computed on the first component of the storage, the simulation is only possible if the configuration of that component after recognizing a control word is again the initial configuration. That is to say that S has to be concatenating.

Lemma 1. *For each concatenating storage type S the following holds:*

$$\mathfrak{L}_{\text{ELLD}}/\mathfrak{L}(S) \subseteq \mathfrak{L}(S \circ_r S_{pd})$$

Proof. Let $G = (N, \Sigma, R, A_{\text{in}}, c_0)$ be an ELL- S -G for some concatenating storage $S = (C, C_0, C_F, P, F, m)$. Construct the required $S \circ_r S_{pd}$ -automaton $M = (N_\epsilon \times N_\epsilon, \Sigma, \delta, A_{\text{in}}, (c_0, \epsilon), \emptyset)$ by setting

$$\delta = \{((A, \epsilon), a, \pi, (B_1, B_2), f) \mid \quad (1)$$

$$A \rightarrow \text{if } \pi \text{ then } aB_1fB_2 \in R\}$$

$$\cup \{((A, B), a, \pi, (A, \epsilon), \text{do}(\text{push}(B))) \mid A, B \in N\} \quad (2)$$

$$\cup \{((A, \epsilon), a, \pi, (\epsilon, \epsilon), \text{id}) \mid \quad (3)$$

$$A \rightarrow \text{if } \pi \text{ then } a \in R\}$$

$$\cup \{((\epsilon, \epsilon), \epsilon, \text{test}(\text{top} = A), (A, \epsilon), \text{do}(\text{pop})) \mid A \in N\} \quad (4)$$

It can be shown by induction that

$$((A, \epsilon), x, (\sigma, \epsilon)) \vdash_M^* ((\epsilon, \epsilon), \epsilon, (c_\epsilon, \epsilon)) \text{ iff } (A, \sigma) \rightrightarrows_G^* x$$

In case $A = A_{\text{in}}$ and $\sigma = c_0$ we see that M accepts exactly the languages that are generated by G . \square

The idea behind the construction of an ELL- S -G simulating an $S \circ_r S_{pd}$ -automaton is again straightforward. The underlying CFG is used to code the configurations of the pushdown and the additional storage of type S makes the same transitions as the corresponding component of the automaton. The computation of a control word and hence the simulation of the first component of the $S \circ_r S_{pd}$ storage has to stop as soon as the foot of a spine in the ELL- S -G is reached. This, however, is unproblematic since this point corresponds to a situation in which the automaton will pop an element from the second component and the first component has to be in a final configuration, so to say accepting the control word. The subsequent configurations of the first component are simulated along a new spine which, according to the definition of LCGs, starts with c_0 . Thus, the final configuration that was reached by the first component has to be c_0 , which is the case if S is concatenating.

Lemma 2. *For each concatenating storage type S the following holds:*

$$\mathfrak{L}(S \circ_r S_{pd}) \subseteq \mathfrak{L}_{\text{ELLD}}/\mathfrak{L}(S)$$

Proof. Let $M = (Q, \Sigma, \delta, q_0, c_0, \emptyset)$ be an $S \circ_r S_{pd}$ -automaton for some concatenating storage type $S = (C, C_0, C_F, P, F, m)$. Assume w.l.o.g. that the only boolean expression applied to the pushdown is $\text{top} = \gamma$ and that this predicate is used only in combination with popping. Construct an ELL- S -G $G = ((Q \times \Gamma_\epsilon \times Q) \cup A_{\text{in}}, \Sigma, R, A_{\text{in}}, c_0)$ with Γ the stack alphabet of the pushdown component S_{pd} , A_{in} a new symbol and

$$R = \{A_{\text{in}} \rightarrow (q_0, \epsilon, q) \mid q \in Q\} \quad (1)$$

$$\cup \{(q_1, A, q_3) \rightarrow \text{if } \pi \text{ then } a(q_2, A, q_3)(f) \mid \quad (2)$$

$$(q_1, a, \pi, q_2, f) \in \delta,$$

$$\pi \in \text{BE}(P), f \in F \cup \{\text{do}(\text{id})\}, A \in \Gamma_\epsilon \text{ and } q_3 \in Q\}$$

$$\cup \{(q_1, A, q_4) \rightarrow \text{if } \pi \text{ then } a(q_2, B, q_3)(\text{id})(q_3, A, q_4) \mid \quad (3)$$

$$(q_1, a, \pi, q_2, \text{do}(\text{push}(B))) \in \delta,$$

$$A \in \Gamma_\epsilon \text{ and } q_3, q_4 \in Q\}$$

$$\cup \{(q_1, A, q_2) \rightarrow \text{if empty then } a \mid \quad (4)$$

$$(q_1, a, \text{test}(\text{top} = A), q_2, \text{do}(\text{pop})) \in \delta,$$

$$A \in \Gamma_\epsilon \text{ and } q_1 \in Q\}$$

To establish that $L(G) = L(M)$ it can be shown by induction on the number of steps in the derivation of G that

$$((p, A, q), \sigma) \rightrightarrows^* x \text{ iff } (p, x, (\sigma, A)) \vdash_M^* (q, \epsilon, (c_\epsilon, \epsilon)) \quad \square$$

Theorem 2. *For each concatenating storage type S the following holds.*

$$\mathfrak{L}(S \circ_r S_{pd}) = L(\mathfrak{G}_{\text{ELLD}}/\mathfrak{L}(S)) \quad \square$$

The language classes of the hierarchy established in [1] can in our notation be defined for each $i \in \mathbb{N}$ as $\mathfrak{L}_i = \mathfrak{L}(S_i)$ where $S_0 = S_{\text{triv}}$ and $S_i = S_{i-1} \circ_r S_{pd}$. According to Theorem 2 this hierarchy can be defined as well as $\mathfrak{L}_0 = \mathfrak{L}_{\text{reg}}$ and $\mathfrak{L}_i = L(\mathfrak{G}_{\text{ELLD}}/\mathfrak{L}_{i-1})$, where $\mathfrak{L}_{\text{reg}}$ denotes the class of regular languages³. Given this representation, it is possible to compare this hierarchy to Weir's ([16]) hierarchy of mildly context sensitive languages that is defined as $\mathfrak{L}_0 = \mathfrak{L}_{\text{reg}}$ and $\mathfrak{L}_i = L(\mathfrak{G}_{\text{LD}}/\mathfrak{L}_{i-1})$. Since each ELLDG is an LDG, we can conclude that each class of the hierarchy of concatenated pushdowns (cf. Table 1) is mildly context sensitive too. Finally, it follows immediately that the language generated by some LD- S -G, with S a concatenation of pushdowns, is mildly context sensitive. This is the result which in fact allows us to use these grammars for linguistic purposes.

The inclusion of the multiple-pushdown languages in the classes from Weir's hierarchy can be intuitively understood in another way, considering the storage types defined by [17]. Take the automata accepting languages of the second class in Weir's hierarchy as an example. These automata use (linear) pushdowns of pushdowns. The allowed operations include not only replacement of the topmost symbol of the topmost pushdown by a sequence of symbols, but also replacing the entire topmost pushdown by a sequence of pushdowns including the original one. This means in fact that we can either write on the top (symbols or pushdowns)

³ Note that Theorem 2 holds for S_{triv} though it is not well-formed.

or below the topmost pushdown (only pushdowns). Thus, in some sense these automata too incorporate the idea of having several possibilities for writing but only one for reading.

4 Linguistic Relevance

In the previous section I have sketched a possible way of defining grammars with more than one pushdown for storing information that is needed to compute non-local dependencies. Now we might ask how well this model fits empirical data and the linguistic theories that I have alleged as a motivation for the use of multiple-pushdowns.

In Rizzi's ([10]) theory of relativized minimality the distinction of different types of nonlocal dependencies plays an important role in the formulation of constraints like minimality. Minimality requires in principle that each trace is antecedent-governed by the closest (c-commanding) antecedent available. Rizzi distinguishes three types of antecedent-government: A-government (from argument positions), A'-government (from non-argument positions) and head-government. Minimality must only be respected among relations of the same type, hence *relativized* minimality. This means that no potential antecedent of a certain type can fall properly between a trace and its governor of the same type. On the one hand, non-local dependencies are in this way divided into three categories in the way we assumed. On the other hand, we do not need three pushdowns to account for these. In fact, the relativized minimality requirement entails that there can be at most three overlapping paths, and a finite storage would suffice for formalization in terms of *S*-grammars (cf. [11]). There is however a possibility to circumvent minimality. Some A'-traces do not have to be antecedent governed, but only have to be bound by their respective antecedents. For binding, however, no minimality condition is formulated.

Minimalist theories like those sketched in [2] capture (relativized) minimality effects with the *minimal link* or *shortest movement* condition. It is assumed that movement is always triggered by the need to check a feature. Basically, features can be divided into licensing features (licensors) and features that have to be licensed (licensees). The verb (or a functional projection of the verb) e.g. has features that license the case features on the arguments. If some licensee is not in a local relation to an appropriate licensor, the latter can attract the phrase bearing the feature that has to be licensed. A licensor can, however, only attract the closest appropriate licensee available. The different types of movement are distinguished by the features that are checked in the target position, as far as A- and A'-movement are concerned. Movement of heads is not further split up in this way. The minimalist program sketches several possibilities of relaxing the minimality condition. The first way comes about along with the creation of complex categories by means of Chomsky-adjunction. The typical example is successive cyclic head movement: a head moves to the next head and adjoins to it. Further movement of the complex head is regarded as one movement step though two new nonlocal dependencies are created. This process can

be carried on recursively. Thus it is possible to create an unbounded number of local dependencies. Since successive cyclic head movement always creates nested dependencies, a formalization using a pushdown seems to be fine. The second possibility indicated by Chomsky to create an unbounded number of nonlocal dependencies arises if we allow a head to have multiple specifiers and an attracting feature to escape erasure after checking. After raising a phrase to the specifier of the attracting head, the next more deeply embedded phrase that can check the same feature becomes the closest candidate for moving. Due to the extension principle it has to be placed in the next higher specifier. Consequently the dependencies created in this way are nested. Since this process can in principle take place independently with regard to different features, our approach using a number of pushdowns fits very well. Finally, phrases can, under certain conditions, become *equidistant*. If two phrases are equidistant either can be attracted. Now movement can be crossing as well. Chomsky however argues that at most two phrases should be allowed to become equidistant. Thus, this is not a possibility to create an unbounded number of nonlocal dependencies.

In some psycholinguistic theories about sentence processing it is argued that the human parser distinguishes between different types of relations and that the cognitive difficulty of a sentence is not determined by the number of relations that has to be computed but rather by the number of relations of the same type ([13,5]). This leads Lewis to propose a model for sentence processing that in fact uses different stacks for different types of relations. For the human parser stacks might have an upper bound on the number of elements they can contain, typically two or three. As soon as more elements of one type have to be remembered the sentence will become too difficult to parse, but not ungrammatical. Thus Rado [8] explicitly argues that we need two pushdowns to compute nonlocal dependencies in Hungarian. One pushdown is used for topicalized phrases, the other for *wh*-phrases.⁴

Finally, we have to ask what price must be paid for the restriction on reading from the pushdowns, that we had to impose on composite storage types in order to domesticate them. Assume that indices are pushed onto the storage when a moved element is encountered, while they are popped in the respective base positions. Suppose further that one pushdown is used for each type of movement. Now the restriction under consideration requires that all indices of one type must be read before the base positions of another type are found, such that the base positions have to be ordered while landing sites might be mixed. In other words we have to be able to locate a region from which all arguments originate, a region in which all *wh*-phrases are generated etc. In natural languages, on the contrary, we find a clustering of the landing sites for each type. This can e.g. be observed in languages with multiple *wh*-movement. The strict adjacency between the fronted *wh*-phrases in these languages even leads [12] to analyze the *wh*-phrases as a single constituent formed by adjunction. Similarly we find in Dutch constructions in which an arbitrary number of verbs can raise a strict

⁴ To account for crossing dependencies that are possible to some degree she assumes that the topmost two elements of the *wh*-stack are visible.

adjacency between these verbs. The clustering of the landing sides chimes in with the minimalist program as well. As I have argued above, the only two possibilities for moving an unbounded number of phrases are constituted by successive cyclic movement and by movement into multiple specifiers of one head. In both cases the landing sites clearly have to be adjacent.

In the following I will show that the restriction on reading in a concatenation of pushdowns can be replaced by a condition on writing. I will show that we obtain similar results with regard to the generative capacity in this case.

5 A Linguistically Inspired Variant

In the previous section I have argued that it is more appropriate to leave popping that is needed in the base positions unrestrained and restrict the writing operations that have to be carried through while the displaced constituents are encountered, rather than the other way around. This results in the following alternative definition for the concatenation of storage types.

Definition 7. For any storage type $S^1 = (C^1, C_0^1, C_F^1, P^1, F^1, m^1)$ and $S^2 = (C^2, C_0^2, C_F^2, P^2, F^2, m^2)$ with $F^2 = F_r^2 \cup F_w^2$, where F_r^2 and F_w^2 denote the sets of reading and writing instructions resp., the *concatenation w.r.t. writing* of S^1 and S^2 is defined as $S^1 \circ_w S^2 = (C, C_0, C_F, P, F, m)$ where C, C_0, C_F, P and F are defined as in the concatenation w.r.t. reading and m is defined for every $c^1 \in C^1, c^2 \in C^2$ as

$$\begin{aligned} m(p)((c^1, c^2)) &= m^1(p)(c^1) \\ m(f)((c^1, c^2)) &= (m^1(f)(c^1), c^2) \\ m(\text{test}(p))((c^1, c^2)) &= m^2(p)(c^2) \\ m(\text{do}(f))((c^1, c^2)) &= (c^1, m^2(f)(c^2)) \text{ if } f \in F_w^2 \text{ and } c^1 \in C_F^1 \\ m(\text{do}(f))((c^1, c^2)) &= (c^1, m^2(f)(c^2)) \text{ if } f \in F_r^2 \end{aligned}$$

Concatenation w.r.t. writing of a storage type S and a pushdown involves control of extended *right* LDGs by the reversal of $\mathfrak{L}(S)$.⁵ This is actually the reason why we need controlled grammars in addition to S -grammars. Control by the reversal of a language cannot be expressed in terms of S -grammars, unless the control language is closed under reversal. This will, however, not be the case for the languages I will consider below.

The idea for the simulation of a linear controlled ERLDG by a $(S \circ_w S_{\text{pd}})$ -automaton is to use the second component both for simulating a derivation and for storing the rules that are used. When a nonterminal is expanded the label and the right hand side of the rule are pushed onto the second component. As soon as the automaton starts computing the control word with the first

⁵ For a string $w = a_1 a_2 \dots a_{n-1} a_n$ the reversal w^R of w is defined as $a_n a_{n-1} \dots a_2 a_1$. For a language L , $L^R = \{w \mid w^R \in L\}$ and for a class of languages \mathfrak{L} , $\mathfrak{L}^R = \{L \mid L^R \in \mathfrak{L}\}$.

component, the second component can only be used for reading. The terminals stored there can be read from the input string and the rules now serve as input for the computation on the first storage. Thus it is clear that the word of rules recognized is the reversal of the sequence of applied rules. Nonterminals on the second component cannot be expanded until the first component is in a final configuration again, and, consequently, it has accepted the control word. Thus, a path can only be controlled if it keeps leading from a rightmost nonterminal to its mother.

Lemma 3. *Let $S = (C, C_0, P, F, m)$ be a concatenating storage type. Then the following holds.*

$$L(\mathfrak{G}_{\text{ERLD}}/\mathfrak{L}(S)^R) \subseteq \mathfrak{L}(S \circ_w S_{pd})$$

Proof. Let $S = (C, C_0, P, F, m)$ be a concatenating storage type and let $K = (G, L(M)^R)$ be an LCG, with $G = (N, \Sigma, R, A_{\text{in}})$ an ERLDG and $M = (Q, R, \delta, q_0, c_0, \emptyset)$ an S -automaton. Construct an $S \circ_w S_{pd}$ -automaton $M' = (Q \times (N \cup \Sigma \cup R \cup \{\epsilon\}), \Sigma, \delta', (q_0, A_{\text{in}}), (c_0, \epsilon), \emptyset)$ such that $L(M') = L(K)$ by setting

$$\delta' = \{((q, A), \epsilon, \text{true}, (q_0, \epsilon), \text{do}(\text{push}(\beta r))) \mid r = A \rightarrow \beta \in R\} \quad (1)$$

$$\cup \{(((q, a), a, \text{true}, (q, \epsilon), \text{id}) \mid a \in \Sigma\} \quad (2)$$

$$\cup \{(((q_1, r), \epsilon, \pi, (q_2, \epsilon), f) \mid (q_1, r, \pi, q_2, f) \in \delta\} \quad (3a)$$

$$\cup \{(((q_1, r), \epsilon, \pi, (q_2, r), f) \mid (q_1, \epsilon, \pi, q_2, f) \in \delta\} \quad (3b)$$

$$\cup \{(((q, \epsilon), \epsilon, \text{test}(\text{top} = X), (q, X), \text{do}(\text{pop})) \mid X \in N \cup \Sigma \cup R, q \in Q\} \quad (4)$$

It can be shown by induction that

$$\begin{aligned} & ((q_0, A), a_1 \dots a_j, (c_0, \epsilon)) \vdash_{M'}^* ((q_0, B), \epsilon, (c_0, b_k r_k b_{k-1} r_{k-1} \dots b_1 r_1)) \\ & \quad \text{and} \\ & ((q_1, \epsilon), b_k b_{k-1} \dots b_1, (c_1, b_k r_k b_{k-1} r_{k-1} \dots b_1 r_1)) \vdash_{M'}^* ((q_2, \epsilon), \epsilon, (c_2, \epsilon)) \\ & \quad \text{iff} \\ & (A, \epsilon) \xrightarrow{\mathfrak{G}}^* (a_1, \omega_1) \dots (a_j, \omega_j) (B, r_1 r_2 \dots r_k) (b_k, \epsilon) \dots (b_1, \epsilon) \\ & \quad \text{and} \\ & (q_1, r_k r_{k-1} \dots r_1, c_1) \vdash_M^* (q_2, \epsilon, c_2) \\ & \quad \text{and} \\ & \omega_i \in L(M)^R \text{ for } 1 \leq i \leq j \end{aligned}$$

with $A \in N$, $B \in N \cup \Sigma$, $\omega_i \in R^*$, $r_i \in R$ and $b_i \in \Sigma \cup R$ (with $1 \leq i \leq k$).

If $r_k r_{k-1} \dots r_1 \in L(M)$ and $B \in \Sigma$, such that $a_1 \dots a_j B b_k \dots b_1 \in L(K)$ we have $q_1 = q_0$, $c_1 = c_0$ and c_2 is a final configuration. Consequently, the two parts of the computation in M' that exists according to the assertion can be connected using a transition of type (2) and we see that $a_1 \dots a_j B b_k \dots b_1 \in L(M')$ as well. If we have a computation in M' , it is easy to see that it can be split up in a way such that the assertion applies. Now the assertion guarantees that the accepted language is in $L(K)$ as well.

Assuming that the assertion is true for some $n \in \mathbb{N}$, for the first direction of the proof the interesting case is given by the following derivation

$$\begin{aligned} & (A, \epsilon) \xRightarrow{\mathcal{C}} \\ & (D, \epsilon)(B, r)(b, \epsilon) \xRightarrow{\mathcal{C}}^n \\ & (a_1, \omega_1) \dots (a_j, \omega_j)(B, r)(b, \epsilon) \end{aligned}$$

Assuming that $\omega_i \in L(M)^R$ for $1 \leq i \leq j$ and $(q_1, r, c_1) \vdash_M (q_2, \epsilon, c_2)$ we find for M' :

$$\begin{aligned} & ((q_0, A), a_1 \dots a_j, (c_0, \epsilon)) \vdash_{M'} && \text{(by definition)} \\ & ((q_0, \epsilon), a_1 \dots a_j, (c_0, DBbr)) \vdash_{M'} && \text{(by definition)} \\ & ((q_0, D), a_1 \dots a_j, (c_0, Bbr)) \vdash_{M'}^* && \text{(by induction hypothesis)} \\ & ((q, \epsilon), \epsilon, (c_0, Bbr)) \vdash_{M'} && \text{(by definition)} \\ & ((q_0, B), \epsilon, (c_0, br)) \end{aligned}$$

and

$$((q_1, \epsilon), b, (c_1, br)) \vdash_{M'}^* ((q_2, \epsilon), \epsilon, (c_2, \epsilon)) \text{ (by definition)}$$

The interesting case for the other direction is a computation starting with a transition introduced by (1):

$$\begin{aligned} & ((q_0, A), a_1 \dots a_j, (c_0, \epsilon)) \vdash_{M'} \\ & ((q_0, \epsilon), a_1 \dots a_j, (c_0, DBbr)) \vdash_{M'} \\ & ((q_0, D), a_1 \dots a_j, (c_0, Bbr)) \vdash_{M'}^* \\ & ((q_0, \epsilon), \epsilon, (c_0, Bbr)) \vdash_{M'} \\ & ((q, B), \epsilon, (c_0, br)) \end{aligned}$$

For some $q_1, q_2 \in Q$ and $c_1, c_2 \in C$ we can moreover assume that

$$((q_1, \epsilon), b_0, (c_1, br)) \vdash_{M'}^* ((q_2, \epsilon), \epsilon, (c_2, \epsilon))$$

From this we can conclude

$$\begin{aligned} & (A, \epsilon) \xRightarrow{\mathcal{C}} \\ & (D, \epsilon)(B, r)(b, \epsilon) \xRightarrow{\mathcal{C}}^* \text{ (by definition)} \\ & (a_1, \omega_1) \dots (a_j, \omega_j)(B, r)(b, \epsilon) \text{ (by induction)} \end{aligned}$$

Furthermore we can infer from the induction hypothesis that $\omega_i \in L(M)^R$ for $1 \leq i \leq j$ and from the way M' was constructed $(q_1, r, c_1) \vdash_M^* (q_2, \epsilon, c_2)$ \square

Given a $(S \circ_w S_{\text{pd}})$ -automaton, the idea for the construction of an LCG is that the nonterminals of the controlled grammar broadly code the symbols of the second component and the states at which they were pushed and popped. Thus rightmost paths code, from a terminal towards the top, successive states at which popping and working on the first component takes place and, consequently, these paths have to be controlled. In the rules of the controlled grammar, terminal symbols that are read while the automaton works on the first storage component are added at the end of the right-hand side, behind the distinguished symbol, yielding the ERLDG rule format. Finally, we have to account for complete computations (especially computations beginning and ending with an initial state)

that are carried through between pushing two symbols to the second component. These are simulated by rewriting of triples coding the states corresponding to that computation and the last terminal symbol that was accepted before.

Lemma 4. *Let $S = (C, C_0, C_F, P, F, m)$ be some concatenating storage type. Then the following holds.*

$$\mathfrak{L}(S \circ_w S_{pd}) \subseteq L(\mathfrak{G}_{\text{ERLD}}/\mathfrak{L}(S)^R)$$

Proof. Let $S = (C, C_0, C_F, P, F, m)$ be a concatenating storage type and $M = (Q, \Sigma, \delta, q_0, (c_0, \epsilon), \emptyset)$ an $(S \circ_w S_{pd})$ -automaton. Assume without loss of generality that the only boolean expression applied to the pushdown is $\text{top} = \gamma$ and that this predicate is used only in combination with popping. Construct an ERLDG $G = (N, \Sigma, R, A_{\text{in}})$ where A_{in} is a new symbol and $N \subseteq \{A_{\text{in}}\} \cup (Q \times \Gamma' \times Q \times \Gamma') \cup (Q \times \Sigma \times Q)$ with Γ the stack alphabet of the pushdown and $\Gamma' = \Gamma \cup \{\#\}$, where $\# \notin \Gamma$. Construct further an S -automaton $M' = (Q \cup \{q'_0\}, R, \delta', q'_0, c_0, \emptyset)$. R and δ' are determined by the following⁶:

For each $q \in Q$ (1)

$$r = A_{\text{in}} \rightarrow (q_0, \#, q, \#) \in R$$

$$(q, r, \text{true}, q, \text{id}) \in \delta'$$

If $(q_1, a, \pi, q_2, f) \in \delta$ with $f \in F \cup \{\text{do}(\text{id})\}$ then (2)

for each $q_x \in Q, A \in \Gamma, B \in \{A, \epsilon\}$ and $b \in \Sigma$

$$r_1 = (q_x, A, q_2, B) \rightarrow (q_x, A, q_1, B)a \in R$$

$$r_2 = (q_1, A, q_2, A) \rightarrow a \in R$$

$$r_3 = (q_x, b, q_2) \rightarrow (q_x, b, q_1)a \in R$$

$$r_4 = (q_1, b, q_2) \rightarrow a \in R$$

$$(q_1, \{r_1, r_2, r_3, r_4\}, \pi, q_2, f) \in \delta'$$

$$(q'_0, \epsilon, \text{true}, q_1, \text{id}) \in \delta'$$

If $(q_1, a, \text{true}, q_2, \text{do}(\text{push}(B))) \in \delta$ then (3)

for each $q_x, q_y, q_z \in Q$ and $A \in \Gamma$

$$r_1 = (q_x, A, q_z, A) \rightarrow (q_x, a, q_2)(q_2, B, q_y, \epsilon)(q_y, A, q_z, A) \in R$$

$$r_2 = (q_x, A, q_z, A) \rightarrow (q_x, a, q_2)(q_2, B, q_z, \epsilon) \in R$$

$$r_3 = (q_x, a, q_2) \rightarrow (q_x, a, q_1)a \in R$$

$$r_4 = (q_1, a, q_2) \rightarrow a \in R$$

$$(q_z, \{r_1, r_2\}, \text{true}, q_z, \text{id}) \in \delta'$$

$$(q_1, \{r_3, r_4\}, \text{true}, q_2, \text{id}) \in \delta'$$

$$(q'_0, \epsilon, \text{true}, q_1, \text{id}) \in \delta'$$

If $(q_1, a, \text{top} = A, q_2, \text{do}(\text{pop})) \in \delta$ then (4)

for each $q_x \in Q$ and $A \in \Gamma$

$$r_1 = (q_x, A, q_2, \epsilon) \rightarrow (q_x, A, q_1, A)a \in R$$

$$r_2 = (q_1, A, q_1, \epsilon) \rightarrow a \in R$$

$$(q_1, \{r_1, r_2\}, \text{true}, q_2, \text{id}) \in \delta'$$

$$(q'_0, \epsilon, \text{true}, q_1, \text{id}) \in \delta'$$

⁶ For transitions I will use the notation $(p, \{a_1, \dots, a_n\}, \pi, q, f)$ as an abbreviation for $(p, a_1, \pi, q, f), \dots, (p, a_n, \pi, q, f)$

Now it can be shown that $L(M) = L(G)$.

“ $L(M) \subseteq L(G)$ ”. It can be shown by induction on n that

$$(q_1, a_1 \dots a_j b_k \dots b_1, (c_1, A)) \vdash_M^n (q_3, \epsilon, (c_2, B))$$

with $B \in \{A, \epsilon\}$ implies for some $q_2 \in Q$

$$\begin{aligned} ((q_1, A, q_3, B), \epsilon) &\Rightarrow^* (a_1, \omega_1) \dots (a_j, \omega_j) (b_k, r_1 r_2 \dots r_k) (b_{k+1}, \epsilon) \dots (b_1, \epsilon) \\ &\text{and} \\ (q'_0, r_k \dots r_1, c_1) &\vdash_{M'}^* (q_3, \epsilon, c_2) \\ &\text{and} \\ \omega_i^R &\in L(M) \text{ for } 1 \leq i \leq j \end{aligned}$$

We start the induction with $n = 1$. In this case, if $A = B$ the transition has involved some $f \in F \cup \{\text{do}(\text{id})\}$. If otherwise $B = \epsilon$ then $f = \text{do}(\text{pop})$. In both cases we have $j = 0$ and $k = 1$ since the a 's are generated only after pushing some symbol onto the second storage component. The grammar G' has the following rule, introduced by (2) if $B = A$ and by (4) if $B = \epsilon$.

$$r_1 = (q_1, A, q_3, B) \rightarrow b_1$$

For M' we find

$$(q'_0, r_1, c_1) \vdash_{M'} (q_1, r_1, c_1) \vdash_{M'} (q_2, \epsilon, c_2)$$

Suppose the implication is true for some $n \in \mathbb{N}$. Consider a computation of length $n + 1$. Suppose the first transition employs some $f \in F \cup \{\text{do}(\text{id})\}$. The computation has the following shape:

$$\begin{aligned} (q_1, da_1 \dots a_j b_k \dots b_1, (c_1, A)) &\vdash_M \\ (q_2, a_1 \dots a_j b_k \dots b_1, (c_2, A)) &\vdash_M^n \\ (q_4, \epsilon, (c_3, B)) &\end{aligned}$$

We distinguish two cases: the number of elements that was pushed onto the second component during the computation is zero (I.) or not zero (II.).

By the induction hypothesis we know there is a derivation in G corresponding to the last n transitions. In case I. the last rule used must be of the type r_2 introduced by (2) or (4), and all other applied rules have to be left linear; consequently the first terminal that is generated is a distinguished child of $((q_1, A, q_3, B), \epsilon)$. By definition this is $(b_k, r_1 \dots r_k)$. If the last rule was introduced by (2), the derivation is of the following form. If the last rule was introduced by (4) the situation is similar.

$$\begin{aligned} ((q_2, A, q_4, B), \epsilon) &\Rightarrow^{n-1} \\ ((q_2, A, q_3, A), r_1 r_2 \dots r_{k-1}) &(b_{k-1}, \epsilon) \dots (b_1, \epsilon) \Rightarrow \\ (b_k, r_1 r_2 \dots r_k) &(b_{k-1}, \epsilon) \dots (b_1, \epsilon) \end{aligned}$$

and for M' we find $(q'_0, r_k \dots r_1, c_2) \vdash_{M'} (q_3, r_k \dots r_1, c_2) \vdash_{M'}^* (q_4, \epsilon, c_3)$. For all the left linear rules rewriting a nonterminal with q_2 as its first component, there are

corresponding rules that have q_1 as the first component of their nonterminals and that are identical to the rules used in the above derivation up to this point. Thus, the grammar also licenses the derivation which we are looking for.

$$\begin{aligned} & ((q_1, A, q_4, B), \epsilon) \Rightarrow^{n-1} \\ & ((q_1, A, q_3, A), r'_1 r'_2 \dots r'_{k-1})(b_{k-1}, \epsilon) \dots (b_1, \epsilon) \Rightarrow \\ & ((q_1, A, q_2, A), r'_1 r'_2 \dots r'_k)(b_k, \epsilon) \dots (b_1, \epsilon) \Rightarrow \\ & (d, r'_1 r'_2 \dots r'_k r'_{k+1})(b_k, \epsilon) \dots (b_1, \epsilon) \end{aligned}$$

It is easy to verify that M' can make the following computation

$$\begin{aligned} (q'_0, r'_{k+1} r'_k \dots r'_1, c_1) & \vdash_{M'} (q_2, r'_{k+1} r'_k \dots r'_1, c_1) \vdash_{M'} \\ (q_3, r'_k \dots r'_1, c_2) & \vdash_{M'}^* (q_4, \epsilon, c_3) \end{aligned}$$

In case II. we again know that there is a derivation in G corresponding to the last n transitions. Now some element has been pushed onto the second component of the storage. Thus, the leftmost derivation corresponding to the last n steps has the following shape.

$$\begin{aligned} & ((q_2, A, q_4, B), \epsilon) \Rightarrow^* \\ & ((q_2, x, q_3), \omega_1) \beta \Rightarrow (a_1, \omega_1 r) \beta \Rightarrow^* \\ & (a_1, \omega_1 r)(a_2, \omega_2) \dots (a_j, \omega_j)(b_k, \omega_{j+1})(b_{k-1}, \epsilon) \dots (b_1, \epsilon) \end{aligned}$$

where β is a sentential form in G . By the same argument as in the previous case we also have the following derivation:

$$\begin{aligned} & ((q_1, A, q_4, B), \epsilon) \Rightarrow^* ((q_1, x, q_3), \omega_0) \beta \Rightarrow \\ & ((q_1, x, q_2), \omega_0 r_1)(a_1, \epsilon) \beta \Rightarrow (a_0, \omega_0 r_1 r_2)(a_1, \epsilon) \beta \Rightarrow^* \\ & (a_0, \omega_0 r_1 r_2)(a_1, \epsilon)(a_2, \omega_2) \dots (a_j, \omega_j)(b_k, \omega_{j+1})(b_{k-1}, \epsilon) \dots (b_1, \epsilon) \end{aligned}$$

Furthermore, it is easy to verify that $r_2 r_1 \omega_0^R$ is recognized by M' .

The other cases in which the first transition involves an operation on the second component are rather easy compared to the previous case.

“ $L(G) \subseteq L(M)$ ”. Somewhat more generally, the following assertion can be shown to hold by induction on the length n of the derivation. From this it can be concluded straightforwardly, as we have seen above, that for each derivation yielding only terminal symbols there is a computation in M accepting the corresponding terminal string.

$$\begin{aligned} & ((q_1, A, q_4, B), \epsilon) \Rightarrow^n \\ & (a_1, \omega_1) \dots (a_j, \omega_j)((q_2, C, q_3, C), r_1 r_2 \dots r_k)(b_k, \epsilon) \dots (b_1, \epsilon) \\ & \text{and} \\ & (q_3, r_k \dots r_1, c_1) \vdash_{M'}^* (q_4, \epsilon, c_2) \\ & \text{and} \\ & \omega_i^R \in L(M) \text{ for } 1 \leq i \leq j \end{aligned}$$

implies

$$\begin{aligned} & (q_1, a_1 \dots a_j, (c_0, A)) \vdash_M^* (q_2, \epsilon, (c_0, C\alpha)) \\ & \text{and} \\ & (q_3, b_k \dots b_1, (c_1, C\alpha)) \vdash_M^* (q_4, \epsilon, (c_2, B)) \end{aligned}$$

In case $n = 0$ we have $q_1 = q_2$, $q_3 = q_4$, $A = B = C$ and $k = 0$ and the assertion is trivially true.

The interesting step for the induction is a derivation starting with a production introduced by (3) of the construction. The derivation will now be of the following form:

$$\begin{aligned}
 & ((q_1, A, q_7, A), \epsilon) \Rightarrow & (1) \\
 & ((q_1, a_k, q_3), \epsilon)((q_3, C, q_4, \epsilon), \epsilon)((q_4, A, q_7, A), r_0) \Rightarrow^* & (2) \\
 & (a_0, \omega_0) \dots (a_k, \epsilon)((q_3, C, q_4, \epsilon), \epsilon)((q_4, A, q_7, A), r_0) \Rightarrow^* & (3) \\
 & (a_0, \omega_0) \dots (a_k, \epsilon)(a_{k+1}, \omega_{k+1}) \dots (a_l, \omega_l)((q_4, A, q_7, A), r_0) \Rightarrow^* & (4) \\
 & (a_0, \omega_0) \dots (a_l, \omega_l) \dots (a_m, \omega_m)((q_5, D, q_6, D), r_0 r_1 \dots r_n)(b_n, \epsilon) \dots (b_1, \epsilon)
 \end{aligned}$$

By induction and by the construction we can conclude,

$$\begin{array}{ll}
 (q_1, a_0 \dots a_m, (c_0, A)) \vdash_M^* & \text{from (2)} \\
 (q_2, a_k \dots a_m, (c_0, A)) \vdash_M & \text{by the construction from (1)} \\
 (q_3, a_{k+1} \dots a_m, (c_0, C A)) \vdash_M^* & \text{by induction from (3)} \\
 (q_4, a_l \dots a_m, (c_0, A)) \vdash_M^* & \text{by induction from (4)} \\
 (q_5, \epsilon, (c_0, D \zeta A)) &
 \end{array}$$

And

$$\begin{array}{ll}
 (q_6, b_n \dots b_1, (c_0, D \zeta A)) \vdash_M^* & \text{by induction from (3)} \\
 (q_7, \epsilon, (c_0, A)) &
 \end{array}$$

Note that the first induction step does not follow from the induction hypothesis as formulated here. But is easy to verify this assertion in order to complete the proof. \square

Theorem 3. *For each concatenating storage type S the following holds.*

$$\mathfrak{L}(S \circ_w S_{\text{pd}}) = L(\mathfrak{G}_{\text{ERLD}}/\mathfrak{L}(S)^R) \quad \square$$

According to this theorem, automata using sequences of pushdowns formed by concatenation w.r.t. writing generate languages of the hierarchy that is defined as follows: $\mathfrak{L}_0 = \mathfrak{L}_{\text{reg}}$ and $\mathfrak{L}_i = L(\mathfrak{G}_{\text{ERLD}}/\mathfrak{L}_{i-1}^R)$. Because $L(\mathfrak{G}_{\text{ERLD}}/\mathfrak{L})^R = L(\mathfrak{G}_{\text{ELLD}}/\mathfrak{L}) \subseteq L(\mathfrak{G}_{\text{LD}}/\mathfrak{L})$, the control language is each time a mildly context sensitive language and consequently the language generated by the controlled grammar is as well. Finally, we can conclude that CFL- S -Gs with S a sequences of pushdowns formed by concatenation w.r.t. writing generate only mildly context sensitive languages.

6 Conclusion

Picking up from current linguistic theories the idea that a grammar should differentiate between various types of dependencies, I have defined two possibilities to extend LIGs. Both ways formalize the idea of using a number of index pushdowns with a restricted accessibility. This induced two hierarchies of languages, which are presented schematically in Table 1. The upper part displays the hierarchy

Table 1. Overview of Concatenated Storages

$S_0 = S_{\text{triv}}$ $S_i = S_{i-1} \circ_r S_{\text{pd}}$	$\mathfrak{C}^{\mathfrak{R}}_0 = \mathfrak{L}_{\text{reg}}$ $\mathfrak{C}^{\mathfrak{R}}_i = L(\mathfrak{G}_{\text{ELLD}}/\mathfrak{C}^{\mathfrak{R}}_{i-1})$
$S_0 = S_{\text{triv}}$ $S_1 = S_{\text{triv}} \circ_r S_{\text{pd}} = S_{\text{pd}}$ $S_2 = S_{\text{pd}} \circ_r S_{\text{pd}}$ $S_3 = (S_{\text{pd}} \circ_r S_{\text{pd}}) \circ_r S_{\text{pd}}$ \dots	$\mathfrak{C}^{\mathfrak{R}}_0 = \mathfrak{L}_{\text{reg}}$ $\mathfrak{C}^{\mathfrak{R}}_1 = L(\mathfrak{G}_{\text{ELLD}}/\mathfrak{L}_{\text{reg}}) = \mathfrak{L}_{\text{CF}}$ $\mathfrak{C}^{\mathfrak{R}}_2 = L(\mathfrak{G}_{\text{ELLD}}/\mathfrak{L}_{\text{CF}}) = \mathfrak{L}_{\text{ELLI}}$ \dots
$S_0 = S_{\text{triv}}$ $S_i = S_{i-1} \circ_w S_{\text{pd}}$	$\mathfrak{C}^{\mathfrak{W}}_0 = \mathfrak{L}_{\text{reg}}$ $\mathfrak{C}^{\mathfrak{W}}_i = L(\mathfrak{G}_{\text{ERLD}}/(\mathfrak{C}^{\mathfrak{W}}_{i-1})^R)$
$S_0 = S_{\text{triv}}$ $S_1 = S_{\text{triv}} \circ_w S_{\text{pd}} = S_{\text{pd}}$ $S_2 = S_{\text{pd}} \circ_w S_{\text{pd}}$ $S_3 = (S_{\text{pd}} \circ_w S_{\text{pd}}) \circ_w S_{\text{pd}}$ \dots	$\mathfrak{C}^{\mathfrak{W}}_0 = \mathfrak{L}_{\text{reg}}$ $\mathfrak{C}^{\mathfrak{W}}_1 = L(\mathfrak{G}_{\text{ERLD}}/(\mathfrak{L}_{\text{reg}})^R)$ $\quad = L(\mathfrak{G}_{\text{ERLD}}/\mathfrak{L}_{\text{reg}}) = \mathfrak{L}_{\text{CF}}$ $\mathfrak{C}^{\mathfrak{W}}_2 = L(\mathfrak{G}_{\text{ERLD}}/(\mathfrak{L}_{\text{CF}})^R)$ $\quad = L(\mathfrak{G}_{\text{ERLD}}/\mathfrak{L}_{\text{CF}}) = \mathfrak{L}_{\text{ERLI}}$ $\mathfrak{C}^{\mathfrak{W}}_3 = L(\mathfrak{G}_{\text{ERLD}}/(\mathfrak{L}_{\text{ERLI}})^R)$ $\quad = L(\mathfrak{G}_{\text{ERLD}}/\mathfrak{L}_{\text{ELLI}})$ \dots

that was established by [1], using a restriction on reading. On the left hand side we have the new definition of the storage types that were used in the original definition. On the right hand side we have the representation using controlled languages according to Theorem 2. The lower part of the table presents the new hierarchy. Interestingly, the second member of this hierarchy is a restricted class *extended right linear indexed languages*, denoted by $\mathfrak{L}_{\text{ERLI}}$, that we have defined in [7]. There we argue that the candidates for inheriting the stack of indices can be restricted while all possibilities that unrestricted LIGs provide for the description of natural languages are maintained. The classes of both hierarchies are subclasses of the classes of a hierarchy of mildly context sensitive languages defined by [16]. Thus, their generative capacity seems appropriate for natural languages. I have, however, argued that the languages and the storage types of the latter hierarchy are more adequate for the description of natural languages on empirical as well as on language theoretical grounds.

Acknowledgment. This paper has been worked out in the *Innovationskolleg ‘Formale Modelle kognitiver Komplexität’ (INK 12)* funded by the DFG. I would like to thank Peter Staudacher, Jens Michaelis, Hans–Martin Gärtner and Lothar Budach for many helpful discussions.

References

1. Breveglieri, L., A. Cherubini, C. Citrini, and S. Crespi Reghizzi. Multi-push-down languages and grammars. *International Journal of Foundations of Computer Science*, 7(3):253–291, 1996.
2. Chomsky, N. *The Minimalist Program*. MIT Press, Cambridge, MA, 1995.
3. Ferguson, K. S., and E. M. Groat. Defining “shortest move”, deriving A-movement, strict cyclicity and incorporation phenomena. In *GLOW Newsletter 32*, 1994.
4. Gazdar, G. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. D. Reidel, Dordrecht, 1988.
5. Lewis, R. L. Interference in short-term memory: The magical number two (or three) in sentence processing. *Journal of Psycholinguistic Research*, 25(1):93–115, 1996.
6. Michaelis, J., and C. Wartena. How linguistic constraints on movement conspire to yield languages analyzable with a restricted form of LIGs. In *Proceedings of the Conference on Formal Grammar (FG '97)*, pages 158–168, Aix en Provence, 1997.
7. Michaelis, J., and C. Wartena. LIGs with reduced derivation sets. In Bouma, G., G.-J. M. Kruijff, E. Hinrichs, and R. T. Oehrle, editors, *Constraints and Resources in Natural Language Syntax and Semantics*, volume II of *Studies in Constrained Based Lexicalism*, pages 263–279. CSLI Publications, Stanford, CA, 1999.
8. Rado, J. *Topic-focus vs. background: The role of structural information in discourse interpretation*. PhD thesis, University of Massachusetts, Amherst, MA, 1997.
9. Rambow, O. *Formal and Computational Aspects of Natural Language Syntax*. PhD thesis, University of Pennsylvania, 1994.
10. Rizzi, L. *Relativized Minimality*. MIT Press, Cambridge, MA, 1990.
11. Rogers, J. On descriptive complexity, language complexity, and GB. In Blackburn, P., and M. de Rijke, editors, *Specifying Syntactic Structures*, chapter 6. Cambridge University Press, 1997.
12. Rudin, C. On multiple question and multiple wh-fronting. *Natural Language and Linguistic Theory*, 6:445–501, 1988.
13. Stabler, E. P. The finite connectivity of linguistic structure. In Clifton, C., L. Frazier, and K. Rayner, editors, *Perspectives on Sentence Processing*, chapter 13, pages 303–336. Lawrence Erlbaum, Hillsdale, NJ, 1994.
14. Vogler, H. Iterated linear control and iterated one-turn pushdowns. *Mathematical Systems Theory*, 19(2):117–133, 1986.
15. Weir, D. J. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania, 1988.
16. Weir, D. J. A geometric hierarchy beyond context-free languages. *Theoretical Computer Science*, 104(4):235–261, 1992.
17. Weir, D. J. Linear iterated pushdowns. *Computational Intelligence*, 10(4):422–430, 1994.

Author Index

Finger, Marcelo	11	Meyer-Viol, Wilfried	159
Francez, Nissim	31	Michaelis, Jens	179
		Monz, Christof	1
Hegner, Stephen J.	48	Niehren, Joachim	106, 199
Jäger, Gerhard	70	Ogata, Norihiro	219
Joshi, Aravind K.	90	Pogodalla, Sylvain	230
Koller, Alexander	106, 199	de Rijke, Maarten	1
Kracht, Marcus	126	Tiede, Hans-Joerg	251
Kulick, Seth	90	Treinen, Ralf	106
Kurtonina, Natasha	90	Wartena, Christian	266
Lecomte, Alain	143		