

Lecture Notes in Artificial Intelligence 3492

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

FoLLI Publications on Logic, Language and Information

Editors-in-Chief

Luigia Carlucci Aiello, *University of Rome "La Sapienza", Italy*

Michael Moortgat, *University of Utrecht, The Netherlands*

Maarten de Rijke, *University of Amsterdam, The Netherlands*

Editorial Board

Carlos Areces, *INRIA Lorraine, France*

Nicholas Asher, *University of Texas at Austin, TX, USA*

Johan van Benthem, *University of Amsterdam, The Netherlands*

Raffaella Bernardi, *Free University of Bozen-Bolzano, Italy*

Antal van den Bosch, *Tilburg University, The Netherlands*

Paul Buitelaar, *DFKI, Saarbrücken, Germany*

Diego Calvanese, *Free University of Bozen-Bolzano, Italy*

Ann Copestake, *University of Cambridge, United Kingdom*

Robert Dale, *Macquarie University, Sydney, Australia*

Luis Fariñas, *IRIT, Toulouse, France*

Claire Gardent, *INRIA Lorraine, France*

Rajeev Goré, *Australian National University, Canberra, Australia*

Reiner Hähnle, *Chalmers University of Technology, Göteborg, Sweden*

Wilfrid Hodges, *Queen Mary, University of London, United Kingdom*

Carsten Lutz, *Dresden University of Technology, Germany*

Christopher Manning, *Stanford University, CA, USA*

Valeria de Paiva, *Palo Alto Research Center, CA, USA*

Martha Palmer, *University of Pennsylvania, PA, USA*

Alberto Policriti, *University of Udine, Italy*

James Rogers, *Earlham College, Richmond, IN, USA*

Francesca Rossi, *University of Padua, Italy*

Yde Venema, *University of Amsterdam, The Netherlands*

Bonnie Webber, *University of Edinburgh, Scotland, United Kingdom*

Ian H. Witten, *University of Waikato, New Zealand*

Philippe Blache Edward Stabler
Joan Busquets Richard Moot (Eds.)

Logical Aspects of Computational Linguistics

5th International Conference, LACL 2005
Bordeaux, France, April 28-30, 2005
Proceedings

Series Editors

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

Volume Editors

Philippe Blache
Université de Provence
CNRS, Laboratoire Parole et Langage
29, Avenue Robert Schuman, 13621 Aix-en-Provence, France
E-mail: pb@lpl.univ-aix.fr

Edward Stabler
UCLA Department of Linguistics
3125 Campbell Hall, Box 951543, Los Angeles, CA 90095-1542, USA
E-mail: stabler@ucla.edu

Joan Busquets
Université Bordeaux 3
CNRS, ERSS
Domaine Universitaire, 33607 Pessac Cedex, France
E-mail: busquets@u-bordeaux3.fr

Richard Moot
Université Bordeaux 1
CNRS, LaBRI
351, Cours de la Libération, 33405 Talence Cedex, France
E-mail: Richard.Moot@labri.fr

Library of Congress Control Number: 2005924437

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

ISSN	0302-9743
ISBN-10	3-540-25783-7 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-25783-7 Springer Berlin Heidelberg New York

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

Springer is a part of Springer Science+Business Media
springeronline.com

© Springer-Verlag Berlin Heidelberg 2005
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11422532 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the 5th International Conference on Logical Aspects of Computational Linguistics held April 28–30, 2005 in Bordeaux, France. This proceedings contains papers on a wide range of logical and formal methods in computational linguistics, with studies of particular grammar formalisms (Categorial Grammars, TAG, Dependency Grammars or Minimalist Grammars) and their computational properties (complexity, determinism, unification), language engineering (grammar development, parsing, translation) and traditional questions about the syntax/semantics interface. Formal aspects are moreover assessed with actual linguistic data from different languages (English, French, Arabic), which is the sign of a maturing field.

This text, as well as the conference itself, is then the occasion to bring together people coming from different horizons: logicians, linguists, computational scientists. This fits perfectly well with the mission of FoLLI, the Association of Logic, Language and Information, and so this textbook inaugurates the new FoLLI/LNAI series.

The Program Committee faced a difficult task because we received many submissions (40% rejected), and the reviewing task had to be done quickly. We thank all our reviewers and especially those who had to be recruited at the last minute. We also thank very much the Organizing Committee. Bordeaux, known as a wine capital, is now becoming a research center in this field.

We would like to thank all the people who made this 5th LACL possible: the Program Committee, the external reviewers, the Organizing Committee, and the LACL sponsors. Last, but not least, very special thanks to Christian Retoré who has been, since the very beginning, the heart of LACL. This conference, and these books, simply would not exist without him.

April 2005

Philippe Blache and Edward Stabler

Organization

LACL 2005 was organized by INRIA, Université Bordeaux 1, and Université Bordeaux 3.

Program Committee

Conference Chairs	Philippe Blache (Université de Provence, France) Edward Stabler (University of California, USA) Joan Busquets (ERSS and INRIA) Richard Moot (LABRI and INRIA)
-------------------	--

Organizing Chairs	Joan Busquets (ERSS and INRIA) Richard Moot (LABRI and INRIA)
-------------------	--

Referees

D. Aguilar-Solis	K. Gerdes	G. Morrill
J.-M. Andreoli	F. Hamm	D. Oehrle
P. Blache	G. Holst	J.-P. Prost
J. Bos	G. Jaeger	C. Retoré
J. Busquets	S. Kepser	F. Richter
H. Christiansen	G. Kobele	J. Rogers
V. Dahl	M. Kracht	V. Shanker
P. de Groote	A. Lecomte	E. Stabler
M. de Rijke	U. Moennich	M. Steedman
D. Duchier	M. Moortgat	
M. Dymetman	R. Moot	

Organizing Committee

Maxime Amblard	(Université Bordeaux 1 and LaBRI-CNRS and INRIA)
Philippe Biais	(CNRS LaBRI)
Karine Cabandé	(CNRS DR Aquitaine et Poitou-Charentes)
Catherine Girard	(INRIA-Futurs)
Patrick Henry	(CNRS LABRI and INRIA-Futurs)
Brigitte Larue-Bourdon	(INRIA-Futurs)
Jean-Louis Lassartesses	(Université Bordeaux 1)
Annie Nadeau	(CNRS DR Aquitaine et Poitou-Charentes)
Christian Retoré	(Université Bordeaux 1 and LaBRI-CNRS and INRIA)

Sponsoring Institutions

CNRS

INRIA

Université Bordeaux 1

Université Bordeaux 3

Pôle Universitaire de Bordeaux

Conseil Régional d'Aquitaine

France Télécom

ERSS

CoLogNet

Table of Contents

LACL

k-Valued Non-associative Lambek Grammars (Without Product) Form a Strict Hierarchy of Languages <i>Denis Béchet, Annie Foret</i>	1
Dependency Structure Grammars <i>Denis Béchet, Alexander Dikovsky, Annie Foret</i>	18
Towards a Computational Treatment of Binding Theory <i>Roberto Bonato</i>	35
Translating Formal Software Specifications to Natural Language. A Grammar-Based Approach <i>David A. Burke, Kristofer Johannisson</i>	51
On the Selective Lambek Calculus <i>Marcelo da S. Corrêa, E. Hermann Haeusler</i>	67
Grammatical Development with XMG <i>Benoît Crabbé</i>	84
Lambek-Calculus with General Elimination Rules and Continuation Semantics <i>Nissim Francez</i>	101
A Note on the Complexity of Constraint Interaction: Locality Conditions and Minimalist Grammars <i>Hans-Martin Gärtner, Jens Michaelis</i>	114
Large Scale Semantic Construction for Tree Adjoining Grammars <i>Claire Gardent, Yannick Parmentier</i>	131
A Compositional Approach Towards Semantic Representation and Construction of ARABIC <i>Bassam Haddad, Mustafa Yaseen</i>	147
Strict Deterministic Aspects of Minimalist Grammars <i>John T. Hale, Edward P. Stabler</i>	162

A Polynomial Time Extension of Parallel Multiple Context-Free Grammar <i>Peter Ljunglöf</i>	177
Learnable Classes of General Combinatory Grammars <i>Erwan Moreau</i>	189
On Expressing Vague Quantification and Scalar Implicatures in the Logic of Partial Information <i>Areski Nait Abdallah, Alain Lecomte</i>	205
Describing Lambda Terms in Context Unification <i>Joachim Niehren, Mateu Villaret</i>	221
Category Theoretical Semantics for Pregroup Grammars <i>Anne Preller</i>	238
Feature Constraint Logic and Error Detection in ICALL Systems <i>Veit Reuer, Kai-Uwe Kühnberger</i>	255
Linguistic Facts as Predicates over Ranges of the Sentence <i>Benoît Sagot</i>	271
How to Build Argumental Graphs Using TAG Shared Forest: A View from Control Verbs Problematic <i>Djamé Seddah, Bertrand Gaiffe</i>	287
When Categorical Grammars Meet Regular Grammatical Inference <i>Isabelle Tellier</i>	301
The Expressive Power of Restricted Fragments of English <i>Allan Third</i>	317
The Complexity and Generative Capacity of Lexicalized Abstract Categorical Grammars <i>Ryo Yoshinaka, Makoto Kanazawa</i>	330
More Algebras for Determiners <i>Richard Zuber</i>	347
Author Index	363

k-Valued Non-associative Lambek Grammars (Without Product) Form a Strict Hierarchy of Languages

Denis Béchet¹ and Annie Foret²

¹ LINA – FRE 2729, Université de Nantes & CNRS

2, rue de la Houssinière — BP 92208, 44322 Nantes Cedex 03, France

Denis.Bechet@univ-nantes.fr

² IRISA – Université de Rennes 1, Campus Universitaire de Beaulieu,

Avenue du Général Leclerc, 35042 Rennes Cedex, France

Annie.Foret@irisa.fr

Abstract. The notion of k -valued categorial grammars where a word is associated to at most k types is often used in the field of lexicalized grammars as a fruitful constraint for obtaining several properties like the existence of learning algorithms. This principle is relevant only when the classes of k -valued grammars correspond to a real hierarchy of languages. This paper establishes the relevance of this notion for two related grammatical systems. In the first part, the classes of k -valued non-associative Lambek (NL) grammars without product is proved to define a strict hierarchy of languages. The second part introduces the notion of generalized functor argument for non-associative Lambek (NL_\emptyset) calculus without product but allowing empty antecedent and establishes also that the classes of k -valued NL_\emptyset grammars without product form a strict hierarchy of languages.

1 Introduction

The field of natural language processing includes lexicalized grammars such as classical categorial grammars [1], the different variants of Lambek calculus [2], lexicalized tree adjoining grammars [3], etc. In these lexicalized formalisms, a k -valued grammar associates at most k categories to each word of the lexicon. For a particular model of lexicalized grammars and their corresponding languages, this definition forms a (strict) hierarchy of classes of grammars when k increases. This hierarchy of grammars corresponds to a growing list of classes of languages that does not necessarily form a strict hierarchy.

In fact, in the field of lexicalized grammars, the concept of k -valued grammars is often used to define sub-classes of grammars and languages that satisfy some property when the whole class does not satisfy it. In particular, this notion is important for a lot of learnability results in Gold's model [4]. Usually, to find a positive result, some kind of restriction is necessary because very often the whole

class of grammars corresponds to (at least) context free languages and this class is known to be unlearnable no matter which grammatical system is used¹.

Usually, when the learnability can be established for a particular k -valued class of grammars, this property can also be proved for any $k \in \mathbb{N}$. When this assumption is true, for each $k \in \mathbb{N}$, there exists a learning algorithm that learns the grammars of this class from positive examples even if the whole class is not learnable (there does not exist a universal learning algorithm for all $k \in \mathbb{N}$): we hope that all the interesting grammars (the grammars for natural languages, for instance) are in a particular k -valued class.

In this context, we can wonder if, for a particular system of lexicalized grammars and languages, the class of classes of languages forms a strict hierarchy. If this assumption is not verified for a system, it usually means that for a certain k the class of k -valued languages is the same as the whole class: the hierarchy is truncated. The other possibility which is very rare consists in a hierarchy that has infinite steps, some steps corresponding to several contiguous integers. In this context, the proof that the hierarchy is not strict is usually a bad news for the concept of k -valued grammars corresponding to a system. For instance, the classes of k -valued classical categorial grammar form a strict hierarchy of languages [5] and for each $k \in \mathbb{N}$, the class of k -valued classical categorial grammar is learnable from positive examples.

In the paper, we are interested to prove that non-associative Lambek grammars without product (written NL) and non-associative Lambek grammars without product but with empty antecedent (written NL_\emptyset) form two strict hierarchies of classes of languages. The results give a direct justification of the notion of k -valued grammars for both systems. The paper also recalls a useful alternative presentation of NL using generalized AB deductions (written GAB) and generalized functor-argument structures (written FA) that were used in [6] for defining a learning algorithm but are a central notion in the paper for proving the strict hierarchy. For NL_\emptyset , the paper introduces similar notions: generalized AB deductions with empty applications (written GAB_\emptyset) and generalized functor-argument structures with empty arguments (written FA_\emptyset). This presentation is also proved, in the paper, to be equivalent to NL_\emptyset and the hierarchy theorem is given.

The paper is organized as follows. Section 2 gives some background knowledge on non-associative Lambek categorial grammars (without product). Section 3 focuses on (recently defined) structures, with alternative deduction rules for NL -grammars (without product) as generalized FA -structures; in fact these rules are extensions of the cancellation rules of classical categorial grammars that lead to the generalization of FA -structures used here. Section 4 presents the proof that the class of k -valued non-associative Lambek categorial grammars without product form a strict hierarchy. Section 5 considers the NL_\emptyset variant of NL ; we

¹ The class of context free language has a limit point: $\exists L, (L_i)_{i \in \mathbb{N}}$ such that $L = \bigcup_{i \in \mathbb{N}} L_i$ and $\forall i \in \mathbb{N} L_i \subsetneq L_{i+1}$. This property entails that any class of grammars that corresponds to (a superclass of) context free languages (of strings) is not learnable from positive examples.

define generalized functor-argument structures for this variant and a strict hierarchy theorem; we also revisit the case of AB (classical categorial grammars). Section 6 concludes.

2 Background

2.1 Categorial Grammars

The reader not familiar with Lambek Calculus and its non-associative version will find nice presentation in the first articles written by Lambek [2, 7] or more recently in [8, 9, 10, 11, 12, 13]. We use in the paper two variants of Lambek calculus: non-associative Lambek calculus without product with (NL_\emptyset) and without (NL) empty antecedent.

Definition 1 (Types). *The types Tp , or formulas, are generated from a set of primitive types Pr , or atomic formulas, by two binary connectives² “/” (over) and “\” (under):*

$$Tp ::= Pr \mid Tp \backslash Tp \mid Tp / Tp$$

Definition 2 (Rigid and k -valued categorial grammars). *A categorial grammar is a structure $G = (\Sigma, I, S)$ where:*

- Σ is a finite alphabet (the words in the sentences);
- $I : \Sigma \mapsto \mathcal{P}^f(Tp)$ is a function (called a lexicon) that maps a finite set of types to each element of Σ (the possible categories of each word);
- $S \in Pr$ is the main type associated to correct sentences.

If $X \in I(a)$, we say that G associates X to a and we write $G : a \mapsto X$.

A k -valued categorial grammar is a categorial grammar where, for every word $a \in \Sigma$, $I(a)$ has at most k elements. A rigid categorial grammar is a 1-valued categorial grammar.

2.2 Non-associative Lambek Calculi NL and NL_\emptyset

NL/NL_\emptyset Derivation $\vdash_{NL}/\vdash_{NL_\emptyset}$. As a logical system, we use Gentzen-style sequent presentation. A sequent $\Gamma \vdash A$ is composed of a binary tree of formulas Γ (the set of such trees is noted \mathcal{T}_{Tp}) which is the antecedent configuration and a succedent formula A . A context $\Gamma[\cdot]$ is a binary tree of formulas with a hole. For X , a formula or a binary tree of formulas, $\Gamma[X]$ is the binary tree obtained from $\Gamma[\cdot]$ by filling the hole with X . Γ can be empty in NL_\emptyset : Γ belongs to $\mathcal{T}_{Tp} \cup \{\emptyset\}$. In fact, to completely define the rules of NL_\emptyset , we use two equivalence relations on binary trees of formulas and the empty set³:

$$\Gamma[(\emptyset, \Delta)] \equiv_{NL_\emptyset} \Gamma[\Delta] \equiv_{NL_\emptyset} \Gamma[(\Delta, \emptyset)]$$

² No product connective is used in the paper.

³ One can define NL_\emptyset without the two equivalence relations by specializing the rules when one of the antecedents is empty but this gives a much longer definition.

Definition 3 (NL/NL_\emptyset). A sequent is valid in NL/NL_\emptyset and is noted $\Gamma \vdash_{NL} A/\Gamma \vdash_{NL_\emptyset} A$ iff $\Gamma \vdash A$ can be deduced from the following rules:

$$\begin{array}{c} \frac{}{A \vdash A} \mathbf{Ax} \qquad \frac{(\Gamma, B) \vdash A}{\Gamma \vdash A/B} /R \qquad \frac{(A, \Gamma) \vdash B}{\Gamma \vdash A \setminus B} \setminus R \\[10pt] \frac{\Gamma \vdash A \quad \Delta[A] \vdash B}{\Delta[\Gamma] \vdash B} \mathbf{Cut} \quad \frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(B/A, \Gamma)] \vdash C} /L \quad \frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(\Gamma, A \setminus B)] \vdash C} \setminus L \end{array}$$

Cut Elimination. We recall that the cut rule can be eliminated in \vdash_{NL} and in \vdash_{NL_\emptyset} : every derivable sequent has a cut-free derivation.

NL/NL_\emptyset Languages. \mathcal{E}^+ denotes the set of non-empty strings over \mathcal{E} . $\mathcal{T}_\mathcal{E}$ is the set of (non-empty) well-bracketed lists (binary trees) of elements of \mathcal{E} .

Definition 4 (Yield). If T is a tree where the leaves are elements of a set \mathcal{E} , $yield_\mathcal{E}(T) \in \mathcal{E}^+$ is the list of leaves of T .

This notation will be used for well-bracketed lists of words $yield_\Sigma$, for binary trees of formulas $yield_{Tp}$ and will be extended to FA structures (see further Definition 7).

Definition 5 (Language). Let $G = (\Sigma, I, S)$ be a categorial grammar.

- G generates a well-bracketed list of words $T \in \mathcal{T}_\Sigma$ (in NL/NL_\emptyset model) iff there exists Γ a binary tree of types, $c_1, \dots, c_n \in \Sigma$ and $A_1, \dots, A_n \in Tp$ such that:

$$\begin{cases} G : c_i \vdash \neg A_i \ (1 \leq i \leq n) \\ \Gamma = T[c_1 \rightarrow A_1, \dots, c_n \rightarrow A_n] \\ \Gamma \vdash_{NL} S / \Gamma \vdash_{NL_\emptyset} S \end{cases}$$

where $T[c_1 \rightarrow A_1, \dots, c_n \rightarrow A_n]$ means the binary tree obtained from T by substituting the left to right occurrences of c_1, \dots, c_n by A_1, \dots, A_n .

- G generates a string $c_1 \cdots c_n \in \Sigma^+$ iff there exists $T \in \mathcal{T}_\Sigma$ such that $yield_\Sigma(T) = c_1 \cdots c_n$ and G generates T .
- The language of well-bracketed lists of words of G , written $\mathcal{BL}_{NL}(G)/\mathcal{BL}_{NL_\emptyset}(G)$, is the set of well-bracketed lists of words generated by G .
- The language of strings corresponding to G , written $\mathcal{L}_{NL}(G)/\mathcal{L}_{NL_\emptyset}(G)$, is the set of strings generated by G .

One interest of NL when compared to classical categorial grammars lies in its possibility to easily encode a restriction on the use of a basic category. For instance when we want to distinguish between a noun phrase and pronouns in subject position or object position, we can proceed as follows.

Example 1. Let $\Sigma_1 = \{John, Mary, likes, he, she, him, her\}$ and let $Pr_1 = \{S, N, X_1, X_2\}$. We define the following rigid grammar:

$$G_1 = \left\{ \begin{array}{l} \text{John, Mary} \vdash \neg N \ ; \ \text{he, she} \vdash \neg N_1; \\ \text{him, her} \vdash \neg N_2 \ ; \ \text{likes} \vdash \neg N_1 \setminus (S/N_2). \end{array} \right.$$

where $N_1 = X_1/(N \setminus X_1)$ and $N_2 = X_2/(N \setminus X_2)$.

We get: $((\text{He likes}) \text{ Mary}) \in \mathcal{BC}_{NL}(G_1)$ but: $\text{John likes she} \notin \mathcal{L}_{NL}(G_1)$.

With NL_\emptyset , we can introduce a restrictive form of optional arguments; “little”, in the following example, is optionally associated to proper nouns. This is not possible directly with NL .

Example 2. Let $\Sigma_2 = \{\text{John, Mary, likes, little}\}$ and let $Pr_2 = \{S, N, L\}$.

We define: $G_2 = \left\{ \begin{array}{l} \text{John} \vdash \neg(L \setminus L) \setminus N \ ; \ \text{Mary} \vdash \neg(L \setminus L) \setminus N \\ \text{little} \vdash \neg L \setminus L \ ; \ \text{likes} \vdash \neg N \setminus (S/N) \end{array} \right.$

G_2 is a rigid (or 1-valued) grammar. We can prove that $((L \setminus L) \setminus N, N \setminus (S/N)), (L \setminus L) \setminus N \vdash_{NL_\emptyset} S$ and $((L \setminus L) \setminus N, N \setminus (S/N)), (L \setminus L, (L \setminus L) \setminus N) \vdash_{NL_\emptyset} S$. Thus, we get:

$$\begin{aligned} \text{John likes Mary} &\in \mathcal{L}_{NL_\emptyset}(G_2) \ ; \ \text{John likes little Mary} \in \mathcal{L}_{NL_\emptyset}(G_2) \\ ((\text{John likes}) \text{ Mary}) &\in \mathcal{BC}_{NL_\emptyset}(G_2) \ ; \ ((\text{John likes}) (\text{little Mary})) \in \mathcal{BC}_{NL_\emptyset}(G_2) \end{aligned}$$

2.3 Some Useful Models

Powerset Residuated Groupoids and Semi-groups. Let (M, \cdot) be a groupoid. Let $\mathcal{P}(M)$ denote the powerset of M . A *powerset residuated groupoid* over (M, \cdot) is the structure $(\mathcal{P}(M), \circ, \Rightarrow, \Leftarrow, \subseteq)$ such that for $X, Y \subseteq M$:

$$\begin{aligned} X \circ Y &= \{x.y : x \in X, y \in Y\} \\ X \Rightarrow Y &= \{y \in M : (\forall x \in X) x.y \in Y\} \\ Y \Leftarrow X &= \{y \in M : (\forall x \in X) y.x \in Y\} \end{aligned}$$

If (M, \cdot) has a unit I (that is : $\forall x \in M : I.x = x.I = x$), then the above structure is a *powerset residuated groupoid with unit* (it has $\{I\}$ as unit).

Interpretation. Given a powerset residuated groupoid $(\mathcal{P}(M), \circ, \Rightarrow, \Leftarrow, \subseteq)$, an *interpretation* is a map from primitive types p to elements $\llbracket p \rrbracket$ in $\mathcal{P}(M)$ that is extended to types and sequences in the natural way :

$$\llbracket C_1 \setminus C_2 \rrbracket = \llbracket C_1 \rrbracket \Rightarrow \llbracket C_2 \rrbracket \ ; \ \llbracket C_1 / C_2 \rrbracket = \llbracket C_1 \rrbracket \Leftarrow \llbracket C_2 \rrbracket \ ; \ \llbracket (C_1, C_2) \rrbracket = (\llbracket C_1 \rrbracket \circ \llbracket C_2 \rrbracket)$$

By a model property for NL : If $\Gamma \vdash_{NL} C$ then $\llbracket \Gamma \rrbracket \subseteq \llbracket C \rrbracket$

If (M, \cdot) is a groupoid with a unit I , we add $\llbracket \emptyset \rrbracket = \{I\}$ for the empty sequence \emptyset and get a model property for NL_\emptyset : if $\Gamma \vdash_{NL_\emptyset} C$ then $\llbracket \Gamma \rrbracket \subseteq \llbracket C \rrbracket$.

3 GAB Deductions and Generalized FA-Structures

In this section we focus on (recently defined) structures [6], with alternative deduction rules for NL -grammars (without product) as generalized FA-structures; in fact these rules are extensions of the cancellation rules of classical categorial grammars that lead to the generalization of FA-structures used here.

3.1 FA Structures over a Set \mathcal{E}

We give a general definition of FA structures over a set \mathcal{E} , whereas in practice \mathcal{E} is either an alphabet Σ or a set of types such as Tp .

Definition 6 (FA structures). *Let \mathcal{E} be a set, a FA structure over \mathcal{E} is a binary tree where each leaf is labelled by an element of \mathcal{E} and each internal node is labelled by $FApp$ (forward application) or $BApp$ (backward application):*

$$\mathcal{FA}_{\mathcal{E}} ::= \mathcal{E} \mid FApp(\mathcal{FA}_{\mathcal{E}}, \mathcal{FA}_{\mathcal{E}}) \mid BApp(\mathcal{FA}_{\mathcal{E}}, \mathcal{FA}_{\mathcal{E}})$$

Definition 7 (Tree yield). *The well-bracketed list of words obtained from a FA structure F over \mathcal{E} by forgetting $FApp$ and $BApp$ labels is called the tree yield of F over \mathcal{E} (notation $tree_{\mathcal{E}}(F)$).*

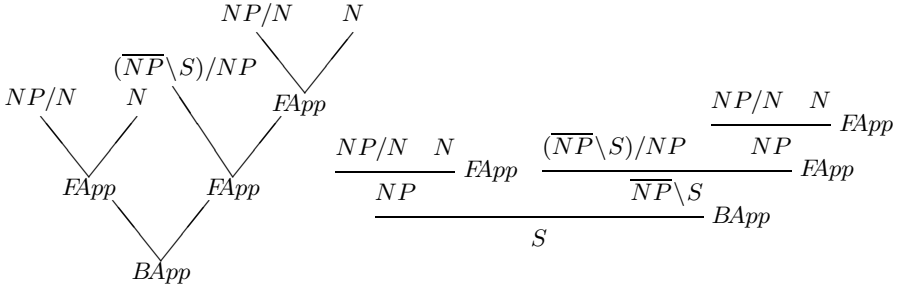
3.2 GAB Deductions

Definition 8 (GAB Deduction). *Generalized AB deductions (GAB deductions) over Tp are the deductions built from formulas on Tp (the base case) using the following conditional rules ($C \vdash_{NL} B$ must be valid in NL):*

$\frac{A/B \quad C}{A} FApp \quad \frac{C \quad B \setminus A}{A} BApp$ $C \vdash_{NL} B \text{ valid in NL}$

GAB deductions can be seen as a generalization of AB deductions in the following sense: for AB application rules C and B must be the same formula.

Definition 9 (FA structure of a GAB deduction). *To each GAB deduction \mathcal{P} , we associate a FA structure, written $FA_{Tp}(\mathcal{P})$, such that each internal node corresponds to the application of a rule in \mathcal{P} and is labelled by the name of this rule and where the leaves are the same as in \mathcal{P} .*



Here, $\overline{NP} = X/(NP \setminus X)$ and thus $NP \vdash_{NL} \overline{NP}$

Definition 10 (GAB Deductions of $F \vdash_{GAB} A$ or $\Gamma \vdash_{GAB} A$).

- For a FA structure $F \in \mathcal{FA}_{Tp}$ (over types) and $A \in Tp$, we say that \mathcal{P} is a GAB deduction⁴ of $F \vdash_{GAB} A$ when A is the type of the conclusion of \mathcal{P} and when $FA_{Tp}(\mathcal{P}) = F$.
- For a tree $\Gamma \in \mathcal{T}_{Tp}$ (over types) and $A \in Tp$, we say that \mathcal{P} is a GAB deduction of $\Gamma \vdash_{GAB} A$ when A is the type of the conclusion of \mathcal{P} and when $tree_{Tp}(FA_{Tp}(\mathcal{P})) = \Gamma$.

GAB Languages. Similarly to classical categorial grammars, we can associate to each categorial grammar a language of FA structures.

Definition 11 (GAB Languages). Let $G = (\Sigma, I, S)$ be a categorial grammar over Tp :

- $G = (\Sigma, I, S)$ generates a FA structure $F \in \mathcal{FA}_{\Sigma}$ (in the GAB derivation model) iff there exists a GAB derivation of a FA structure $D \in \mathcal{FA}_{Tp}$, $c_1, \dots, c_n \in \Sigma$ and $A_1, \dots, A_n \in Tp$ such that:

$$\begin{cases} G : c_i \vdash -A_i \ (1 \leq i \leq n) \\ D = F[c_1 \rightarrow A_1, \dots, c_n \rightarrow A_n] \\ D \vdash_{GAB} S \end{cases}$$

where $F[c_1 \rightarrow A_1, \dots, c_n \rightarrow A_n]$ means the FA structure obtained from F by substituting respectively the left to right occurrences of c_1, \dots, c_n by A_1, \dots, A_n .

- G generates a well-bracketed list of words $T \in \mathcal{T}_{\Sigma}$ iff there exists $F \in \mathcal{FA}_{\Sigma}$ such that $tree_{\Sigma}(F) = T$ and G generates F .
- G generates a string $c_1 \dots c_n \in \Sigma^+$ iff there exists $F \in \mathcal{FA}_{\Sigma}$ such that $yield_{\Sigma}(tree_{\Sigma}(F)) = c_1 \dots c_n$ and G generates F .
- The language of FA structures corresponding to G , written $\mathcal{FL}_{GAB}(G)$, is the set of FA structures generated by G .
- The language of well-bracketed lists of words corresponding to G , written $\mathcal{BL}_{GAB}(G)$, is the set of well-bracketed lists of words generated by G .
- The language of strings corresponding to G , written $\mathcal{L}_{GAB}(G)$, is the set of strings generated by G .

Example 3. If we take the categorial grammar that is defined in Example 1, we get:

$$He \text{ likes } Mary \quad \in \mathcal{L}_{GAB}(G_1)$$

$$((He \text{ likes}) \text{ Mary}) \quad \in \mathcal{BL}_{GAB}(G_1)$$

$$FApp(BApp(He, likes), Mary) \in \mathcal{FL}_{GAB}(G_1)$$

because we can build the following GAB deduction (where $N_2 = X_2/(N \setminus X_2)$) that entails $N \vdash N_2$:

⁴ In fact, given a FA structure F , there is at most one GAB deduction \mathcal{P} s.t. $FA_{Tp}(\mathcal{P}) = F$ (FA_{Tp} is injective)

$$\frac{\overbrace{N_1}^{He} \quad \overbrace{N_1 \setminus (S/N_2)}^{likes} \quad BApp \quad \overbrace{N}^{Mary}}{S/N_2} \quad \overbrace{N}^{Mary} \quad FApp}{S}$$

however: *Mary likes he* $\notin \mathcal{L}_{GAB}(G_1)$

3.3 NL and GAB Languages

In fact, there is a strong correspondence between *GAB* deductions and *NL* derivations. In particular Theorem 1 shows that the respective string languages and binary tree languages are the same.

Theorem 1 ([6]). *If A is an atomic formula, $\Gamma \vdash_{GAB} A$ iff $\Gamma \vdash_{NL} A$*

Corollary 1. $\mathcal{BC}_{NL}(G) = \mathcal{BC}_{GAB}(G)$ and $\mathcal{L}_{NL}(G) = \mathcal{L}_{GAB}(G)$

4 A Strict Hierarchy

For each $k \in \mathbb{N}$, we can consider the class \mathcal{C}_{NL}^k of languages corresponding to k -valued non-associative Lambek grammars (without product) that is the grammars with at most k types associated to each word of the lexicon. This section proves that this family forms a strict hierarchy (if the lexicon has at least 2 elements):

Theorem 2. $\forall k \in \mathbb{N} \quad \mathcal{C}_{NL}^k \subsetneq \mathcal{C}_{NL}^{k+1}$

For instance, a first very easy result is given by the fact that $\mathcal{C}_{NL}^0 \subsetneq \mathcal{C}_{NL}^1$ because $\mathcal{C}_{NL}^0 = \emptyset$ and \mathcal{C}_{NL}^1 contains the (finite) language $\{a\} = \mathcal{L}_{NL}(G)$ for the rigid grammar $G : a \vdash -S$.

4.1 Overview

Previous Works. A similar problem was solved by Kanazawa in [5] for the classes of k -valued classical categorial grammars. The proof scheme was as follows:

- Languages: for $k > 0$, $L_{AB,k} =_{def} \{a^i b a^i b a^i \mid 1 \leq i \leq 2k\}$
- Grammars:⁵ for $k > 0$,

$$G_k = \begin{cases} a \vdash -x, \\ (\cdots (S/x) \cdots /x) /y \underbrace{\cdots /x}_i /y \underbrace{\cdots /x}_i /y \underbrace{\cdots /x}_{i-1} & (1 \leq i \leq k) \\ b \vdash -y, \\ (x \setminus (\cdots \setminus (x \setminus (\cdots (S/x) \cdots /x) /y) /x) \cdots /x) \cdots & (k+1 \leq i \leq 2k) \end{cases}$$

⁵ In fact, the second type of a can be abbreviated as $S/x^i y x^i y^{i-1}$ and the second type of b can be abbreviated as $x^i \setminus (S/x^i y x^i)$.

- The language (for AB) of G_k is $L_{AB,k}$.
- Property: for $k > 0$, $L_{AB,k}$ is a $(k + 1)$ -valued language but is not a k -valued language for classical categorial grammars.

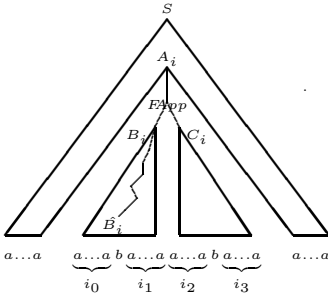
Towards NL. We can show easily that the languages of grammars G_k is the same when we consider the NL calculus instead of the AB rules : because the order of the types in the grammars are at most one, and in such a case it is well-known that the NL-languages and the AB-languages associated to a grammar are the same.

This shows that the languages $L_{AB,k}$ are also $(k + 1)$ -valued languages for NL. It is thus natural to ask whether they are k -valued for NL as well. In fact, when we proceed to a proof based on functor-argument structures, some of the arguments are no longer valid for NL, in the case of GAB-deductions.

We have thus constructed another language, based on the previous one, with some very similar proof steps. One key difference appears at stage 4.(e)(f) that does not apply to $L_{AB,k}$ that has only $2k$ words. An important point of our treatment for *NL* is precisely that the language constructed is both $k + 1$ -valued and with $2k + 1$ words.

The proof scheme is as follows:

1. Obviously, we have $\forall k \in \mathbb{N} \quad \mathcal{C}_{NL}^k \subseteq \mathcal{C}_{NL}^{k+1}$
2. For $k > 0$, we consider $L_{NL,k} =_{def} \{abb\} \cup \{a^i ba^i ba^i \mid 1 \leq i \leq 2k\}$
3. We prove that $L_{NL,k}$ is a $(k + 1)$ -valued language for NL languages.
Proof: for $k > 0$, We define a $k + 1$ -valued grammar and show that it is associated to $L_{NL,k}$. This part of the proof uses models of *NL* (see 4.2 below).
4. We prove that $L_{NL,k}$ is not a k -valued language for NL languages.
Proof : suppose G is a k -valued grammar with language $L_{NL,k}$
 - (a) For each element of $L_{NL,k}$, there exists a GAB deduction: \mathcal{T}_i for $a^i ba^i ba^i$ ($1 \leq i \leq 2k$) and \mathcal{T}_0 for abb .
 - (b) For $0 \leq i \leq 2k$, we define A_i as the root type of the smallest subtree in \mathcal{T}_i with a yield including both b :



This gives two subtrees with one b with yields $a^{i_0} ba^{i_1}$ and $a^{i_2} ba^{i_3}$ ($i_1 + i_2 = i$). Then, we consider $B_i = A_i / D_i$ (or $B_i = D_i \setminus A_i$) and C_i with $C_i \vdash_{NL} D_i$ the antecedents of A_i in \mathcal{T}_i and we define \widehat{B}_i as the type in G “providing” B_i (following functors) in \mathcal{T}_i .

- (c) We prove that: $\forall i, j : B_i \neq B_j$ (see 4.2 below)
- (d) and: $\forall i, j : \widehat{B}_i \neq \widehat{B}_j$ (see 4.2 below)
- (e) As a consequence, we need $2k + 1$ distinct \widehat{B}_i .

- (f) Contradiction: $2k + 1$ distinct \widehat{B}_i are needed with a k -valued grammar with an appropriate lexicon of 2 words (a and b).
5. Thus $\forall k > 0 \quad \mathcal{C}_{NL}^k \neq \mathcal{C}_{NL}^{k+1}$ (we have also seen in the introduction to the section that the property is also true for $k = 0$).

4.2 Details of Proof

- For (4.c): let $y_{ce}(X_i)$ denote the center part of the yield with root X_i (this is i_1 for the left subtree with yield $a^{i_0}ba^{i_1}$ and i_2 for the right subtree with yield $a^{i_2}ba^{i_3}$), we have $\forall i : y_{ce}(B_i) + y_{ce}(C_i) = i$.
Suppose (by contradiction) (i) $B_i = B_j$, for some $i \neq j$; from (i) we get $D_i = D_j$, and $C_i \vdash D_i$ then also $C_i \vdash D_j$, $C_j \vdash D_i$.
Since $i \neq j$, either $y_{ce}(B_i) \neq y_{ce}(B_j)$ or $y_{ce}(C_i) \neq y_{ce}(C_j)$.
Suppose first (ii) $y_{ce}(B_i) \neq y_{ce}(B_j)$; from (ii) replacing in T_j , ($j \neq 0$), B_j by B_i is a parse of a word $\dots ba^{j'}ba^j$ or $a^jba^{j'}b\dots$, where $j' = y_{ce}(B_i) + y_{ce}(C_j)$ this word is not in $L_{NL,k}$ since $j' = y_{ce}(B_i) + y_{ce}(C_j) \neq j$ this contradicts the assumption that G has $L_{NL,k}$ as language (for NL).
Suppose instead (ii)' $y_{ce}(C_i) \neq y_{ce}(C_j)$, replacing in T_j , ($j \neq 0$), C_j by C_i yields a similar word not in $L_{NL,k}$ with $j' = y_{ce}(B_j) + y_{ce}(C_i)$ occurrence of a between the b and $j' \neq j$, (ii)' also brings a contradiction.
Therefore (i) is not possible.
- For (4.d): we use the notation $X|Y$ as an abbreviation for X/Y or for $Y \setminus X$ (functor first). We show $\forall i, j : \widehat{B}_i \neq \widehat{B}_j$.
Suppose $\widehat{B}_i = \widehat{B}_j$, one (say B_i) is a head subtype of the other (B_j) then of the form:

$$B_j = \dots(B_i|D'_1\dots)|D'_n = A_j|D_j$$

we then get A_j in the subtree ending in B_i in T_i , which is impossible since the yield would have three b instead of two.

- For (3) the language description, we consider the $k + 1$ -valued grammar $\sigma(G_k)$ where G_k is as above, with substitution $\sigma = x := (S/y)/y$, and we show $\mathcal{L}_{NL}(\sigma(G_k)) = L_{NL,k}$.

- We show that $L_{NL,k} \subseteq \mathcal{L}_{NL}(\sigma(G_k))$ by:
For $(i = 0, abb) : (((S/y)/y), y), y) \vdash S$ we write $F_0 = ((S/y)/y)$.
For $(i \leq k, a^i ba^i ba^i) : (\dots(S/x^i y x^i y x^{i-1}, x) \dots, x), y, x) \dots, x), y, x) \dots, x) \vdash S$
and let $F_i = S/x^i y x^i y x^{i-1}$ denote the corresponding type of a .
For $(i > k, a^i ba^i ba^i) : (\underbrace{(x, \dots, (x, x^i \setminus S/x^i y x^i, x) \dots, x)}_i, y, x) \dots, x) \vdash S$
and let $F_i = x^i \setminus S/x^i y x^i$ denote the corresponding type of b .
- To show that $\mathcal{L}_{NL}(\sigma(G_k)) \subseteq L_{NL,k}$ we consider the following powerset residuated groupoid on V^* (also with unit):
 $\llbracket S \rrbracket = L_{NL,k}$, $\llbracket y \rrbracket = \{b\}$;
we then calculate the type images of $\sigma(F_i)$ (see above):
 $\llbracket \sigma(F_0) \rrbracket = \{a\}$ (with $\llbracket (S/y) \rrbracket = \{ab\}$)

if $(i \leq k)$ then $\llbracket \sigma(F_i) \rrbracket = \{a\}$,
 if $(i' > k)$ then $\llbracket \sigma(F_{i'}) \rrbracket = \{b\}$
 hence the language inclusion $(\Gamma \vdash S \text{ implies } \llbracket \Gamma \rrbracket \subseteq \llbracket S \rrbracket = L_{NL,k})$.

5 Variants

5.1 A Variant of NL

In this section, we propose an adaptation of *GAB* deductions to the variant NL_\emptyset (without product). The presentation of NL_\emptyset using GAB_\emptyset and generalized FA_\emptyset structures is original. This is done by substituting NL by NL_\emptyset for the conditions on the premises of the *GAB* rules and also by adding two new rules written $FApp_\emptyset$ and $BApp_\emptyset$ that correspond to a “pseudo-application” of a functor to an “empty” argument. In the second part of the section, we show how to adapt the previous hierarchy construction to NL_\emptyset with the use of GAB_\emptyset deductions.

Defining GAB_\emptyset Deductions for NL_\emptyset . Like NL , NL_\emptyset can be presented as a GAB_\emptyset deduction system. This construction enables us to define generalized functor-argument structures for NL_\emptyset proofs and as elements of languages of structures. The FA structures have two new unary combinators $FApp_\emptyset$ and $BApp_\emptyset$ that correspond to the application of an empty argument.

Definition 12 (GAB_\emptyset Deduction for NL_\emptyset). Generalized *AB* deductions with empty applications for NL_\emptyset (*GAB_\emptyset deductions*) over Tp are the deductions built from formulas on Tp (the base case) using the following conditional rules ($C \vdash_{NL_\emptyset} B$ must be valid in NL_\emptyset in the first two rules and $\vdash_{NL_\emptyset} B$ must be valid in NL_\emptyset in the last two rules):

$\frac{A/B \quad C}{A} FApp \quad \frac{C \quad B \setminus A}{A} BApp$ $C \vdash_{NL_\emptyset} B \text{ valid in } NL_\emptyset$	$\frac{A/B}{A} FApp_\emptyset \quad \frac{B \setminus A}{A} BApp_\emptyset$ $\vdash_{NL_\emptyset} B \text{ valid in } NL_\emptyset$
--	--

We now extend the notion of *FA* structure, written FA_\emptyset so as to include the case of an empty antecedent (empty argument).

Definition 13 (FA_\emptyset structures). Let \mathcal{E} be a set, a FA_\emptyset structure over \mathcal{E} is a tree where each leaf is labelled by an element of \mathcal{E} and each internal node is labelled by $FApp$ (forward application), $BApp$ (backward application), $FApp_\emptyset$ (forward empty application), $BApp_\emptyset$ (backward empty application):

$$FA_{\emptyset, \mathcal{E}} ::= \mathcal{E} \mid FApp(FA_{\emptyset, \mathcal{E}}, FA_{\emptyset, \mathcal{E}}) \mid BApp(FA_{\emptyset, \mathcal{E}}, FA_{\emptyset, \mathcal{E}}) \\
\mid FApp_\emptyset(FA_{\emptyset, \mathcal{E}}) \mid BApp_\emptyset(FA_{\emptyset, \mathcal{E}})$$

Definition 14 (FA_\emptyset structure of a GAB_\emptyset deduction for NL_\emptyset). To each GAB_\emptyset deduction \mathcal{P} for NL_\emptyset , we associate a FA_\emptyset structure, written $FA_{\emptyset, Tp}(\mathcal{P})$, such that each internal node corresponds to the application of a rule in \mathcal{P} and is labelled by the name of this rule and where the leaves are the same as in \mathcal{P} .

Now, using two axioms and $(/L)$ for proving $(A/B, B) \vdash A$ and two cuts, we find that $tree_{Tp}(FApp_{\emptyset}(F)) = tree_{Tp}(F) \vdash_{NL_{\emptyset}} A$. The other possibilities $((BApp_{\emptyset}), (FApp))$ or $(BApp)$ as first rule are very similar and the base case is obvious.

Proof of $\Gamma \vdash_{NL_{\emptyset}} A \Rightarrow \Gamma \vdash_{GAB_{\emptyset}} A$ (A atomic): This property results from an adaptation to NL_{\emptyset} of the alternative presentation of NL where contexts are in a limited form [9]. This presentation, as far as we know, is original. The proof of the equivalence of NL_{\emptyset} and this system that we call NL_{\emptyset}^* is given as an appendix to the paper:

$$\begin{array}{c}
\frac{}{A \vdash A} \text{Ax} \quad \frac{(C, B) \vdash A}{C \vdash A/B} /R^* \quad \frac{(A, C) \vdash B}{C \vdash A \setminus B} \setminus R^* \\
\\
\frac{B \vdash A}{\emptyset \vdash A/B} /R_{\emptyset}^* \quad \frac{A \vdash B}{\emptyset \vdash A \setminus B} \setminus R_{\emptyset}^* \\
\\
\frac{D \vdash C \quad \Delta[B] \vdash A}{\Delta[(B/C, D)] \vdash A} /L^* \quad \frac{D \vdash C \quad \Delta[B] \vdash A}{\Delta[(D, C \setminus B)] \vdash A} \setminus L^* \\
\\
\frac{\emptyset \vdash C \quad \Delta[B] \vdash A}{\Delta[B/C] \vdash A} /L_{\emptyset}^* \quad \frac{\emptyset \vdash C \quad \Delta[B] \vdash A}{\Delta[C \setminus B] \vdash A} \setminus L_{\emptyset}^*
\end{array}$$

If we have a derivation of $\Gamma \vdash_{NL_{\emptyset}} A$ with A atomic using this presentation, the first rule on the main branch of the derivation must be a left rule. For instance, for $(/L_{\emptyset}^*)$, Γ can be written $\Delta[B/C]$ and we have a derivation of $\vdash C$ and another one of $\Delta[B] \vdash A$. We can apply our hypothesis to the second derivation. At this point, we have a GAB_{\emptyset} deduction $\mathcal{P}[B]$ of $\Delta[B] \vdash_{GAB_{\emptyset}} A$. In this deduction, we replace the leaf node corresponding to B by a node corresponding to the conclusion of $(FApp_{\emptyset})$ rule:

$$\begin{array}{ccc}
\begin{array}{c} B/C \\ B \\ \vdots \\ \mathcal{P} \end{array} \rightarrow \begin{array}{c} B \\ \vdots \\ \mathcal{P} \end{array} & \xrightarrow{FApp_{\emptyset}} & \begin{array}{l} \text{This transformation gives a } GAB_{\emptyset} \text{ deduction corresponding} \\ \text{to } \Delta[B/C] \text{ since } \vdash_{NL_{\emptyset}} C. \text{ The other three possibilities for} \\ (\setminus L_{\emptyset}^*), (/L^*) \text{ or } (\setminus L^*) \text{ are similar and give respectively a new} \\ BApp_{\emptyset}, FApp \text{ or } BApp \text{ node and the base case where the} \\ \text{derivation is an axiom is obvious.} \end{array}
\end{array}$$

5.2 A Strict Hierarchy Theorem for NL_{\emptyset}

For each $k \in \mathbb{N}$, we consider the class $\mathcal{C}_{NL_{\emptyset}}^k$ of languages corresponding to k -valued NL_{\emptyset} Lambek grammars (without product). This section proves that this family also forms a strict hierarchy (if the lexicon has at least 2 elements):

Theorem 4. $\forall k \in \mathbb{N} \quad \mathcal{C}_{NL_{\emptyset}}^k \subsetneq \mathcal{C}_{NL_{\emptyset}}^{k+1}$

5.3 Details of Proof

We consider the same languages $L_{NL,k}$ as for NL .

- We first show that $L_{NL,k}$ is also $k+1$ -valued for NL_\emptyset using the same grammars $\sigma(G_k)$ as for NL and we show $\mathcal{L}_{NL_\emptyset}(\sigma(G_k)) = L_{NL,k}$.
 - the fact that $L_{NL,k} \subseteq \mathcal{L}_{NL_\emptyset}(\sigma(G_k))$ follows from the property that a sequent valid in NL is also valid in NL_\emptyset ;
 - the converse is obtained by the same powerset residuated semi-groupoid (with unit) as for NL (a model of both systems).
- We then have to show that $L_{NL,k}$ is not k -valued for NL_\emptyset .
 We proceed similarly as for NL , with the observation that in step 4.(b), A_i does give two subtrees with one b and is deduced by FApp or BApp, since if A was deduced by FAppvide or BAppvide, this would contradict the assumption that A_i is the smallest subtree with a yield including both b .

5.4 A Proof Revisited for AB (Classical)

We consider again the same languages $L_{NL,k}$ as for NL and show how to recover a hierarchy theorem for the classical system using the former construction.

- We first show that $L_{NL,k}$ is also $k+1$ -valued for AB using the same grammars $\sigma(G_k)$ as for AB and we show $\mathcal{L}_{AB}(\sigma(G_k)) = L_{NL,k}$.
 - the fact that $L_{NL,k} \subseteq \mathcal{L}_{AB}(\sigma(G_k))$ follows from the derivations schemas shown for NL that only involve AB rules. ⁷
 - the converse is deduced from the fact that a AB deduction is also a NL deduction: $\mathcal{L}_{AB}(\sigma(G_k)) \subseteq \mathcal{L}_{NL}(\sigma(G_k)) = L_{NL,k}$.
- We then have to show that $L_{NL,k}$ is not k -valued for AB .
 We proceed similarly as for NL where the derivations conditions $C_i \vdash D_i$ are replaced by $C_i = D_i$.

6 Conclusion

The paper studies two related lexicalized grammatical systems: non-associative Lambek grammars without product with (NL_\emptyset) and without (NL) antecedent. For each system, we prove that the classes of k -valued categorial grammars form a strict hierarchy of classes of languages. Thus, the notion of k -valued grammars is relevant for both systems: each $k \in \mathbb{N}$ defines a particular class of languages.

A second important contribution of the paper consists in defining a system of generalized AB deductions and their corresponding generalized functor-argument structures for NL_\emptyset (without product). This construction enables us to

⁷ An alternate justification is to start from $L_{AB,k} = \mathcal{L}(G_k)$, we get by substitution $L_{AB,k} \subseteq \mathcal{L}(\sigma(G_k))$ and finally consider the derivation for abb ($L_{NL,k} = L_{AB,k} \cup \{abb\}$).

define languages of structured sentences as for classical categorial grammars or for *NL* languages.

The result can not be adapted directly to other systems like non-associative Lambek calculus with product or associative Lambek calculus because the proofs depend on the existence of generalized AB deductions for both systems studied here and we do not know how to define such structures for those logical systems. Thus the questions of a strict hierarchy of languages for k -valued grammars are still open for them.

References

1. Bar-Hillel, Y.: A quasi arithmetical notation for syntactic description. *Language* **29** (1953) 47–58
2. Lambek, J.: The mathematics of sentence structure. *American mathematical monthly* **65** (1958) 154–169
3. Joshi, A.K., Shabes, Y.: Tree-adjoining grammars and lexicalized grammars. In: *Tree Automata and LGS*. Elsevier Science, Amsterdam (1992)
4. Gold, E.: Language identification in the limit. *Information and control* **10** (1967) 447–474
5. Kanazawa, M.: *Learnable Classes of Categorial Grammars*. Studies in Logic, Language and Information. Center for the Study of Language and Information (CSLI) and The European association for Logic, Language and Information (FOLLI), Stanford, California (1998)
6. Béchet, D., Foret, A.: k -valued non-associative lambek grammars are learnable from function-argument structures. In de Queiroz, R., Pimentel, E., Figueiredo, L., eds.: *Electronic Notes in Theoretical Computer Science*. Volume 84., Elsevier (2003) 1–13
7. Lambek, J.: On the calculus of syntactic types. In Jakobson, R., ed.: *Structure of language and its mathematical aspects*. American Mathematical Society (1961) 166–178
8. Kandulski, M.: The non-associative lambek calculus. In W. Buszkowski, W.M., Bentem, J.V., eds.: *Categorial Grammar*. Benjamins, Amsterdam (1988) 141–152
9. Aarts, E., Trautwein, K.: Non-associative Lambek categorial grammar in polynomial time. *Mathematical Logic Quaterly* **41** (1995) 476–484
10. Buszkowski, W.: Mathematical linguistics and proof theory. [14] chapter 12 683–736
11. Moortgat, M.: Categorial type logic. [14] chapter 2 93–177
12. de Groote, P.: Non-associative Lambek calculus in polynomial time. In: *8th Workshop on theorem proving with analytic tableaux and related methods*. Number 1617 in *Lecture Notes in Artificial Intelligence*, Springer-Verlag (1999) 128–139
13. de Groote, P., Lamarche, F.: Classical non-associative lambek calculus. *Studia Logica* **71(3)** (2002) 355–388
14. van Benthem, J., ter Meulen, A., eds.: *Handbook of Logic and Language*. North-Holland Elsevier, Amsterdam (1997)

A Proof of the Equivalence of NL_\emptyset and NL_\emptyset^*

Lemma 1. *NL_\emptyset and NL_\emptyset^* are two equivalent logical systems*

Proof. Of course because NL_\emptyset^* rules are restrictions of NL_\emptyset rules, a proof of a sequent $\Gamma \vdash A$ in NL_\emptyset^* is also a proof in NL_\emptyset . For the reverse implication, since we can eliminate cut in NL_\emptyset derivations, we suppose that we have only cut free NL_\emptyset derivations. In fact, we prove by induction on $k \in \mathbb{N}$ that we can rewrite a cut free NL_\emptyset derivation \mathcal{P} of a sequent $\Gamma \vdash A$ containing at most k operators $/$ and \backslash into a NL_\emptyset^* derivation.

- For $k = 0$, the sequent is $B \vdash A$ with A and B primitive types. \mathcal{P} can only be an axiom and $A = B$. Thus we have a NL_\emptyset^* derivation.
- If the last rule \mathcal{P} is an axiom, we already have a NL_\emptyset^* derivation.
- If the last rule is also a NL_\emptyset^* rule, by induction, we can find a proof of the premise(s) in NL_\emptyset^* (the sequents have less than k connectives) and build a proof in NL_\emptyset^* of the initial sequent.
- If the last rule of \mathcal{P} is $(/R)$ (the case of $(\backslash R)$ is symmetrical) but is not a NL_\emptyset^* rule, we have the following proof:

$$\frac{\begin{array}{c} \mathcal{P}_1 \\ \vdots \\ (\Gamma, B_1) \vdash A_1 \end{array}}{\Gamma \vdash A_1/B_1} /R \quad \text{Because the rule is not a } NL_\emptyset^* \text{ rule, } \Gamma \text{ is neither } \emptyset \text{ nor a type. By induction, there exists a } NL_\emptyset^* \text{ derivation } \mathcal{P}'_1 \text{ of } (\Gamma, B_1) \vdash A_1. \text{ We consider the last rule of } \mathcal{P}'_1. \text{ The possible rules are } (/L^*), (/L_\emptyset^*), (\backslash L^*) \text{ and } (\backslash L_\emptyset^*). \text{ The four cases are very similar and we just look here at the first one, the case where the rule is } (/L^*).$$

We have already mentioned that Γ is neither \emptyset nor a type. At this point we have the following NL_\emptyset^* deduction, where $\Gamma = \Delta_2[(B_2/C_2, D_2)]$:

$$\frac{\begin{array}{c} \mathcal{P}_2 \\ \vdots \\ D_2 \vdash C_2 \end{array} \quad \begin{array}{c} \mathcal{Q}_2 \\ \vdots \\ (\Delta_2[B_2], B_1) \vdash A_1 \end{array}}{(\Delta_2[(B_2/C_2, D_2)], B_1) \vdash A_1} /L^*$$

Now, we can consider the following NL_\emptyset derivation (each NL_\emptyset^* rule is considered as a NL_\emptyset rule inside the derivation):

$$\frac{\begin{array}{c} \mathcal{Q}_2 \\ \vdots \\ (\Delta_2[B_2], B_1) \vdash A_1 \end{array}}{\Delta_2[B_2] \vdash A_1/B_1} /R$$

By induction hypothesis, there exists a NL_\emptyset^* derivation \mathcal{Q}'_2 of $\Delta_2[B_2] \vdash A_1/B_1$ and we finally have a derivation of $\Gamma \vdash A$ that uses NL_\emptyset^* rules only:

$$\frac{\begin{array}{c} \mathcal{P}_2 \\ \vdots \\ D_2 \vdash C_2 \end{array} \quad \begin{array}{c} \mathcal{Q}'_2 \\ \vdots \\ \Delta_2[B_2] \vdash A_1/B_1 \end{array}}{\Delta_2[(B_2/C_2, D_2)] \vdash A_1/B_1} /L^*$$

- If the last rule of \mathcal{P} is $(/L)$ (the case of $(\backslash L)$ is symmetrical) but is not a NL_\emptyset^* rule, we have the following proof, where $\Gamma = \Delta_1[(B_1/C_1, \Gamma_1)]$:

$$\frac{\begin{array}{c} \mathcal{P}_1 \\ \vdots \\ \Gamma_1 \vdash C_1 \end{array} \quad \begin{array}{c} \mathcal{Q}_1 \\ \vdots \\ \Delta_1[B_1] \vdash A \end{array}}{\Delta_1[(B_1/C_1, \Gamma_1)] \vdash A} /L$$

Because the rule is not a NL_\emptyset^* rule, Γ_1 is neither \emptyset nor a type. By induction, there exists a NL_\emptyset^* derivation \mathcal{P}'_1 of $\Gamma_1 \vdash C_1$. We consider the last rule of \mathcal{P}'_1 . The possible rules are $(/L^*)$, $(/L_\emptyset^*)$, $(\backslash L^*)$ and $(\backslash L_\emptyset^*)$ (Γ_1 is neither \emptyset nor a type).

The four cases are very similar and we just look here at the first one, the case where the rule is $(/L^*)$. At this point we have the following NL_\emptyset^* deduction, where $\Gamma_1 = \Delta_2[(B_2/C_2, D_2)]$:

$$\frac{\begin{array}{c} \mathcal{P}_2 \\ \vdots \\ D_2 \vdash C_2 \end{array} \quad \begin{array}{c} \mathcal{Q}_2 \\ \vdots \\ \Delta_2[B_2] \vdash C_1 \end{array}}{\Delta_2[(B_2/C_2, D_2)] \vdash C_1} /L^*$$

Now, we can consider the following NL_\emptyset derivation (each NL_\emptyset^* rule is considered as a NL_\emptyset rule inside the derivation):

$$\frac{\begin{array}{c} \mathcal{Q}_2 \\ \vdots \\ \Delta_2[B_2] \vdash C_1 \end{array} \quad \begin{array}{c} \mathcal{Q}_1 \\ \vdots \\ \Delta_1[B_1] \vdash A \end{array}}{\Delta_1[(B_1/C_1, \Delta_2[B_2])] \vdash A} /L$$

By induction hypothesis, there exists a NL_\emptyset^* derivation \mathcal{Q}'_2 of $\Delta_1[(B_1/C_1, \Delta_2[B_2])] \vdash A$ and we finally have a derivation of $\Gamma \vdash A$ that uses NL_\emptyset^* rules only:

$$\frac{\begin{array}{c} \mathcal{P}_2 \\ \vdots \\ D_2 \vdash C_2 \end{array} \quad \begin{array}{c} \mathcal{Q}'_2 \\ \vdots \\ \Delta_1[(B_1/C_1, \Delta_2[B_2])] \vdash A \end{array}}{\Delta_1[(B_1/C_1, \Delta_2[(B_2/C_2, D_2)])] \vdash A} /L^*$$

■

Dependency Structure Grammars

Denis Béchet¹, Alexander Dikovsky², and Annie Foret³

¹ LINA, Université de Nantes, 2, rue de la Houssinière,

BP 92208 F 44322 Nantes cedex 3 France

denis.bechet@univ-nantes.fr

<http://www.sciences.univ-nantes.fr/info/perso/permanents/bechet/>

² LINA, Université de Nantes, 2, rue de la Houssinière,

BP 92208 F 44322 Nantes cedex 3 France

alexandre.dikovsky@univ-nantes.fr

<http://www.sciences.univ-nantes.fr/info/perso/permanents/dikovsky/>

³ IRISA - Université de Rennes 1, Campus Universitaire de Beaulieu,

Avenue du Général Leclerc., 35042 Rennes Cedex France

annie.foret@irisa.fr

Abstract. In this paper, we define Dependency Structure Grammars (DSG), which are rewriting rule grammars generating sentences together with their dependency structures, are more expressive than CF-grammars and non-equivalent to mildly context-sensitive grammars.

We show that DSG are weakly equivalent to Categorical Dependency Grammars (CDG) recently introduced in [6, 3]. In particular, these dependency grammars naturally express long distance dependencies and enjoy good mathematical properties.

1 Introduction

Dependency grammars (DGs) are formal grammars, which define syntactic relations between words in the sentences. Following to the tradition going back to L. Tesnière, the DGs are lexicalized and define the surface syntactic structure in terms of syntactic valences of individual words and of constraints imposed on valency saturation, in particular, on licensed feature values and on word order. There are numerous and rather different definitions of DGs (cf. [1, 2]). Most of them are not generative string or graph-substitution rule based grammars. This can be simply explained by the absence of substructure markers (nonterminals) in the dependency structures. Meanwhile, the formalization of dependency syntax in the form of generative style grammars is an important issue for various reasons. Firstly, such grammars allow for a straightforward interface relying compositional dependency structure with other compositional structures, for instance, with constituent structure or with semantic structure of some kind. Secondly, sometimes they allow for improvement of parsing performance, in particular, for disambiguation using meta-rules or other means of compact encoding of unifiable substructures. Thirdly but not lastly, the rule-based formal grammars have remarkable mathematical properties, which are the source of well

founded and efficient methods of analysis, translation, optimization and semantic interpretation of grammars.

Some definitions of generative dependency grammars can be found in the literature (cf. [7, 4, 5, 2]). In this paper, we develop the idea put forward in [4, 5] to distinguish between local and long distance dependencies and to treat them differently: the former by the composition of right-hand-side dependency trees of the rules, and the latter by the unique global rule of pairing a long distance dependency valency with the *first available* (i.e. the closest not used) dual valency: **FA**-rule. Recently, this idea was implemented in the form of calculus of syntactic types: *Categorical Dependency Grammars (CDG)* generalizing classical categorical grammars [6, 3]. In this paper, we dramatically simplify the rather technical definition of *polarized dependency grammars* of [4, 5] by renouncing the tree constraints and considering general graph dependency structures. The resulting *generalized Dependency Structure Grammars (gDSG)* prove to be weakly equivalent to *generalized Categorical Dependency Grammars (gCDG)* resulting from CDG by a similar dependency structure generalization. This equivalence of two completely different simple and elegant formal models shows the invariant nature of the rule **FA**. At the same time, this equivalence proves that the languages in this family have an efficient polynomial parsing algorithm due to their gCDG definition, and that they enjoy good mathematical properties due to their gDSG definition.

The paper is organized as follows. In section 2, we introduce the generalized Dependency Structure Grammars and some their important particular cases. In the next section, we summarize the main definitions and notation of the Categorical Dependency Grammars and define the generalized Categorical Dependency Grammars. In section 3, we prove the equivalence of the two definitions. Finally, in section 4, we establish the results characterizing the expressive power and main properties of this class of dependency grammars.

2 Dependency Structure Grammars

2.1 Dependency Valency

We follow the proposals in [5, 6] and specify long distance (in particular, non-projective discontinuous) dependencies by polarized dependency types, which we call *valences*. A *positive valency* specifies the name and the direction of an outgoing long distance dependency. The corresponding negative valency with the same name has the opposite direction and specifies the end of this incoming dependency (we say that the two valences are *dual*). Long distance dependencies are specified by correctly paired dual valences. In this pairing, positive valences needing the corresponding negative valency on the right and negative valences needing the corresponding positive valency on the right are considered as left brackets. Symmetrically, the valences needing the corresponding dual valences on the left are considered as right brackets.

For instance, the first member of the french discontinuous negation *ne .. pas* must have the left positive valency ($\nearrow n\text{-compound}$), whereas the second mem-

ber must have the dual right negative valency ($\searrow n$ -compound). Together they define the long distance dependency n -compound.

Formally, we consider a finite set \mathbf{C} of elementary dependency types and introduce four *polarities*: left and right positive: \nearrow, \nwarrow (*outgoing from left* (respectively, right) *to right* (respectively, left)) and left and right negative: \swarrow, \searrow (*incoming from right* (respectively, left) *to left* (respectively, right)). For each polarity v , there is the unique “dual” polarity \check{v} : $\nearrow = \searrow$, $\nwarrow = \swarrow$, $\swarrow = \nwarrow$, $\searrow = \nearrow$. A *polarized valency* is an expression (vC) , in which v is one of the four polarities and $C \in \mathbf{C}$. For instance, in the phrase *upon what dependency theory we rely*, the *right positive* valency ($\nwarrow \text{pre-UPON-obj}$) of the transitive verb *rely* requires the beginning of the long distance dependency *pre-UPON-obj* relating this verb with the subordinate object *dependency theory* dislocated from right to left and headed by the preposition ‘UPON’. The end of this dependency will be required by the type of the preposition *UPON* through the dual left negative valency ($\swarrow \text{pre-UPON-obj}$)¹.

$\nearrow \mathbf{C}, \nwarrow \mathbf{C}, \swarrow \mathbf{C}$ and $\searrow \mathbf{C}$ denote the corresponding sets of polarized valences. For instance, $\nearrow \mathbf{C} = \{(\nearrow C) \mid C \in \mathbf{C}\}$ is the set of *left positive* valences. $V^+(\mathbf{C}) = \nearrow \mathbf{C} \cup \nwarrow \mathbf{C}$ is the set of positive valences, $V^-(\mathbf{C}) = \swarrow \mathbf{C} \cup \searrow \mathbf{C}$ is the set of those negative.

2.2 Generalized Dependency Structures

Definition 1. Potentials. A potential is a string $\Gamma \in \mathcal{P} =_{df} (V^+(\mathbf{C}) \cup V^-(\mathbf{C}))^*$.

Let $\Gamma = \Gamma_1(vC)\Gamma_2(\check{v}C)\Gamma_3$ and $\Gamma' = \Gamma_1\Gamma_2\Gamma_3$ be two potentials such that $(vC) = (\nearrow A)$, $(\check{v}C) = (\searrow A)$ or $(vC) = (\swarrow A)$, $(\check{v}C) = (\nwarrow A)$. We say that (vC) is **first available (FA)** for $(\check{v}C)$ in Γ and both are neutralized in Γ' (denoted $\Gamma \rightarrow_{FA} \Gamma'$) if Γ_2 has no occurrences of (vC) and $(\check{v}C)$. This reduction of potentials \rightarrow_{FA} is terminal and confluent. So each potential Γ has a unique FA-normal form² denoted $[\Gamma]_{FA}$. Therefore, we can define the product \odot of potentials as follows: $\Gamma_1 \odot \Gamma_2 =_{df} [\Gamma_1\Gamma_2]_{FA}$.

Clearly, this product is associative. So we obtain the *monoid of potentials* $\mathbf{P} = (\mathcal{P}, \odot)$ under the product \odot with the unit ε .

Definition 2. Generalized dependency structures. Let W and N be two disjoint sets of terminals and nonterminals. A generalized dependency structure (gDS) over $W \cup N$ is a graph δ with linearly ordered nodes in which:

- the nodes are labeled by symbols in $W \cup N$,
- one maximal connected component D_0 and one node $n_0 \in D_0$ are selected, called respectively *head component* and *head* of δ ³. The decomposition of δ into maximal connected components (called below just *components*) will be denoted by $\delta = \{\underline{D}_0, D_1, \dots, D_k\}$.

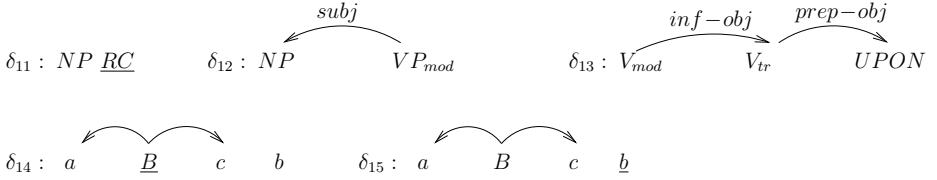
¹ See [6] and [3] for more details.

² Irreducible potential.

³ We visualize D_0 underlining its head n_0 if δ has at least two components.

Due to the linear order, δ determines the string of node labels $w(\delta) \in (W \cup N)^+$ called *framework* of δ . We will also say that δ is a *gDS* of $w(\delta)$. In particular, each component D_i is a *gDS* of the corresponding string $w(D_i)$.

Example 1. For instance, the following graphs are dependency structures:



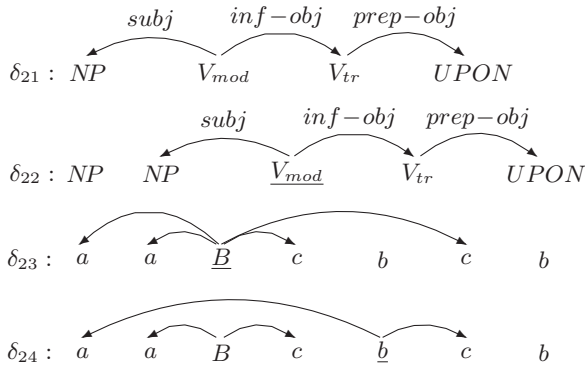
gDS δ_{11} has two components, the second is head. *gDS* δ_{12} , δ_{13} are dependency trees. The head of δ_{14} is B and the head of δ_{15} is b .

Definition 3. Composition of gDS. Let $\delta_1 = \{\underline{D_0}, D_1, \dots, D_k\}$ be a *gDS*. Let a nonterminal A have an occurrence in δ_1 : $w(\delta_1) = xAy$ and δ_2 be a *gDS* with the head n_0 . Then the **composition** of δ_2 into δ_1 in the selected occurrence of A , denoted $\delta_1[A \setminus \delta_2]$, is the *gDS* δ resulting from the union of δ_1 and δ_2 by unifying A and n_0 and by defining the order and labeling by the string substitution of $w(\delta_2)$ in the place of A in $w(\delta_1)$. Formally:

1. $nodes(\delta) =_{df} (nodes(\delta_1) - \{A\}) \cup nodes(\delta_2)$.
2. $arcs(\delta) =_{df} arcs(\delta_2) \cup (arcs(\delta_1) - \{d \in arcs(\delta_1) \mid \exists n(d = (A, n) \vee d = (n, A))\}) \cup \{(n_0, n) \mid \exists n((A, n) \in arcs(\delta_1))\} \cup \{(n, n_0) \mid \exists n((n, A) \in arcs(\delta_1))\}$.
3. The order of $nodes(\delta)$ is uniquely defined by equation $w(\delta) = xw(\delta_2)y$.
4. The head of δ is the head of the component resulting from D_0 .

$\delta = \delta_0[A_1, \dots, A_n \setminus \delta_1, \dots, \delta_n]$ will denote the result of simultaneous composition of *DS* $\delta_1, \dots, \delta_n$ into A_1, \dots, A_n in δ_0 .

Example 2. The following *gDS* are compositions of the *gDS* in example 1:



Namely, $\delta_{21} = \delta_{12}[VP_{mod} \setminus \delta_{13}]$, $\delta_{22} = \delta_{11}[CR \setminus \delta_{21}]$, $\delta_{23} = \delta_{14}[B \setminus \delta_{14}]$, $\delta_{24} = \delta_{14}[B \setminus \delta_{15}]$.

2.3 Grammar Definition

Definition 4. A **generalized Dependency Structure Grammar (gDSG)** is a system $G = (W, N, \mathbf{C}, S, R)$, where W, N and \mathbf{C} are finite sets of terminals (words), nonterminals and elementary types, $S \in N$ is the axiom and R is a finite set of rules. Each rule r consists of a substitution $s(r)$ of the form $A \rightarrow \delta$, where $A \in N$ and δ is a gDS, and of potential assignments of the form $\omega(r, a)[\Gamma]$, where $\omega(r, a)$ is an occurrence of a terminal a in δ and Γ is a (unique) potential in normal form⁴ assigned to this occurrence.

For each substitution $A \rightarrow \delta$, $A \rightarrow w(\delta)$ is the corresponding framework rule. The framework cf-grammar $f(G)$ consists of all framework rules of G .

Definition 5. Derivations. In definition 4, s is a many-to-one relation between the rules of G and $f(G)$. It is naturally extended to trees. A terminal derivation tree⁵ T_0 of $f(G)$ corresponds through s to a composition tree T of G if T results from T_0 by assigning to each non-terminal node n a rule $r(n) \in R$ such that $s(r)$ is applied to n in T_0 .

For each node n of a composition tree T , we define its potential $\pi(T, n)$ and gDS $gDS(T, n)$ induced by n in T as follows:

1. Let $n = a_i \in W$ be a terminal node of T , n' be its parent node, $\omega(r, a_i)$ be the occurrence of a_i in the right-hand side of rule $r = r(n')$ and $\omega(r, a_i)[\Gamma]$ be its potential assignment. Then $gDS(T, n) = a_i$ and $\pi(T, n) = \Gamma$. We suppose that each valency $v \in \Gamma$ keeps the position i of a_i in the generated string w (denoted v^i). The positions are needed only for gDS construction and can be neglected if the gDS are not pertinent.

2. Let $n = A \in N$ be a node in T with assigned rule $r(n) = (A \rightarrow \delta)$, whose framework rule is $A \rightarrow \alpha_1 \dots \alpha_k$. This means that n has in T k sons: n_1, \dots, n_k corresponding to $\alpha_1, \dots, \alpha_k$ (in this order). Let the potentials and the gDS of the sons be defined as: $\pi(T, n_i) = \Gamma_i$ and $gDS(T, n_i) = \delta_i$, $1 \leq i \leq k$. Then

$$\pi(T, n) =_{df} \Gamma_1 \odot \dots \odot \Gamma_k$$

$$gDS(T, n) =_{df} \delta[\alpha_1 \dots \alpha_k \setminus gDS(T, n_1) \dots gDS(T, n_k)] \cup \Delta_n,$$

where Δ_n is the set of all long distance dependencies ($a_i \xleftarrow{C} a_j$) or ($a_i \xrightarrow{C} a_j$) between terminals a_i, a_j , induced by neutralization of dual valences $(\swarrow C)^i$, $(\searrow C)^j$ (respectively $(\nearrow C)^i$, $(\nwarrow C)^j$).

The maximal length of potentials $\pi(T, n)$ in T is called valency deficit of T (denoted $\sigma(T)$).

A composition tree T is **derivation tree** if the potential of its root S is neutral: $\pi(T, S) = \varepsilon$. We set $G(D, w)$ if there is a derivation tree T of G from the axiom S , such that $D = gDS(T, S)$ and $w = w(D)$.

⁴ For instance, the rule $r = (A \rightarrow \underline{a}[\searrow D_1 \nearrow D_2] B)$ has the substitution $s(r) = (A \rightarrow \underline{a} B)$ and the assignment $\omega(r, a)[\searrow D_1 \nearrow D_2]$. We omit assignments $\omega(r, a)[\varepsilon]$.

⁵ I.e., in which all leaves are terminal.

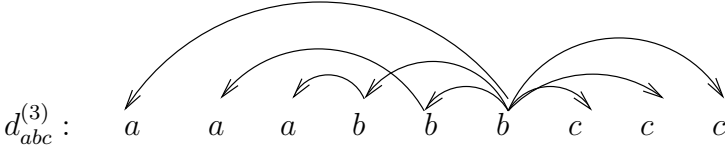
$\Delta(G) = \{D \mid \exists w \in W^+ \ G(D, w)\}$ is the **gDS-language** generated by G .
 $L(G) = \{w \in W^+ \mid \exists D \ G(D, w)\}$ is the **language** generated by G .

Intuitively, the derivation trees are induced by the framework grammar derivation trees, which correspond through s to composition trees. Only those composition trees derive gDS, in which all valencies are neutralized. Each derivation step can neutralize some dual dependency valences, and in this way, establish long distance dependencies between the words to which these valences are assigned.

Example 3. For instance, the gDSG

$$\begin{array}{c}
 S \rightarrow a[\swarrow a] \quad \underline{S} \quad | \quad A \quad c \\
 G_1 : \quad A \rightarrow b[\nwarrow a] \quad A \quad c \quad | \quad b[\nwarrow a]
 \end{array}$$

generates the language $L(G_1) = \{a^n b^n c^n \mid n \geq 1\}$ and the gDS-language $gDS(G_1) = \{d_{abc}^{(3)} \mid n \geq 1\}$, where e.g., $d_{abc}^{(3)}$ has the form:



The gDSG can generate dependency structures, which are arbitrary ordered graphs and not dependency trees. Even in the case, where the gDS in the rules have only dependency tree components, the generated structures may have cycles as it is the case of the following trivial gDSG:

$$S \rightarrow a[(\swarrow A)(\nearrow B)] \quad b[(\searrow B)(\nwarrow A)] \quad \underline{c}.$$

If we want that the grammars generate only dependency trees, then some additional constraints must be imposed.

2.4 Dependency Structure Grammars

We show the constraints, which guarantee only that the gDSG have the most important property of dependencies: *the uniqueness of the governor*. In particular, these constraints do not guarantee connectedness and cycle-freeness. The resulting Dependency Structure Grammars represent a reasonable compromise between acceptable divergence from classical dependency trees on the one hand, and simplicity of grammar rules and elimination of excess technical details on the other hand.

We split the set of nonterminals N in two parts: $N = N^+ \cup N^-$, $N^+ \cap N^- = \emptyset$. N^- corresponds to dependency structures with negative potential, and N^+ embodies the inherited through derivation impossibility of negative valences.

Definition 6. Dependency structures. *Let us call an oriented graph P unique governor if each node in P is entered by at most one arrow.*

A $gDS \delta = \{\underline{D}_0, D_1, \dots, D_m\}$ is a dependency structure (DS) if it is a unique governor graph and if each nonterminal B labelling a dependent node ⁶ is positive: $B \in N^+$.

Clearly, the composition preserves such dependency structures.

Proposition 1. *For any DS $\delta, \delta_1, \dots, \delta_k$, $\delta[A_1, \dots, A_n \setminus \delta_1, \dots, \delta_k]$ is a DS.*

Definition 7. *We call a potential Γ non-negative if $\Gamma \in (V^+(\mathbf{C}))^*$, neutral if $\Gamma = \varepsilon$ and definitely negative if $\Gamma \in (V^+(\mathbf{C}))^*V^-(\mathbf{C})(V^+(\mathbf{C}))^*$.*

Definition 8. *A Dependency Structure Grammar (DSG) is a $gDSG G = (W, N, \mathbf{C}, S, R)$, in which $N = N^+ \cup N^-$, $N^+ \cap N^- = \emptyset$, $S \in N^+$ and:*

- (c₁) *in potential assignments $\omega(r, a)[\Gamma]$, Γ is either neutral, or non-negative, or definitely negative;*
- (c₂) *in substitutions $r = (A \rightarrow \{\underline{D}_0, D_1, \dots, D_m\})$, if a terminal $a \in W$ labels a non-head node of a component of D_i or it labels the head n_0 of D_0 and $A \in N^+$, then only a non-negative potential Γ can be assigned to a through an assignment rule $\omega(r, a)[\Gamma]$;*
- (c₃) *if in a substitution $A \rightarrow \delta$ $A \in N^+$ and the head n_0 of δ is labeled with a nonterminal B , then $B \in N^+$.*

We denote the structure language of a DSG G by $DS(G)$. This notation is justified by the following proposition.

Proposition 2. *If G is a DSG, then $gDS(G)$ contains only terminal DS.*

Proof. Proposition 2 is immediately implied by the following lemma.

Lemma 1. *Let T be a derivation tree of a $gDS \delta_T$ with the head node a_h . Then:*

1. *If T is a derivation tree from a positive nonterminal $A \in N^+$, then a_h has no negative valences.*
2. *For all nodes n in T and for each terminal node a of $gDS(T, n)$, if among the valences assigned to a there is one not neutralized negative valency v , then a is not dependent in $gDS(T, n)$.*

Can be proven by induction on the structure of the derivation tree T . □

3 Categorical Dependency Grammars

In this section, we summarize the main notions related with the Categorical Dependency Grammars needed to define some their generalization.

⁶ I.e. a node, into which a dependency enters.

3.1 Dependency Types

Categorial dependency grammars are simply related with classical categorial grammars. They use “curried” variants of first order types: $[l_1 \setminus \dots \setminus m / \dots / r_1]$. In these types, all subtypes: left argument (l_i), right argument (r_i) and main (m) can be elementary or polarized. The elementary subtypes define local dependencies and the polarized subtypes define long distance dependencies. In particular, elementary left argument type l corresponds to the beginning of the local dependency l outgoing to the left, whereas main subtype l corresponds to the end of incoming local dependency l . As in DSG, the polarized subtypes represent long distance dependency valences. They have the same meaning. There is however a fundamental difference between the two formal models. In DSG, the linear order is directly defined by the right-hand-side gDS of rules. Categorial dependency grammars are completely lexicalized. To define a linear order on long distance dependencies, they use so called “anchored” valences. For instance, in the sentence *It was yesterday that they had this meeting* the discontinuous dependency *it*–*cleft* starting from the conjunction *that* must enter the expletive pronoun *It* in the position immediately preceding the main verb. To express this requirement, two adjacency markers: # and b are applied to dependency valences. Assigning to *It* the type $\#(\swarrow it\text{--}cleft)$ one requires that the long distance dependency *it*–*cleft* must enter *It* from the right and that the position of *It* must be *anchored* to some host word. To make *was* the host word for *It*, the type $[b(\swarrow it\text{--}cleft) \setminus S / subj / circ]$ is assigned to *was*. This type requires that the end of the long distance dependency *it*–*cleft* must immediately precede *was* (i.e. be *anchored* on its left), that two local dependencies *subj* and *circ* must start from *was* to its right and that *was* becomes the root of the dependency tree if the three requirements are met. Below we summarize the definitions of dependency types and type calculus and address the reader to [6, 3] for more details.

We call syntactic types *categories*. Let \mathbf{C} be a nonempty set of *elementary categories*. Elementary categories, e.g. *subj*, *inf-subj*, *dojb*, *det*, *modif*, etc. are dependency names. For instance, *subj* is the dependency, whose subordinate is a noun or a pronoun in the syntactic role of the subject and whose governor is a verb. Elementary categories may be *iterated*. For $a \in \mathbf{C}$, a^* denotes the corresponding *iterative* category. For instance, *modif*^{*} is the type of iterated category *modif*. For a set $X \subseteq \mathbf{C}$, $X^* = \{C^* \mid C \in X\}$. The elementary and iterated categories are *local*.

The negative valences in $V^-(\mathbf{C})$ do not constrain the position of the end of the required long distance dependency. So they are called *loose*.

To specify the positions of the ends of long distance dependencies, we use two markers: # (*anchor*) and b (*host*). For each negative valency $vC \in V^-(\mathbf{C})$, the expressions $\#(vC)$ and $b(vC)$ are the corresponding *anchor* and *host valences*. We distinguish *left-argument* and *right-argument* host valences and the corresponding *left* and *right positioned* anchor valences:

$$\begin{aligned}
 Host^l(\mathbf{C}) &=_{df} \{b^l(\alpha) \mid \alpha \in V^-(\mathbf{C})\}, & Anc^l(\mathbf{C}) &=_{df} \{\#^l(\alpha) \mid \alpha \in V^-(\mathbf{C})\}, \\
 Host^r(\mathbf{C}) &=_{df} \{b^r(\alpha) \mid \alpha \in V^-(\mathbf{C})\}, & Anc^r(\mathbf{C}) &=_{df} \{\#^r(\alpha) \mid \alpha \in V^-(\mathbf{C})\}, \\
 Host(\mathbf{C}) &=_{df} Host^l(\mathbf{C}) \cup Host^r(\mathbf{C}), & Anc(\mathbf{C}) &=_{df} Anc^l(\mathbf{C}) \cup Anc^r(\mathbf{C}).
 \end{aligned}$$

The sets $Host^l(\mathbf{C})$, $Host^r(\mathbf{C})$, $Anc^l(\mathbf{C})$ and $Anc^r(\mathbf{C})$ are supposed to be disjoint.

Definition 9. *The set $Cat(\mathbf{C})$ of categories is the least set such that:*

1. $\mathbf{C} \cup V^-(\mathbf{C}) \cup Anc(\mathbf{C}) \subset Cat(\mathbf{C})$.
2. For $C \in Cat(\mathbf{C})$, $A_1 \in (\mathbf{C} \cup \mathbf{C}^* \cup Host^l(\mathbf{C}) \cup \nwarrow \mathbf{C} \cup \searrow \mathbf{C})$ and $A_2 \in (\mathbf{C} \cup \mathbf{C}^* \cup Host^r(\mathbf{C}) \cup \nearrow \mathbf{C} \cup \swarrow \mathbf{C})$, the categories $[A_1 \setminus C]$ and $[C / A_2]$ also belong to $Cat(\mathbf{C})$.

Categories, which cannot have left arguments in $\searrow \mathbf{C}$ and right arguments in $\swarrow \mathbf{C}$ are called dependency categories (denoted $DCat(\mathbf{C})$); those which do not have subcategories in $V^-(\mathbf{C}) \cup V^+(\mathbf{C})$, are called continuous dependency categories (denoted $CCat(\mathbf{C})$).

We suppose that the constructors \setminus , $/$ are associative. So every complex category α can be presented in the form $\alpha = [L_k \setminus \dots L_1 \setminus C / R_1 \dots / R_m]$.

For instance, $[b^l(\swarrow \textit{clit} - \textit{dobj}) \setminus \textit{subj} \setminus S / \textit{aux}]$ is one of possible categories of an auxiliary verb in French, which defines it as the host word for a cliticized direct object, requires a local subordinate subject on its left and a local subordinate through dependency *aux* on its right.

3.2 Definition of Categorical Dependency Grammars

Definition 10. *A generalized Categorical Dependency Grammar (gCDG) is a system $G = (W, \mathbf{C}, S, \delta)$, where W is a finite set of words, \mathbf{C} is a finite set of elementary categories containing the selected category S , and δ - called lexicon - is a finite-set-valued function on W such that $\delta(a) \subset Cat(\mathbf{C})$ for each word $a \in W$. G is a Categorical Dependency Grammar (CDG) if $\delta(W) \subseteq DCat(\mathbf{C})$.*

We index categories by their positions in a string of categories related by G with a given sentence $w = a_1 \dots a_n$: α^i is a (positioned) category of a dependency structure with the root position a_i . As in gDSG, these indices serve only to define dependency structures.

Definition 11. *A D-sentential form of a sentence $w = a_1 \dots a_n \in W^+$ is a pair (Δ, Γ) , where Δ is an oriented labelled graph with the set of nodes $V = \{a_1, \dots, a_n\}$ and a set of arcs labeled by elementary categories, and Γ is a nonempty string of positioned categories.*

An initial D-sentential form of $w = a_1 \dots a_n$ is an expression $((V, \emptyset), C_1^1 \dots C_n^n)$, in which $C_i \in \delta(a_i)$ for all $1 \leq i \leq n$. D-sentential forms (Δ, S^j) are terminal.

gCDG derivations are proofs in the following dependency calculus.

Definition 12. *Sub-commutative dependency calculus (only left constructor rules R^l are presented; the corresponding right constructor rules R^r are similar).*

Local dependency rule:

$\mathbf{L}^l.$ $((V, E), \Gamma_1 C^i [C \setminus \beta]^j \Gamma_2) \vdash ((V, E \cup \{a_i \xleftarrow{C} a_j\}), \Gamma_1 \beta^j \Gamma_2)$ for $C \in \mathbf{C}$.

Iterative dependency rules:

$\mathbf{I}^l.$ $((V, E), \Gamma_1 C^i [C^* \setminus \alpha]^j \Gamma_2) \vdash ((V, E \cup \{a_i \xleftarrow{C} a_j\}), \Gamma_1 [C^* \setminus \alpha]^j \Gamma_2)$ for $C \in \mathbf{C}$.

$\mathbf{\Omega}^l.$ $((V, E), \Gamma_1 [C^* \setminus \alpha]^i \Gamma_2) \vdash ((V, E), \Gamma_1 \alpha^i \Gamma_2)$ for $C \in \mathbf{C}$.

Argument valency rule:

$\mathbf{V}^l.$ $((V, E), \Gamma_1 [\beta \setminus \alpha]^i \Gamma_2) \vdash ((V, E), \Gamma_1 \beta^i \alpha^i \Gamma_2)$, where β is a host or polarized valency.

Anchored dependency rule:

$\mathbf{A}^l.$ $((V, E), \Gamma_1 \#^l(\alpha)^i \flat^l(\alpha)^j \Gamma_2) \vdash ((V, E), \Gamma_1 \alpha^i \Gamma_2)$ for $\#^l(\alpha) \in \text{Anc}^l(\mathbf{C})$ and $\flat^l(\alpha) \in \text{Host}^l(\mathbf{C})$.

Sub-commutativity rule:

$\mathbf{C}^l.$ $((V, E), \Gamma_1 C^i \alpha^j \Gamma_2) \vdash ((V, E), \Gamma_1 \alpha^j C^i \Gamma_2)$ if $\alpha \in (V^-(\mathbf{C}) \cup V^+(\mathbf{C}))$ and
 (i) $C \in \text{Host}(\mathbf{C})$ or
 (ii) $C \in \text{Cat}(\mathbf{C})$ and C has no subexpressions $\alpha, \#(\alpha), \flat(\alpha)$, and $\check{\alpha}$.

Long distance dependency rule:

$\mathbf{D}^l.$ $((V, E), \Gamma_1 (\swarrow C)^i (\searrow C)^j \Gamma_2) \vdash ((V, E \cup \{a_i \xleftarrow{C} a_j\}), \Gamma_1 \Gamma_2)$ for $(\swarrow C) \in \swarrow \mathbf{C}$ and $(\searrow C) \in \searrow \mathbf{C}$.

The one-step provability relation in this calculus is denoted by \vdash^R , where R is one of the rules above, or just by \vdash , if R is irrelevant. The transitive closure of this relation is denoted by \vdash^* .

Besides this sub-commutative calculus, we consider its restriction to the continuous categories in $\text{CCat}(\mathbf{C})$ with the additional equivalence $\#^\alpha(t) \equiv \flat^\alpha(t)$ and to the first three rules \mathbf{L}, \mathbf{I} and $\mathbf{\Omega}$. We call this restricted calculus projective.

The one-step provability relation in the projective calculus is denoted by \vdash_p^R (or just \vdash_p). Its transitive closure is denoted by \vdash_p^* .

We see that rule \mathbf{L} is a direct analogue of the classical elimination rule. Rules \mathbf{I} and $\mathbf{\Omega}$ extend \mathbf{L} to the iterative categories. In projective calculus, anchor and host types are not distinguished, e.g. $[\alpha/\flat^r(d)]\#^r(d) \vdash_p \alpha$. Particular are the polarized valences' rules. Rule \mathbf{V} extracts non-local valences from complex categories. Rule \mathbf{C} moves the valences in the indicated directions towards the **first available** valency, to which one can apply rules \mathbf{A} or \mathbf{D} . Rule \mathbf{D} adds a long distance dependency C , when two loose dual valences with the same name C become adjacent. The crucial difference between gCDG and CDG is that due to negative argument subtypes available in gCDG, the rule \mathbf{D} can violate the uniqueness of the governor, which is impossible in CDG, where non-local argument subtypes are positive or host. Rule \mathbf{A} verifies that an anchored valency $\#(\alpha)$ has become adjacent to the corresponding host valency $\flat(\alpha)$, consumes $\flat(\alpha)$ and loses $\#(\alpha)$. Intuitively, this means that α is well-placed with respect to the

category with the corresponding host argument. If this test succeeds, α becomes available to the long distance dependency rule **D**. We address the reader to [6, 3] for linguistic examples.

Definition 13. Let $G = (W, \mathbf{C}, S, \delta)$ be a *gCDG*. A *gDS* D is assigned by G to a sentence w (denoted $G(D, w)$) if $(\Delta_0, \Gamma_0) \vdash^* (D, S^j)$ for some initial sentential form (Δ_0, Γ_0) of w and some $1 \leq j \leq n$.

The *D-language* generated by G is the set of *gDS* $gDS(G) =_{df} \{D \mid \exists w G(D, w)\}$. The language generated by G is the set of sentences $L(G) =_{df} \{w \mid \exists D G(D, w)\}$.

Proposition 3. 1. For each *CDG* G , $gDS(G)$ contains only *DS*.

2. If *gCDG* is projective, it is a *CDG* and $DS(G)$ contains only projective *DS*.

We denote by $\mathcal{L}(gCDG)$, $\mathcal{L}(CDG)$ and $\mathcal{L}(pCDG)$ the families of languages generated by *gCDG*, *CDG* and projective *CDG*. If G is a *CDG*, then we use notation $DS(G)$ in the place of $gDS(G)$.

gCDG have the following fundamental property established in [3].

Definition 14. Local projection $\|\gamma\|_l$ of $\gamma \in Cat(\mathbf{C})^*$ is defined as follows:

11. $\|\varepsilon\|_l = \varepsilon$; $\|C\gamma\|_l = \|C\|_l \|\gamma\|_l$ for $C \in Cat(\mathbf{C})$ and $\gamma \in Cat(\mathbf{C})^*$.
12. $\|C\|_l = C$ for $C \in \mathbf{C} \cup \mathbf{C}^* \cup Anc(\mathbf{C})$.
13. $\|C\|_l = \varepsilon$ for $C \in V^+(\mathbf{C}) \cup V^-(\mathbf{C})$.
14. $\|[\alpha]\|_l = \|\alpha\|_l$ for all $\alpha \in Cat(\mathbf{C})$.
15. $\|[a\backslash\alpha]\|_l = [a\backslash \|\alpha\|_l]$ and $\|[\alpha/a]\|_l = [\|\alpha\|_l/a]$ for $a \in \mathbf{C} \cup \mathbf{C}^* \cup Host(\mathbf{C})$ and $\alpha \in Cat(\mathbf{C})$.
16. $\|[(\backslash a)\backslash\alpha]\|_l = \|[\alpha/(\nearrow a)]\|_l = \|\alpha\|_l$ for all $a \in \mathbf{C}$ and $\alpha \in Cat(\mathbf{C})$. Valency projection $\|\gamma\|_v$ of a string $\gamma \in Cat(\mathbf{C})^*$ is defined as follows:
 - v1. $\|\varepsilon\|_v = \varepsilon$; $\|C\gamma\|_v = \|C\|_v \|\gamma\|_v$ for $C \in Cat(\mathbf{C})$ and $\gamma \in Cat(\mathbf{C})^*$.
 - v2. $\|C\|_v = \varepsilon$ for $C \in \mathbf{C} \cup \mathbf{C}^*$.
 - v3. $\|C\|_v = C$ for $C \in V^+(\mathbf{C}) \cup V^-(\mathbf{C})$.
 - v4. $\|\#(C)\|_v = C$ for $C \in V^-(\mathbf{C})$.
 - v5. $\|[\alpha]\|_v = \|\alpha\|_v$ for all $[\alpha] \in Cat(\mathbf{C})$.
 - v6. $\|[a\backslash\alpha]\|_v = \|[\alpha/a]\|_v = \|\alpha\|_v$ for $a \in \mathbf{C} \cup \mathbf{C}^* \cup Host(\mathbf{C})$.
 - v7. $\|[a\backslash\alpha]\|_v = a \|\alpha\|_v$, if $a \in V^+(\mathbf{C})$.
 - v8. $\|[\alpha/a]\|_v = \|\alpha\|_v a$, if $a \in V^+(\mathbf{C})$.

Definition 15. For a category $C = [\alpha D^* \backslash \beta]$, the categories $[\alpha\beta]$, $[\alpha D \backslash \beta]$, $[\alpha D \backslash D \backslash \beta]$, $[\alpha D \backslash D \backslash D \backslash \beta]$, etc. are realizations of C (similar for right iterative categories). To obtain a realization of a string of categories $\gamma \in Cat(\mathbf{C})^+$, each of its elements having iterative subcategories should be replaced by one of its realizations. Let $R(\gamma)$ denote the set of all realizations of γ .

Theorem 1. Let $G = (W, \mathbf{C}, S, \delta)$ be a *gCDG*. $x \in L(G)$ iff there is a string of categories $\alpha \in \delta(x)$ such that for some its realization $\gamma \in R(\alpha)$:

1. $\|\gamma\|_l \vdash_p^* S$, 2. $\|[\gamma]\|_v \vdash_{FA} \varepsilon$.

In fact, this property is proven for *CDG* but the proof holds for *gCDG* too.

Corollary 1. [3] *There is a polynomial time parsing algorithm for gCDG.*

4 Expressive Power of gDSG

Definition 16. A gDSG $G = (W, N, \mathbf{C}, S, R)$ is in generalized Greibach normal form (GNF) iff for each rule $A \rightarrow \delta \in R$, $w(\delta) \in WN^*$.

Remark 1 The condition $w(\delta) \in WN^*$ is the conjunction of three conditions: (i) all $w(\delta)$ are not empty, (ii) the first symbol of $w(\delta)$ must be a terminal, (iii) all other symbols in $w(\delta)$ must be non-terminals.

The first condition is always true for gDSG and the third one is not difficult to obtain because it is always possible to introduce, for each terminal, a new non-terminal that replaces it in the right members of the rules, where the condition is not true. Thus, only the second condition is not trivial.

Proposition 4. For any gDSG G , a weakly equivalent gDSG G' in generalized GNF can be constructed.

Proof. Let $G = (W, N, \mathbf{C}, S, R)$ be a gDSG. As we are interested only in weak equivalence, we can chose arbitrary heads and dependencies to transform the frame rules to the form $N \rightarrow W(W \cup N)^*$. We follow the Greibach's construction and proceed by induction on the number of *critical* non-terminals, i.e. the non-terminals occurring in the first position of right-hand-sides of framework rules:

$$n = \#(\{A \in N \mid \exists (B \rightarrow \delta) \in R (w(\delta) \in A(W \cup N)^*)\}).$$

- In the case of $n = 0$, we already have a gDSG in generalized GNF.
- If $n > 0$, let A be one of these n non-terminals. Let A' be a new non-terminal and $N' =_{df} N \cup \{A'\}$. Let us classify the rules of R corresponding to the following framework rules:

$$A \rightarrow A \tag{1}$$

$$A \rightarrow B_1 \cdots B_k \quad k \geq 1, B_1 \cdots B_k \in (W \cup N)^+, B_1 \neq A \tag{2}$$

$$A \rightarrow AB_1 \cdots B_k \quad k \geq 1, B_1 \cdots B_k \in (W \cup N)^+ \tag{3}$$

$$C \rightarrow AB_1 \cdots B_k \quad k \geq 0, B_1 \cdots B_k \in (W \cup N)^*, C \in N, C \neq A \tag{4}$$

For $1 \leq i \leq 4$, we denote $R_{(i)} \subset R$ the rules in the class (i). The rules in $R_{(3)}$ and $R_{(4)}$ need to be modified. We define successively:

$$R_A = R_{(2)} \cup \{A \rightarrow \delta A' \mid A \rightarrow \delta \in R_{(2)}\}$$

$$R_{A'} = \{A' \rightarrow \delta[A \setminus \epsilon] \mid A \rightarrow \delta \in R_{(3)}\} \cup \{A' \rightarrow \delta[A \setminus \epsilon]A' \mid A \rightarrow \delta \in R_{(3)}\}$$

$$R_C = \{A' \rightarrow \delta[A \setminus \delta'] \mid C \rightarrow \delta \in R_{(4)} \wedge A \rightarrow \delta' \in R_A\}$$

$$R' = (R - R_{(1)} - R_{(2)} - R_{(3)} - R_{(4)}) \cup R_A \cup R_{A'} \cup R_C$$

$$G' = (W, N', \mathbf{C}, S, R')$$

The framework languages of G and G' are the same. Let T be a derivation tree of a string w in G . There exists a derivation tree T' of w in the framework grammar of G' . In T , each leaf is associated to a potential. Let us keep in T' the same potentials assignment as in T and extend the frame rules to the corresponding dependency structure rewriting rules. Then T' will become a composition tree in G' . Given that the product \odot is associative, in the transformed tree T' exactly the same potential is calculated. So T' is a derivation tree for w in G' , which proves that $w \in L(G')$ and $L(G) \subseteq L(G')$. The reverse inclusion is similar, so G and G' are weakly equivalent. Now, the induction hypothesis can be applied because the critical non-terminals of G' are fewer than those of G . \square

Theorem 2. *gDSG and gCDG are weakly equivalent.*

Proof. (\Rightarrow) To prove that $\mathcal{L}(gDSG) \subseteq \mathcal{L}(gCDG)$, we use a gDSG in generalized GNF. Let $G = (W, N', \mathbf{C}, S, R')$ be such a gDSG. We will simulate G by the gCDG $G' = (W, \mathbf{C}, S, \lambda)$, where the lexicon λ is computed from G as follows.

Let $r = (A \rightarrow \delta) \in R$ be a rule of G , whose framework rule has the form $A \rightarrow aB_1 \cdots B_i, a \in W$, and let $\omega(r, a)[\Gamma]$ be a potential assignment. Keeping in mind the associativity of potential product and the sub-commutativity rule \mathbf{C} , we can group together similar valences and represent Γ in the form:

$$\Gamma \equiv (\backslash C_1) \cdots (\backslash C_j) (\swarrow D_1) \cdots (\swarrow D_k) (\searrow E_1) \cdots (\searrow E_l) (\nearrow F_n) \cdots (\nearrow F_n).$$

To these rules we associate in $\lambda(a)$ the category:

$$\begin{aligned} & (\backslash C_1) \backslash \cdots \backslash (\backslash C_j) \backslash (\swarrow D_1) \backslash \cdots \backslash (\swarrow D_k) \backslash A/B_1 / \cdots \\ & \cdots / B_i / (\searrow E_1) / \cdots / (\searrow E_l) / (\nearrow F_n) / \cdots / (\nearrow F_n). \end{aligned}$$

The equivalence $L(G) = L(G')$ is relatively evident ⁷. The first part $L(G) \subseteq L(G')$ holds because a derivation tree of any string $w \in L(G)$ uniquely determines a sequence of reduction steps of categories assigned to w by G' . Indeed, the potential of a leaf of the derivation tree constitutes the part of the category determining the same long distance dependencies of a as those defined by the rule r . The rest of the category is uniquely determined by the rule r . One should first eliminate all long distance dependency valences (which is always possible), and then apply the category to its argument subtypes. This application is also possible because it directly simulates the application of the framework rule $w(r)$. This means that, using this tactics, the sequence of categories assigned by λ to the string w following the structure of the derivation tree of w in G will be reduced to S .

The converse inclusion $L(G') \subseteq L(G)$ is similar and follows from the fact that in a reduction to S of categories assigned by the lexicon λ , we can always start

⁷ This construction cannot serve to prove the strong equivalence, because in the case, when the head valency is negative, the resulting type has a negative argument subtype, which is impossible in CDG.

with reductions of long distance dependencies and continue with reductions of local dependencies.

(\Leftarrow) The converse relation between the two families is stronger: for each gCDG $G_1 = (W, \mathbf{C}, S, \lambda)$, we can construct a gDSG $G_2 = (W, N, \mathbf{C}, S, R)$ such that $\Delta(G_2) = \Delta(G_1)$. This strong simulation is implied by theorem 1. Namely, the grammar G_2 is defined as the union $\bigcup_{a \in W, C \in \lambda(a)} \mathcal{M}(a, C)$, where each module

$\mathcal{M}(a, C)$ is defined as follows.

Let us suppose for simplicity that in $\|C\|_l = [\alpha \setminus B / \beta]$ $\alpha \neq \varepsilon$ and $\beta = \varepsilon$. The three other cases are similar. Then $\alpha = B_n \setminus \dots \setminus B_1$ for some $n > 0$. In this case,

$$\mathcal{M}(a, C) =_{df} \{r^{(0)}, r^{(1)}, \dots, r^{(n)}, r^{(n+1)}\},$$

where $r^{(0)} = (M_C \rightarrow \Lambda \underline{M}_{aC}^{(1)} \Lambda)$, $M_C = B$, if $B \neq \varepsilon$ and $M_C = E$ otherwise, $r^{(n+1)} = (M_{aC}^{(n+1)} \rightarrow a[\|C\|_v])$, $\Lambda \in \{E, \varepsilon\}$, and the resting rules $r^{(i)}$ are as follows:

$$r^{(i)} = (M_{aC}^{(i)} \rightarrow \Lambda \quad \overset{\curvearrowright}{B_i \quad \Lambda \quad \underline{M}_{aC}^{(i+1)}})$$

if B_i is not iterative and

$$r^{(i)} = (M_{aC}^{(i)} \rightarrow \Lambda \quad \overset{\curvearrowright}{B_i \quad \Lambda \quad \underline{M}_{aC}^{(i)}} \quad | \quad \Lambda \quad \underline{M}_{aC}^{(i+1)})$$

if it is iterative. In this construction, E and $M_{aC}^{(i)}$ are new pairwise different non-terminals different from all types. The equality $\Delta(G_2) = \Delta(G_1)$ immediately follows from theorem 1. \square

Without constraint that gDS must be trees, the main result of [5] can be easily carried over to gDSG.

Theorem 3. *If in a gDSG G the valency deficit $\sigma(T)$ of correct terminal derivation trees is uniformly bounded by a constant c then G generates a CF-language.*

Proof. We can simply consider nonterminals $A[\Gamma]$, where Γ is a potential of the size not exceeding c , and define the rules so that for each node n of a complete derivation tree T its label should be $A[\pi(T, n)]$, A being the original nonterminal label of n . Clearly, $S[\varepsilon]$ becomes the axiom. \square

Using the classical constructions, one can easily prove that $\mathcal{L}(gDSG)$ is an abstract family of languages.

Proposition 5. *$\mathcal{L}(gDSG)$ is closed under ε -free homomorphism, inverse homomorphism, intersection with regular sets, union, concatenation and $+$.*

Seemingly, $\mathcal{L}(gDSG)$ is not closed under intersection and complementation.

Conjecture. *The copy language $L_{copy} = \{wcw \mid w \in \{a, b\}^*\}$ cannot be generated by a gDSG.*

Meanwhile, the complement of L_{copy} is linear and so belongs to $\mathcal{L}(gDSG)$. It is also well-known that L_{copy} is generated by a basic TAG. On the other hand, in [6, 3] it is proven that each language $L^{(m)} = \{d_0 a_0^n d_1 a_1^n \dots d_m a_m^n d_{m+1} \mid n \geq 0\}$ is generated by a CDG. So they can be generated by gCDG. Meanwhile, starting from $m = 5$, the languages $L^{(m)}$ cannot be generated by basic TAG. The languages $L^{(m)}$ are mildly context-sensitive [9]. This leads to the question of comparison of mildly CS languages and gDSG-languages. Seemingly, the two families are incomparable. Indeed, there is another strong conjecture that the mildly CS grammars cannot generate the language *MIX* of Emmon Bach consisting of all permutations of words $a^n b^n c^n, n > 0$:

$$MIX = \{w \in \{a, b, c\}^+ \mid |w|_a = |w|_b = |w|_c\}.$$

At the same time, we show that *MIX* is generated by a CDG.

Theorem 4. *There is a CDG generating MIX.*

Proof. We can construct a CDG G_{mix} with only loose valences and with only anchored valences. We show the former, because it is simpler:

TABLE OF	CATEGORY	ASSIGNMENTS
left	right	middle
$a \mapsto [\backslash B \setminus \backslash C \setminus S]$	$a \mapsto [S / \nearrow C / \nearrow B]$	$a \mapsto [\backslash B \setminus S / \nearrow C], [\backslash C \setminus S / \nearrow B]$
$a \mapsto [\backslash B \setminus \backslash C \setminus S \setminus S]$	$a \mapsto [S \setminus S / \nearrow C / \nearrow B]$	$a \mapsto [\backslash B \setminus S \setminus S / \nearrow C], [\backslash C \setminus S \setminus S / \nearrow B]$
$b \mapsto \nearrow B$	$b \mapsto \setminus B$	
$c \mapsto \nearrow C$	$c \mapsto \setminus C$	

Inclusion (\subseteq). $L(G_{mix}) \subseteq MIX$.

Let us consider the following *commutative* group interpretation of non-iterative categories (where $k_{l,x}, k_{r,x}$ are new symbols for each elementary x):

$$\begin{aligned} < p > = p \text{ for elementary } p, \\ < [x \setminus y] > = < x >^{-1} < y >, < [y / x] > = < y > < x >^{-1}, \\ < \nearrow x > = k_{l,x}^{-1}, < \setminus x > = k_{r,x}^{-1}, < \nearrow x > = k_{l,x}, < \setminus x > = k_{r,x}. \end{aligned}$$

Fact. $\Gamma \vdash S$ implies $< \Gamma > = S$. (By evident induction on the derivation length.)

Being applied to the categories of G_{mix} , this interpretation shows that the number of a , b and c is the same in all $w \in L(G_{mix})$.

Inclusion (\supseteq). $MIX \subseteq L(G_{mix})$.

Let us consider a word $w_0 \in MIX$. We construct a canonical assignment of categories to the occurrences of a, b, c in w_0 as follows.

Canonical assignment algorithm CCA

$w := w_0$;

WHILE $w \not\models \varepsilon$

DO

Phase I. Basic triangulation

FIND in w the *leftmost* occurrences α, β such that:

$w = u_1 \alpha u_2 \beta u_3$, where $u_2 \in c^*$, $\alpha \not\models \beta$, $\alpha, \beta \in \{a, b\}$;

FIND in w the occurrence γ of c *closest* to α , if $\alpha = a$, else closest to β ;

IF the selected $a \in \{\alpha, \beta\}$ is leftmost in w_0

THEN $X := S$;

ELSE $X := S \setminus S$

END;

CASE

$w = v_1 \gamma v_2 \alpha v_3 \beta v_4 \wedge \alpha = a \rightarrow \alpha := [\setminus C \setminus X / \setminus B]; \gamma := \setminus C; \beta := \setminus B$;

$w = v_1 \gamma v_2 \alpha v_3 \beta v_4 \wedge \alpha = b \rightarrow \beta := [\setminus B \setminus \setminus C \setminus X]; \gamma := \setminus C; \alpha := \setminus B$;

$w = v_1 \alpha v_2 \gamma v_3 \beta v_4 \wedge \alpha = a \rightarrow \alpha := [X / \setminus B / \setminus C]; \gamma := \setminus C; \beta := \setminus B$;

$w = v_1 \alpha v_2 \gamma v_3 \beta v_4 \wedge \alpha = b \rightarrow \beta := [\setminus C \setminus \setminus B \setminus X]; \gamma := \setminus C; \alpha := \setminus B$;

$w = v_1 \alpha v_2 \beta v_3 \gamma v_4 \wedge \alpha = a \rightarrow \alpha := [X / \setminus C / \setminus B]; \gamma := \setminus C; \beta := \setminus B$;

$w = v_1 \alpha v_2 \beta v_3 \gamma v_4 \wedge \alpha = b \rightarrow \beta := [\setminus B \setminus X / \setminus C]; \gamma := \setminus C; \alpha := \setminus B$

END;

Phase II. Elimination

$w := v_1 v_2 v_3 v_4$

END

It is easy to see that **CCA** exits successfully the loop on the condition $w = \varepsilon$ for each $w_0 \in MIX$. Being applied to w_0 , **CCA** defines the canonical assignment of categories **CCA**(w_0). The inclusion $MIX \subseteq L(G_{mix})$ is implied by the following fact.

Fact. **CCA**(w_0) $\vdash S$ holds for all $w_0 \in MIX$.

(By evident induction on the number of a .)

□

5 Conclusions

We can resume the relations between structure languages and languages generated by the dependency grammars considered in this paper as follows:

$$\mathcal{D}(CDG^{proj}) \subsetneq \mathcal{D}(CDG) \subsetneq \mathcal{D}(gCDG) \subseteq \mathcal{D}(gDSG) \text{ and} \\ CFL = \mathcal{L}(CDG^{proj}) = \mathcal{L}(gDSG^{\sigma < \omega}) \subsetneq \mathcal{L}(CDG) \subseteq \mathcal{L}(gCDG) = \mathcal{L}(gDSG),$$

where CDG^{proj} is the class of projective CDG and $gDSG^{\sigma < \omega}$ is the class of gDSG with bounded valency deficit.

The dependency structure and categorial dependency grammars can be easily adopted to practical large scale definitions of surface dependency syntax of natural languages. For this, one should relate dependency names with bounded length feature value products admitting feature unification and value propaga-

tion through dependencies. Besides this, the explicit use of anchored categories in DSG and CDG make possible to express a variety of word order constraints. In fact, the potential assignments are closely related to Debusmann and Duchier’s formulation of dependency grammar [8]. However, the FA-constraint excludes crossing of similarly labeled long distance dependencies.

CDG and DSG have an efficient parsing algorithm ($\mathbf{O}(n^5)$ in the worst case) [3]. In practice, the valency deficit is bounded by a small constant (2 or 3). In this situation, this parsing algorithm has complexity $\mathbf{O}(n^3)$ even if there are discontinuous long distance dependencies. So the dependency grammars studied in this paper represent an interesting class of grammars competitive with respect to mild context-sensitive grammars.

References

1. Les grammaires de dépendance. In Sylvain Kahane, editor, *Traitement automatique des langues*, volume 41, Paris, 2000. Hermes.
2. Proc. of the workshop “Recent Advances in Dependency Grammars”. in conjunction with coling 2004. In Geert-Jan M. Kruijff and Denys Duchier, editors, “*Recent Advances in Dependency Grammars*”. *COLING’04 Workshop, August 28 2004*, Geneva, Switzerland, August 2004.
3. Michael Dekhtyar and Alexander Dikovsky. Categorical dependency grammars. In *Proc. of Int. Conf. on Categorical Grammars*, pages 76–91, Montpellier, France, 2004.
4. Alexander Dikovsky. Grammars for local and long dependencies. In *Proc. of the 39th Intern. Conf. ACL’2001*, pages 156–163. ACL & Morgan Kaufman, 2001.
5. Alexander Dikovsky. Polarized non-projective dependency grammars. In Ph. de Groote, G. Morill, and Ch. Retoré, editors, *Proc. of the Fourth Intern. Conf. on Logical Aspects of Computational Linguistics*, Lecture Notes in Artificial Intelligence. vol. 2099, pages 139–157. Springer, 2001.
6. Alexander Dikovsky. Dependencies as categories. In G-J.M. Kruijff and D. Duchier, editors, *Proc. of Workshop “Recent Advances in Dependency Grammars”*. In conjunction with *COLING 2004*, pages 90–97, Geneva, Switzerland, August, 28th 2004.
7. Alexander Dikovsky and Larissa Modina. Dependencies on the other side of the Curtain. *Traitement Automatique des Langues (TAL)*, 41(1):79–111, 2000.
8. Denis Duchier and Ralph Debusmann. Topological dependency trees: A constraint based account of linear precedence. In *Proc. of the 39th Intern. Conf. ACL’2001*, pages 180–187. ACL & Morgan Kaufman, 2001.
9. Aravind K. Joshi, Vijay K. Shanker, and David J. Weir. The convergence of mildly context-sensitive grammar formalisms. In P. Sells, S. Shieber, and T. Wasow, editors, *Foundational issues in natural language processing*, pages 31–81, Cambridge, MA, 1991. MIT Press.

Towards a Computational Treatment of Binding Theory

Roberto Bonato

Dipartimento di Informatica, Università degli Studi di Verona,
Strada Le Grazie, 15 - 37134 Verona, Italy
`bonato@sci.univr.it`

Abstract. We present two approaches to the task of computing in an inductive compositional way semantic representations of the meaning of a sentence that take into account the principles of Binding Theory. The two algorithms reflect two different interpretations that have been proposed for the principles of Binding Theory as first formulated by Noam Chomsky. We present the two algorithms as additional machinery to enrich well-known bottom-up procedures to compute the logical form of a sentence in a (non intensional) Montagovian style.

1 Introduction to Binding Theory

Binding Theory (henceforth BT for short) aims at describing referential and mutual distributional properties among Determiner Phrases (or DPs) in a given sentence. DPs include a wide range of linguistic expressions, but we restrict our attention to three kinds of DP: anaphors (or reflexives, such as *himself*, *herself*, *themselves*), pronouns (such as *he*, *him*, *them*), and referential expressions (or full-DP, such as *John*, *the maid*, *the woman that married Bill*). A referential expression bears an independent semantic content, while this is not the case for an anaphor or a pronoun: their semantic interpretation relies on other entities within or outside the sentence they belong to, with which they are said to be *coreferential*. In order to compute the correct interpretation for a given sentence, a speaker must be able to recover that semantic content for pronouns and anaphors, and BT provides some general principles to rule this process commonly carried out very quickly by a human speaker.

Binding Theory was first formulated as a module of Government and Binding Theory by Chomsky in [1] (see also [2] for an updated account on BT). It relies on a syntactic device called *coindexing*, and on a structural relation between nodes of the parse tree (or *phrase-marker*) of a given sentence called *c-command*. *Indexing* is the practice of denoting the interpretation relations among the DPs of a sentence by means of indices, i.e. integers attached to DPs in such a way that elements bearing the same index are taken to denote the same entity, while different indices correspond to different denotations.

- (1) a. John₁ thinks that Mary likes him₁.
- b. John₂ thinks that Mary likes him₁.

In (1-a) *him* is *coindexed* with *John*, and thus it's supposed to have the same denotation, while in (1-b) different indexes imply that their denotations must be different.

In the parse tree for a given sentence, we say that a node n_1 *c-commands* another node n_2 when n_1 does not dominate n_2 and the first node dominating n_1 also dominates n_2 . More intuitively, we could say that either n_2 is a sister of n_1 , or it's the descendant of a sister of n_1 .



In (2) B c-commands C, D and E; C c-commands B; D c-commands E; E c-commands D. No other c-command relation exists.

On the basis of the notions of coindexing and c-command, two DPs are said to be (syntactically) *bound* when one c-commands the other and they are coindexed. BT formulates three principles which rule situations in which a given DP can, must or must not be bound within a sentence.

Principle A. A reflexive pronoun must be bound within its local domain.

- (3) Principle A examples:
- a. John_i hates himself_i.
 - b. John_i thinks that Bill_j hates himself_j.
 - c. *John_i thinks that Bill_j hates himself_i.

Principle B: A non-reflexive pronoun must not be bound within its local domain.

- (4) Principle B examples:
- a. *John_i hates him_i.
 - b. John_i hates him_j.
 - c. *John_i thinks that Bill_j hates him_j.
 - d. John_i thinks that Bill_j hates him_i.

Principle C: A referential expression must not be bound.

- (5) Principle C examples:
- a. *John_i hates John_i.
 - b. He_i hates John_j.
 - c. *He_i hates John_i.
 - d. *He_i thinks that Bill_j hates John_i.

The notion of *local domain* of a DP deserves further analysis but a formal definition goes beyond the scope of the present work. We rely on the very rough approximation that considers the local domain of a pronoun as the *smallest*

clause it belongs to. For example, clauses introduced by complementizer *that* in the previous examples are the local domains for the DPs within them, instead of the whole sentence.

Principles A, B and C describe three conditions (two negative and one positive) that every sentence must fulfill in order to be well formed. Whether such well-formedness has to be considered of a syntactic or semantic nature is still the subject of active debate. Binding Theory lies at the very core of the debate on the interface between syntax and semantics.

From a computational point of view such a formulation of Binding Theory is quite unsatisfactory. Indeed, BT principles provide a procedure to *verify* that a given indexing for a sentence is BT-compliant, but they are *not constructive*: no effective procedure to associate correct indexing to DPs in a sentence is provided. We could think of an algorithm which generates all possible indexing for a sentence and then filters them through a BT-principles verification module: obviously enough, not an efficient approach. We would like to incorporate such principles into a computational semantic framework that compositionally computes semantic representations for a sentence which are expressive enough to take into account possible, impossible or optional indexing for a sentence. As it will be made clear in sections 3 and 4, different interpretations that have been proposed to principles A, B and C yield different computational solutions to this problem. In section 3 we outline the so-called bound-variable interpretation of Binding Theory, largely due to the work of Tanya Reinhart in [3], while in 4 we present a more classical, coreferential interpretation of BT. For each of them we sketch a possible computational treatment.

2 Basic Apparatus

Both algorithms presented share a common basic structure: (a) the input are generative parse trees; (b) additional information on the structural relations between DPs is inductively collected during a preprocessing phase; (c) the semantic interpretation procedure possibly re-arranges the structure of the parse trees by means of Quantifier Raising according to the information collected during the previous phase; (d) suitable semantic representations are generated by means of classical computational semantics methods enriched with some additional machinery to deal with relations computed during phase (b). The algorithms share a basic formal apparatus, the main differences lying in different interpretation procedures (points (c) and (d)).

We assume that the input parse trees are binary branching. Although most of the algorithmic apparatus we present is compatible with other types of parse trees, we stick to this assumption stemmed from the most recent developments of generative syntax (like in [4] for example) which is computationally convenient and still accounts for a large range of phenomena. On the basis of evidence which seemed to jeopardize the validity of the notion of c-command, radically different "c-command"-less approaches have been proposed (see for example [5]). Nevertheless, recent developments like Pesetsky's "cascades" (see [6]) strongly

advocate for a binary tree approach and confirm the central role and predictive power of the notion of c-command for Binding Theory.

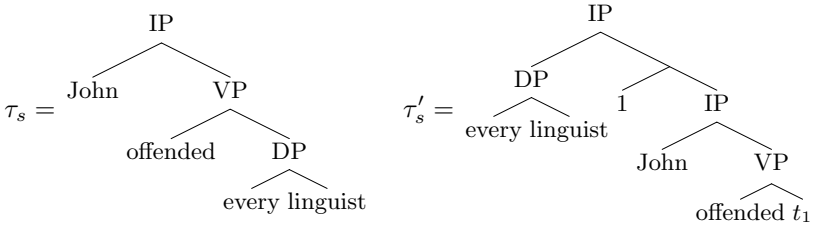
For a given sentence s , let τ_s be a generative parse tree for s and $\mathcal{N} = \{n_1, n_2, \dots, n_q\}$ be the set of nodes in τ_s , each corresponding to a different syntactic constituent¹. Let $\mathcal{D} \subseteq \mathcal{N}$ the set of nodes in τ_s which correspond to Determiner Phrases (DPs), and the three disjoint sets \mathcal{A} , \mathcal{P} and \mathcal{R} be subsets of \mathcal{D} whose members are the nodes corresponding to anaphors, pronouns and full-NP (or r-expressions) respectively.

Definition 1. A binary predicate *local* is defined on $\mathcal{D} \times \mathcal{D}$ such that $\text{local}(n_i, n_j) = \text{TRUE}$ iff nodes n_i and n_j correspond to two DP constituents which belong to the same local domain².

Definition 2. A binary predicate *agr* is defined in $\mathcal{D} \times \mathcal{D}$, such that $\text{agr}(n_i, n_j) = \text{TRUE}$ iff the agreement features of DP constituents corresponding to n_i and n_j are mutually compatible.

Definition 3. A function $\text{dps}(n)$ is defined from \mathcal{N} to $P(\mathcal{N})$, which returns the set of nodes which correspond to DP constituents within the constituent rooted in n .

Quantifier Raising is an operation which restructures a parse tree, usually to correct a type mismatch between the types of the constituents involved. Basically, for a given parse tree τ_s and one of its DP constituents d , d is moved to a "higher" position (or landing-site) in the tree, leaving a trace which will be interpreted as a bound variable.



In τ_s the quantificational DP *every linguist* is traditionally given type $\langle\langle e, t \rangle, t\rangle$: it's a function from predicates (functions from entities to truth values) to truth values. On the other hand, verb *offended* has type $\langle e, \langle e, t \rangle \rangle$, that is, it's a function from entities to functions from entities to truth values. Therefore, in τ_s no

¹ When this doesn't generate confusion, we'll often blur the difference between a node in the parse tree and the corresponding syntactic constituent, e.g. we'll say that node n is an anaphor.

² Deciding whether two NPs belong to the same local domain is far from trivial and it represents an important part of BT, but in the present work we won't go into the details about how that can be done algorithmically.

functional application is possible between *offended* and *every linguist*. The mismatch is amended in τ'_s : constituent DP is raised right above the first IP node, leaving a trace t_1 which will be interpreted as a variable. Node 1 in τ'_s marks the presence of an operation of lambda abstraction for the variable associated to t_1 on the subterm rooted in IP.

3 First Approach

3.1 Reinhart's Interpretation of BT

The first algorithm we are going to detail basically adheres to the interpretation of BT first given by Tanya Reinhart in [3]. According to this interpretation, Binding Theory does not really constrain *coreference* relations between distinct linguistic entities in a sentence, i.e. whether two distinct DPs can, must or must not *refer to* (or *denote*) the same entity in the real world. Instead, Binding Theory only rules the conditions under which is it possible, impossible or mandatory to give a bound-variable semantic reading between two entities. Two semantically bound entities receive the same semantic interpretation, but this purely grammatical "coreferential" effect must be kept conceptually distinct from other *accidental* coreference phenomena which will be dealt by pragmatics or discourse theory modules. In a sentence like *John likes himself* what principle A of Binding Theory actually tells us is that *himself* and *John* must be given a bound-variable interpretation. That is, the actual semantic representation computed for the sentence is $(\lambda x.\text{LIKE}(x, x))(\text{JOHN}) = \text{LIKE}(\text{JOHN}, \text{JOHN})$. So coreference between *himself* and *John* is more a byproduct of a purely grammatical property as bound variable reading, than a principled correspondence between linguistic entities and real world referents.

According to this interpretation, principle B doesn't forbid at all sameness of reference between a non-reflexive pronoun and another DP. All that it states is that a pronoun cannot be given a bound variable reading with another DP within its local domain. Of course, that could very well happen outside. With examples:

- (6) a. John likes him.
- b. John thinks that he likes Ann.

In (6-a) principle B only forbids a semantic representation where *John* and *him* are semantically bound³. That is, $(\lambda x.\text{LIKE}(x, x))(\text{JOHN})$ cannot be a good semantic representation for (6-a). However, nothing prevents *John* and *him* from referring to the same individual in the real world. Indeed, we can think of many contexts in which this is the case, like in the well known (although not uncontroversial) example: "It is not true that nobody likes John. *John* likes him!". To put it differently, coreferential relations are a matter of discourse-theoretical

³ Classical generative semantics states that a DP α semantically binds a DP β iff β and the trace of α are semantically bound by the same variable binder.

issues, and not the topic of Binding Theory, which only rules a very specific, “internal” property as semantic binding. The optional character of principle B for pronouns outside their local domain is evident in example (6-b), where it licenses a bound variable reading between *John* and *he*. Thus, we *can* have a semantic representation of (6-b) as $(\lambda x. \text{THINK}(x, \text{LIKE}(x, \text{ANN}))) (\text{JOHN})$, in which *he* and *John* are semantically bound by the same variable binder.

Principle C basically disappears in this reformulation of BT. A referential expression, from a computational semantics point of view, is not a variable, and so it doesn’t make sense for it to forbid any bound variable reading with any other c-commanding entity in the sentence.

At first sight, Reinhart’s interpretation looks very attractive from a computational point of view. “Forbidden coreference” relations between two DPs (often referred to as *obviation* relations) basically disappear because they are converted into non-bound variable readings, which are provided by default by using fresh variables at each occurrence of a pronoun or anaphor. A sentence like *John likes him* will be naturally translated into $\text{LIKE}(\text{JOHN}, x)$ with no further constraints on the fact that usually such a sentence entails (in the vast majority of contexts at least, although not all), that *John* and *him* have different denotations. Additional computational machinery is only needed to account for optional or mandatory bound-variable readings when principle B or principle A apply, respectively.

However, this superficial “unambitious” view of Reinhart’s interpretation hides a major computationally challenging issue. As a matter of fact, a speaker who wants to convey the meaning *John likes himself* (where *himself* behaves like a bound variable later saturated by *John*) will never utter the sentence *John likes him* (where the pronoun *him* behaves like a free variable which can be mapped by the context into *John*). How does Reinhart’s interpretation of BT account for such evidence? Reinhart provides an elegant solution by means of “principle I”: if a given message can be conveyed by two minimally different logical forms of which one involves variable binding where the other has co-reference, then the variable binding structure is always the preferred one. Although conceptually neat, this principle is computationally ruinous. Indeed, it implies being able to decide whether two distinct logical forms (conveyed messages corresponding to two different sentences) are satisfied by the same sets of variable assignments, and such a computational problem is NP-complete. At the present stage of work we still don’t address this issue.

3.2 Algorithm’s Outline

The algorithm takes as input a generative parse tree τ_s associated to a given sentence s . It returns a set of semantic representations, each corresponding to a possible reading for the sentence according to the interpretation of BT principles given by Reinhart and detailed in the previous section.

First Step: Computing O and M Relations. We need to compute some structural relations existing between DPs in τ_s . We enrich the basic formal ma-

chinery traditionally used to compute the first order semantic representation of a parse tree (see [7] for further details) by means of two binary predicates:

Definition 4. A binary predicate O (for optional) is defined in $\mathcal{D} \times \mathcal{D}$, such that $O(n_i, n_j) = TRUE$ iff:

- $n_j \in \mathcal{P}$;
- n_i c-commands n_j ;
- $local(n_i, n_j) = FALSE$;
- $agr(n_i, n_j) = TRUE$.

Predicate O models the situation in which two distinct nodes in the tree corresponds to two DP constituents which, according to principle B, *can* be given a bound-variable reading.

Definition 5. A binary predicate M (for mandatory) is defined in $\mathcal{D} \times \mathcal{D}$, such that $M(n_i, n_j) = TRUE$ iff:

- $n_j \in \mathcal{A}$;
- n_i c-commands n_j ;
- $local(n_i, n_j) = TRUE$;
- $agr(n_i, n_j) = TRUE$.

Predicate M characterizes the situation in which two distinct nodes in the parse tree correspond to two DP constituents which, according to principle A, *must* be given a bound variable interpretation.

Each node n in the parse tree τ_s comes with two sets associated, $\mathcal{O}_n \subseteq \mathcal{D} \times \mathcal{D}$ and $\mathcal{M}_n \subseteq \mathcal{D} \times \mathcal{D}$, which contain the new couples (n_i, n_j) for which $O(n_i, n_j)$ and $M(n_i, n_j)$ respectively hold in the subtree rooted in n . For each node n , sets \mathcal{O}_n and \mathcal{M}_n can be inductively computed as follows:

- if n is a leaf, $\mathcal{O}_n = \mathcal{M}_n = \emptyset$;
- let n_1 and n_2 be the two daughters of n , then:
 1. If n_1 corresponds to a DP constituent, for each $n_i \in dps(n_2)$:
 - if $n_i \in \mathcal{A}$ and $local(n_1, n_i) = TRUE$ and $agr(n_1, n_i) = TRUE$, then add (n_1, n_i) to \mathcal{M}_n ;
 - if $n_i \in \mathcal{P}$ and $local(n_1, n_i) = FALSE$ and $agr(n_1, n_i) = TRUE$, then add (n_1, n_i) to \mathcal{O}_n .
 2. If n_2 corresponds to an NP constituent, for each $n_j \in dps(n_1)$:
 - if $n_j \in \mathcal{A}$ and $local(n_2, n_j) = TRUE$ and $agr(n_2, n_j) = TRUE$, then add (n_2, n_j) to \mathcal{M}_n ;
 - if $n_j \in \mathcal{P}$ and $local(n_2, n_j) = FALSE$ and $agr(n_2, n_j) = TRUE$, then add (n_2, n_j) to \mathcal{O}_n .

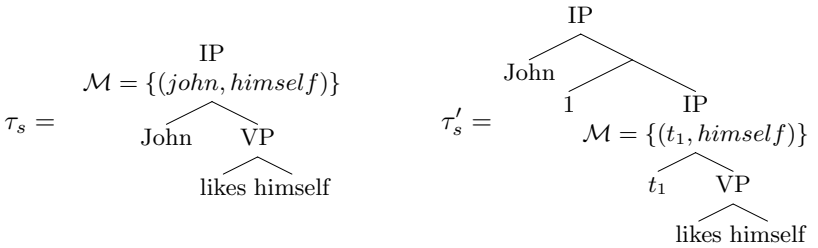
As it can be easily proved, for each n in τ_s , \mathcal{O}_n and \mathcal{M}_n contain nodes for which relations O and M hold, respectively. Agreement and locality conditions are verified by definition, while c-command relation is verified by construction: both point 1 and 2 take into account only (and all) couples (n_i, n_j) where n_j is either a sister or a descendant of a sister of n_i , which is one of the way we can characterize c-command relation, as seen in section 2.

Second Step: Rearranging the Parse Tree. Information on relations M and O that hold between DP nodes collected during the first step is now used to induce some transformations on the parse tree. Quantifier Raising is used in a new way in addition to the classical case of type mismatch resolution.

The intuitive idea behind this step is that, for reasons that will be made clear in the next step, when we come to the semantic interpretation, we want relations O and M to hold between entities which are semantically mapped into (free or bound) variables. For each couple (n_i, n_j) which belong to O or M , we have by construction that $n_j \in \mathcal{P}$ or $n_j \in \mathcal{A}$, that is n_j is either a pronoun or an anaphor, traditionally interpreted as variables in classical computational semantics. In general this is not the case for n_i which may very well belong to \mathcal{P} , that is, it can be a referential expression. How can we restructure the tree in such a way that the semantic interpretation of the node n_i will be a variable and yet we don't change the overall semantic representation for the sentence? The answer lies in QR, which rearranges a tree in a way which is convenient to the purposes of step 3, leaving a variable as the semantic interpretation associated to node n_i , yet without affecting the final semantics of the sentence, as it is shown by the fundamental equivalence of lambda calculus: $\alpha(\beta) \equiv (\lambda x. \alpha(x))(\beta)$.

In this case we apply QR to the constituent rooted in n_i , "raising" it up to the closest IP node in the tree. The result is a new parse tree τ'_s which will be the input to the semantic interpretation procedure detailed in step 3. It is important to note that sets \mathcal{O}_n and \mathcal{M}_n associated to a node n don't "move" along with the raised constituents and still model the actual BT relations computed in step 2, although the semantic denotation associated to the constituents involved has (momentarily) changed.

(7) $s = \text{John likes himself.}$



In licensing Quantifier Raising for type e DPs and not just quantificational DPs, we adhere both to theoretical and computational convenience criterions. As seen in section 2, QR was first introduced to amend type mismatches induced by quantificational DPs in object position. In [3] Reinhart provides convincing evidence that also type e DP can undergo QR, basically to solve the sloppy-strict identity puzzle for elliptic construals. Furthermore, it can be proved that truth conditions of a sentence in which a type e DP has been quantifier-raised are equivalent to those in which the DP is left *in situ*. As pointed out in [7], in this situation it is difficult to see what could forbid the possibility that DPs of type e undergo the same movements of DPs of type $\langle\langle e, t \rangle, t \rangle$. We thus choose to license

such kind of movements for reasons of computational convenience that will be made clear in the following steps of the algorithm.

Third Step: Computing Semantics. Traditional computational semantics (as presented, for example, in [7]) computes a (unique) semantic representation for a node n in an inductive way: if n is a leaf its semantics is directly provided by the lexicon for the lexical entry associated; if n is an internal node, its semantics is computed through some basic operations (functional application, boolean conjunction, lambda abstraction) out of the semantics of its children nodes. We need to enrich such a framework so as to compute *multiple* semantic representations associated to a given node, each of them corresponding to a different reading for the anaphora and pronouns involved⁴. In order to do so, for each node n of the parse tree τ'_s now associated to a given sentence s , instead of single a lambda term we compute a set \mathcal{S}_n of lambda terms, each corresponding to a different semantic representation for the constituent associated to node n : the information collected in sets \mathcal{O}_n and \mathcal{M}_n for each node enrich the semantic interpretation procedure in a way that takes into account Reinhart's interpretation of binding principles. Let's see how this can be done in an inductive algorithmic way.

Let τ'_s be the parse tree associated to a given sentence s after having been restructured by step 2, in which we suppose that each node n is decorated with additional information provided by sets \mathcal{O}_n and \mathcal{M}_n associated, as described in section 3.2. We want to compute the set \mathcal{S}_n of semantic representations associated to node n . This can be done inductively as follows:

- if n is a leaf, $\mathcal{S}_n = \{S\}$, where S is the semantic representation of the corresponding lexical entry as provided by the lexicon;
- if n is an internal node, let n_1 and n_2 be its children, and \mathcal{S}_1 and \mathcal{S}_2 be the sets of semantic representations associated to each of them. If we use the symbol \circ to indicate "semantic composition" (see [7] for further details on the basic mechanisms of semantic composition), for each $S_i \in \mathcal{S}_1$ and $S_j \in \mathcal{S}_2$, we initially define $\mathcal{S}_n = \bigcup_{i,j} \{S_i \circ S_j\}$.

For each $S(x_1, \dots, x_q) \in \mathcal{S}_n$, let \mathcal{O}_n and \mathcal{M}_n be the sets associated to node n defined as in section 3.2. The interpretation procedure goes on as follows:

1. for each $(n_i, n_j) \in \mathcal{M}_n$, $\llbracket n_i \rrbracket = x_i$ and $\llbracket n_j \rrbracket = x_j$ by construction of step 2, with x_i and x_j fresh distinct variables. Add $S(x_1, \dots, x_q)[x_i/x_j]$ to the set $\mathcal{S} - \{S(x_1, \dots, x_q)\}$;
2. for each $(n_k, n_l) \in \mathcal{O}_n$, $\llbracket n_k \rrbracket = x_k$ and $\llbracket n_l \rrbracket = x_l$ by construction of step 2, with x_k and x_l fresh distinct variables. Add $S(x_1, \dots, x_q)[x_k/x_l]$ to the set \mathcal{S} .

Point 1 reflects the situation in which node n_j corresponds to an anaphor for which n_i is a possible antecedent. In that case the free variable reading $S(x_1, \dots, x_q)$ *must be replaced* by one in which the two corresponding variables

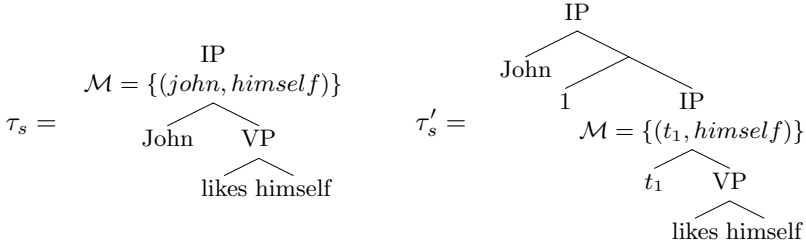
⁴ An underspecified representation along the lines of [8], for example, would be highly recommended to deal with the explosion of semantic readings, and it will be the next step in our work.

are equated (which is implemented as a substitution in $S(x_1, \dots, x_q)[x_i/x_j]$) and will become bound by the later application of lambda abstraction over the outermost one.

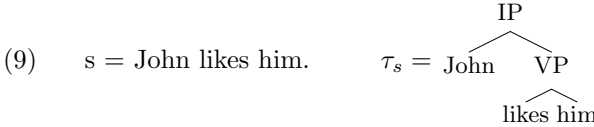
Point 2 deals with all the situations in which node n_k has been identified as a possible antecedent of a pronoun corresponding to node n_l . In this case we must provide both readings: one in which the two corresponding entities are not bound ($S(x_1, \dots, x_q)$, which is generated by default by the algorithm which associates unique distinct variables to every pronoun and reflexive), and a second one ($S(x_1, \dots, x_q)[x_k/x_q]$) in which the two entities equated and (later) bound by a lambda abstraction over the outermost one.

3.3 Some Examples

(8) s = John likes himself.

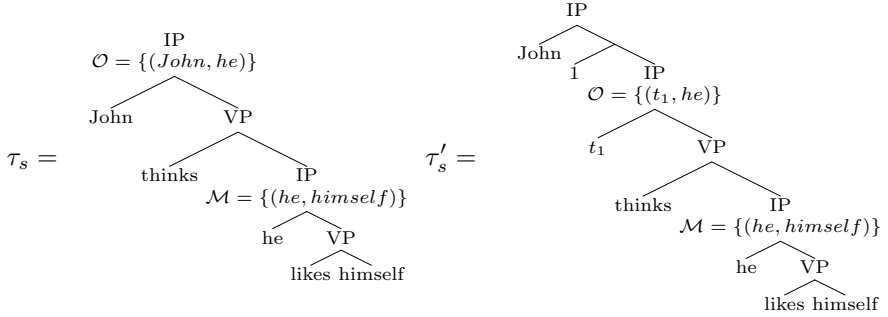


In interpreting node IP of τ_s , we must interpret also the associated set $\mathcal{M} = \{(john, himself)\}$. Since the denotation of *John* is not a variable, step 2 from the previous section triggers the quantifier raising for *John* in τ_s , whose result is τ'_s , over which the interpretation process goes on. If we set $\llbracket t_1 \rrbracket = x_1$ and $\llbracket himself \rrbracket = x_2$, with x_1, x_2 fresh new variables, the semantic representation computed for the "smaller" IP subtree in τ'_s is $\mathcal{S} = \{\text{LIKE}(x_1, x_2)\}$, with $\mathcal{M} = \{(t_1, himself)\}$. According to point 1, the resulting semantics is $\mathcal{S} \cup \{\text{LIKE}(x_1, x_2)[x_1/x_2]\} - \{\text{LIKE}(x_1, x_2)\} = \{\text{LIKE}(x_1, x_1)\}$. Next step (lambda abstraction over x_1) yields $\mathcal{S} = \{\lambda x_1. \text{LIKE}(x_1, x_1)\}$, later saturated by functional application of $\llbracket John \rrbracket = \text{JOHN}$ in $\mathcal{S} = \{\text{LIKE}(\text{JOHN}, \text{JOHN})\}$.



Nothing special happens: sets \mathcal{O} and \mathcal{M} are empty for any node, semantics for the IP node is simply $\mathcal{S} = \{\text{LIKE}(\text{JOHN}, x_1)\}$ in compliance with Reinhart's treatment of this case.

(10) $s = \text{John thinks that he likes himself.}$



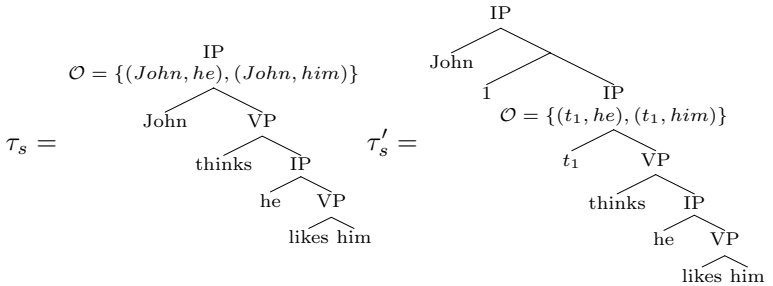
Since $\llbracket John \rrbracket$ is not a variable, according to step 2 in the previous section, this triggers the quantifier raising for *John* in τ_s , resulting in tree τ'_s . If we set $\llbracket he \rrbracket = x_3$ and $\llbracket himself \rrbracket = x_2$, the interpretation process computes $\mathcal{S}' = \{\text{LIKE}(x_3, x_2)\}$ with $\mathcal{M} = \{(he, himself)\}$ for the smallest IP constituent in τ'_s . By application of rule 1 we then have $\mathcal{S}' = \{\text{LIKE}(x_2, x_2)\}$. If we set $\llbracket t_1 \rrbracket = x_1$, when the interpretation process reaches the middle IP node, we have $\mathcal{S}'' = \{\text{THINK}(x_1, \text{LIKE}(x_2, x_2))\}$ with $\mathcal{O} = \{(t_1, he)\}$. Application of point 2 from the previous section results in the following semantics for the node:

$$\begin{aligned} \mathcal{S}'' \cup \{\text{THINK}(x_1, \text{LIKE}(x_2, x_2))[x_1/x_2]\} = \\ = \{\text{THINK}(x_1, \text{LIKE}(x_2, x_2)), \text{THINK}(x_1, \text{LIKE}(x_1, x_1))\}. \end{aligned}$$

Successive steps yield the following semantic representations:

$$\begin{aligned} \mathcal{S}'' &= \{\lambda x_1. \text{THINK}(x_1, \text{LIKE}(x_2, x_2)), \lambda x_1. \text{THINK}(x_1, \text{LIKE}(x_1, x_1))\} \quad (\lambda\text{-abstr.}) \\ &= \{\text{THINK}(\text{JOHN}, \text{LIKE}(x_1, x_1)), \text{THINK}(\text{JOHN}, \text{LIKE}(\text{JOHN}, \text{JOHN}))\} \quad (\text{f. appl.}) \end{aligned}$$

(11) $s = \text{John thinks that he likes him.}$



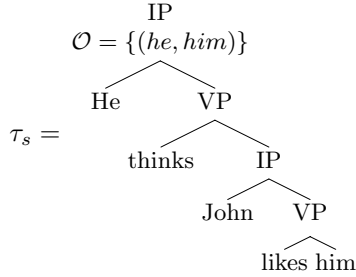
Once again the fact that $\llbracket John \rrbracket$ is not a variable triggers the quantifier raising of *John* in τ_s , the result being τ'_s . It is easy to verify that the semantic computation for the middle IP node yields $\mathcal{S} = \{\text{THINK}(x_1, \text{LIKE}(x_2, x_3))\}$ with $\mathcal{O} = \{(t_1, he), (t_1, him)\}$. According to point 2 from the previous section the full semantics for the node will be:

$$\begin{aligned}\mathcal{S}' &= \mathcal{S} \cup \{\text{THINK}(x_1, \text{LIKE}(x_2, x_3))[x_2/x_1]\} \cup \{\text{THINK}(x_1, \text{LIKE}(x_2, x_3))[x_3/x_1]\} \\ &= \{\text{THINK}(x_1, \text{LIKE}(x_2, x_3)), \text{THINK}(x_1, \text{LIKE}(x_1, x_3)), \text{THINK}(x_1, \text{LIKE}(x_2, x_1))\}\end{aligned}$$

Successive interpretation steps yield:

$$\begin{aligned}\mathcal{S}' &= \{\lambda x_1. \text{THINK}(x_1, \text{LIKE}(x_2, x_3)), \lambda x_1. \text{THINK}(x_1, \text{LIKE}(x_1, x_3)), \\ &\quad \lambda x_1. \text{THINK}(x_1, \text{LIKE}(x_2, x_1))\} = \\ &= \{\text{THINK}(\text{JOHN}, \text{LIKE}(x_2, x_3)), \text{THINK}(\text{JOHN}, \text{LIKE}(\text{JOHN}, x_3)), \\ &\quad \text{THINK}(\text{JOHN}, \text{LIKE}(x_2, \text{JOHN}))\}.\end{aligned}$$

(12) s = he thinks that John likes him.



In this case, being $\llbracket he \rrbracket$ a variable, no "mismatch" occurs when the interpretation process reaches the highest IP, where $\mathcal{S} = \{\text{THINK}(x_1, (\text{LIKE}(\text{JOHN}, x_2)))\}$ with $\mathcal{O} = \{(he, him)\}$. So, according to the interpretation rules for \mathcal{O} , we have that the complete semantics for the sentence is the set:

$$\begin{aligned}\mathcal{S}' &= \mathcal{S} \cup \{\text{THINK}(x_1, (\text{LIKE}(\text{JOHN}, x_2)))[x_1/x_2]\} \\ &= \{\text{THINK}(x_1, (\text{LIKE}(\text{JOHN}, x_2))), \text{THINK}(x_1, (\text{LIKE}(\text{JOHN}, x_1)))\}\end{aligned}$$

4 Second Approach

4.1 The Coreferential Interpretation of BT

The second approach we present sticks to a somewhat more classical interpretation of Binding Theory, that is, a *coreferential* one. Anaphoric elements such as pronouns and reflexives are linguistic items that don't have intrinsic denotation or reference. Their *linguistic antecedent* is the linguistic element from which the anaphoric element obtains its reference, and thus with which it is said to be *coreferential*. Besides, this approach states that in order for a sentence like *John likes John* to be semantically correct, the first occurrence of *John* must necessarily refer to a distinct individual than the second one, that happens to have the same name. This coreferential interpretation of Binding Theory advocates a tight correspondence between linguistic and real world entities.

In the perspective of this interpretation, principle A states that a reflexive *must* take its reference within its local domain. That is, it establishes a *functional*

dependence between the reference of the reflexive and exactly one of the DPs which belong to its local domain. This functional dependence cannot but be some kind of identity function. Principle B states that a pronoun *must not* take its reference within its local domain, but it doesn't say anything about what must its actual reference. We thus have a condition under which the interpretation procedure must fail, that is, denotations of the two entities *must be different*. Principle C is similar to principle B, here again we can imagine to translate it in its computational counterpart as an additional condition that can make the interpretation process fail.

4.2 Second Algorithm's Outline

Semantic interpretation according to this alternative interpretation of BT differs from the previous one under some important respects. This entails substantive changes in the formal apparatus needed to compute the corresponding semantic representations.

First Step: Computing F and M Relations. Principle B takes an eminently *obviative* character. Whereas in the previous approach it made *binding possible* between a pronoun and a c-commanding entity *outside* its local domain, it now makes *coreferentiality forbidden* between a pronoun and any other c-commanding DP *within* its local domain. Outside this domain, it can corefer with any other entity, either c-commanding or not. Principle C (previously implicit in the semantic interpretation process) takes now an obviative character too: a full NP must not be coreferential with any other c-commanding entity.

We need a new relation between entities and a modified interpretation mechanism in order to express such obviative conditions, such that semantic interpretation fails if those conditions are violated. This will be done by replacing binary predicate O and set \mathcal{O} of the previous approach with binary predicate F and set $\mathcal{F} \subseteq \mathcal{D} \times \mathcal{D}$ associated to each node of a parse tree.

Definition 6. A binary predicate F (for "forbidden") is defined in $\mathcal{D} \times \mathcal{D}$, such that $F(n_i, n_j) = TRUE$ iff:

- $n_j \in \mathcal{P}$;
- n_i c-commands n_j ;
- $local(n_i, n_j) = TRUE$;
- $agr(n_i, n_j) = TRUE$.

Predicate F holds between two nodes in the parse tree iff they correspond to a generic DP c-commanding a non reflexive pronoun within its local domain and which due to principle B cannot be coreferential.

Each node n in the parse tree τ'_s comes with two sets associated, $\mathcal{M}_n \subseteq \mathcal{D} \times \mathcal{D}$ and $\mathcal{F}_n \subseteq \mathcal{D} \times \mathcal{D}$. They are inductively computed as follows:

- if n is a leaf, $\mathcal{M}_n = \mathcal{F}_n = \emptyset$;
- if n is an internal node, let n_1 and n_2 be the two daughters of n , then:

1. If n_1 corresponds to an NP constituent, for each $n_i \in nps(n_2)$:
 - if $n_i \in \mathcal{A}$ and $local(n_1, n_i) = TRUE$, then add (n_1, n_i) to \mathcal{M}_n ;
 - if $n_i \in \mathcal{P}$ and $local(n_1, n_i) = TRUE$, then add (n_1, n_i) to \mathcal{F}_n .
2. If n_2 corresponds to an NP constituent, for each $n_j \in nps(n_1)$:
 - if $n_j \in \mathcal{A}$ and $local(n_2, n_j) = TRUE$, then add (n_2, n_j) to \mathcal{M}_n ;
 - if $n_j \in \mathcal{P}$ and $local(n_2, n_j) = FALSE$, then add (n_2, n_j) to \mathcal{F}_n .

It can be easily proved that for each n in τ_s , \mathcal{F}_n and \mathcal{M}_n contain nodes for which previous section relations F and M hold, respectively. Agreement and locality conditions are verified by definition, while c-command relation is verified by construction: both point 1 and 2 take into account only (and all) couples (n_i, n_j) where n_j is either a sister or a descendant of a sister of n_i , which is one of the possible characterizations for c-command relation, as seen in section 2.

Second Step: Rearranging the Parse Tree. This step is entirely analogous to third step of the previous algorithm presented. The fact that the semantic interpretation of n_i in a couple (n_i, n_j) belonging either to \mathcal{F} or to \mathcal{M} is not a variable triggers the QR of the constituent rooted in n_i .

Third Step: Computing Semantics. Semantic interpretation for a sentence s takes as its input the restructured parse tree τ'_s , where all $n_i \in \mathcal{R}$ has been quantifier-raised and whose nodes are decorated with sets \mathcal{M} and \mathcal{F} . It outputs a set of semantic representations \mathcal{S} inductively computed in the following way for a generic subtree rooted at node n :

- if n is a leaf, then $\mathcal{S} = \{S\}$, with S the lambda term which describes the semantics for the lexical entry as given in the lexicon.
- if n is an internal node, let n_1 and n_2 be its children, and \mathcal{S}_1 and \mathcal{S}_2 be the sets of semantic representations associated to each of them. Then for each $S_i \in \mathcal{S}_1$ and $S_j \in \mathcal{S}_2$, we initially define $\mathcal{S} = \bigcup_{i,j} \{S_i \circ S_j\}$.

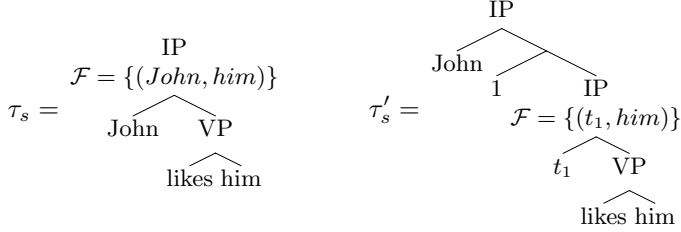
For each $S(x_1, \dots, x_q) \in \mathcal{S}$, let \mathcal{M}_n and \mathcal{F}_n be the sets associated to node n , then for each $(n_i, n_j) \in \mathcal{M}_n \cup \mathcal{F}_n$ we have by construction that $\llbracket n_i \rrbracket = x_i$ and $\llbracket n_j \rrbracket = x_j$, with x_i, x_j unique distinct variables. Then:

1. for each $(n_i, n_j) \in \mathcal{M}_n$, add $S(x_1, \dots, x_q)[x_i/x_j]$ to the set $\mathcal{S} - \{S(x_1, \dots, x_q)\}$;
2. for each $(n_k, n_l) \in \mathcal{F}_n$, add $S(x_1, \dots, x_q) \wedge (x_k \neq x_l)$ to the set \mathcal{S} .

Point 1 is the same as the corresponding point in the previous approach. In order to be well defined, point 2 makes an implicit assumption, that is lambda term S must be of type t (truth-value), otherwise the conjunction wouldn't make sense. We can easily prove that by construction this is always the case. We can also prove by construction that if $(n_i, n_j) \in \mathcal{F}$ or $(n_i, n_j) \in \mathcal{M}$, then n_i and n_j are either reflexives, pronouns, or traces. In any case, linguistic objects whose semantic representations are variables. The expression $\llbracket n_k \rrbracket \neq \llbracket n_l \rrbracket$ will always translate into a relation between variables in semantics.

4.3 Some Examples

(13) $s = \text{John likes him.}$



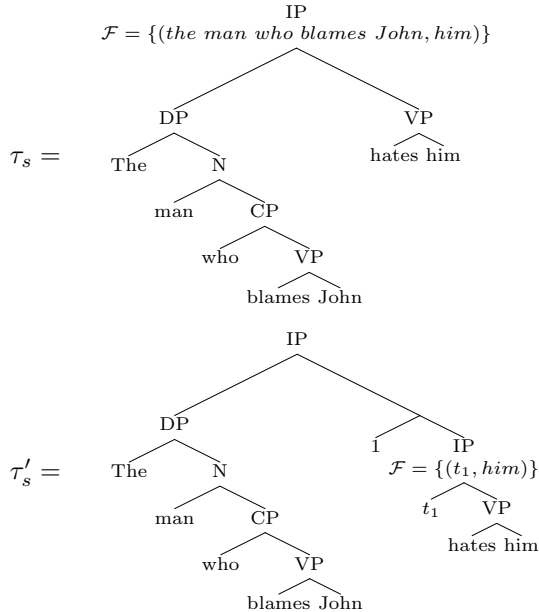
As usual, since the denotation of *John* is not a variable, parse tree τ_s is rearranged into τ'_s . When the semantic interpretation process reaches the lower IP node, it computes $\mathcal{S} = \{\text{LIKE}(x_1, x_2)\}$ with $\mathcal{F} = \{(t_1, him)\}$. According to point 2 of the previous section, this will reduce to

$$\begin{aligned} \mathcal{S}' &= \{\text{LIKE}(x_1, x_2)\} - \{\text{LIKE}(x_1, x_2)\} \cup \{(\text{LIKE}(x_1, x_2) \wedge (\llbracket t_1 \rrbracket \neq \llbracket him \rrbracket))\} = \\ &= \{\text{LIKE}(x_1, x_2) \wedge (x_1 \neq x_2)\} \end{aligned}$$

By successive semantic operations we get:

$$\begin{aligned} \mathcal{S} &= \{\lambda x_1. [\text{LIKE}(x_1, x_2) \wedge (x_1 \neq x_2)]\} \text{ (}\lambda\text{-abstr. over } x_1\text{)} \\ &= \{\text{LIKE}(\text{JOHN}, x_2) \wedge (\text{JOHN} \neq x_2)\} \text{ (functional application of } \llbracket John \rrbracket\text{)} \end{aligned}$$

(14) $\text{The man who blames John hates him.}$



For the smaller IP subtree the situation is the same as in the previous example and so we have $\mathcal{S}_{IP} = \{\text{HATE}(x_1, x_2) \wedge (x_1 \neq x_2)\}$ which becomes $\{\lambda x_1. [\text{HATE}(x_1, x_2) \wedge (x_1 \neq x_2)]\}$ after lambda abstraction over x_1 . We assume for DP node a standard Russellian semantics, that is $\mathcal{S}_{DP} = \{\lambda Q. [\exists x(\text{MAN}(x) \wedge \text{BLAME}(x, \text{JOHN}) \wedge \forall y((\text{MAN}(y) \wedge \text{HATE}(y, \text{JOHN})) \rightarrow (y = x)) \wedge Q(x))]\}$. Resulting semantics is thus

$$\begin{aligned}
\mathcal{S}_{DP} \circ \mathcal{S}_{IP} &= \lambda Q. [\exists x(\text{MAN}(x) \wedge \text{BLAME}(x, \text{JOHN}) \wedge \forall y((\text{MAN}(y) \wedge \text{HATE}(y, \text{JOHN})) \rightarrow \\
&\quad \rightarrow (y = x)) \wedge Q(x))] \circ \lambda x_1. [\text{HATE}(x_1, x_2) \wedge (x_1 \neq x_2)] = \\
&= \exists x(\text{MAN}(x) \wedge \text{BLAME}(x, \text{JOHN}) \wedge \forall y((\text{MAN}(y) \wedge \text{HATE}(y, \text{JOHN})) \rightarrow \\
&\quad \rightarrow (y = x)) \wedge (\lambda x_1. [\text{HATE}(x_1, x_2) \wedge (x_1 \neq x_2)](x)) = \\
&= \exists x(\text{MAN}(x) \wedge \text{BLAME}(x, \text{JOHN}) \wedge \forall y((\text{MAN}(y) \wedge \text{HATE}(y, \text{JOHN})) \rightarrow \\
&\quad \rightarrow (y = x)) \wedge \text{HATE}(x, x_2) \wedge (x \neq x_2))
\end{aligned}$$

5 Conclusion and Further Work

We have presented the general schemas for two algorithms which compute semantic representations for natural language sentences with intrasentential anaphora. The first is inspired by Reinhart's interpretation of Binding Theory, the second by the more classical coreferential approach. We have to delve deeper into the nature of the \neq relation which we used for the second algorithm, and to further explore the computational complexity issues of both approaches.

References

1. Chomsky, N.: Lectures on Government and Binding. Foris, Dordrecht (1981)
2. Buring, D.: Binding Theory. Cambridge Textbooks in Linguistics. Cambridge University Press, Cambridge (to appear)
3. Reinhart, T.: Anaphora and Semantic Interpretation. University of Chicago Press (1983)
4. Chomsky, N.: The minimalist program. MIT Press, Cambridge, MA (1995)
5. Reinhart, T., Reuland, E.: Reflexivity. *Linguistic Inquiry* **24** (1993) 657–720
6. Pesetsky, D.: Zero Syntax: Experiencers and Cascades. The MIT Press (1994)
7. Heim, I., Kratzer, A.: Semantics in Generative Grammar. Blackwell, Oxford (1998)
8. Egg, M., Niehren, J., Ruhrberg, P., Xu, F.: Constraints over lambda-structures in semantic underspecification. In Boitet, C., Whitelock, P., eds.: Proceedings of the Thirty-Sixth Annual Meeting of the ACL, San Francisco, California, Morgan Kaufmann Publishers (1998) 353–359

Translating Formal Software Specifications to Natural Language

A Grammar-Based Approach

David A. Burke and Kristofer Johannisson

Department of Computing Science, Chalmers University of Technology and Göteborg
University, SE-41296 Göteborg, Sweden
`krijo@cs.chalmers.se`

Abstract. We describe a system for automatically translating formal software specifications to natural language. The system produces natural language which is acceptable to a human reader, and it supports by-hand optimization by users who are not experts of our system. The translation system is implemented using the Grammatical Framework, a grammar formalism based on Martin-Löf's type theory. We show that this grammar-based approach scales well enough to handle a non-trivial case study: translating the Object Constraint Language specifications of the Java Card API into English.

1 Introduction

The goal of this work is to automatically translate formal software specifications into natural language. Our motivation is a wish to link formal specifications (as needed for formal methods) to informal ones (as found in software engineering practice). Our work is a part of the KeY project [1], which integrates formal software specification and verification into the industrial software engineering processes.

We have implemented a system, earlier described in [2], using the Grammatical Framework (GF), a grammar formalism based on Martin-Löf's type theory [3, 4]. In this paper we show that our grammar-based approach scales to such a degree that we can handle a non-trivial case study.

The case study consists of specifications for the Java Card API [5], written in the Object Constraint Language, which have been translated into English by our system. To improve the quality of the translation, we have extended our system with formatting and automatic generation of grammar modules for domain-specific vocabulary, which can then be modified without requiring GF expertise. We have also added various simple stylistic improvements inspired by techniques familiar from Natural Language Generation [6]. As far as possible, all these improvements are implemented in a declarative way in the GF grammars; some of them also require manipulation of syntax trees by a separate program, external to the grammars.

1.1 Paper Overview

We start with background on the GF formalism, formal specifications, the KeY project, and the Java Card API specification case study in Sect. 2. In Sect. 3 we then give a motivating example from the case study, showing an example formal specification, as well as English translations before and after our improvements.

Sect. 4 explains the overall architecture of the translation system, while Sect. 5 is concerned with grammar engineering: how to design the grammar based system to meet our goals.

Sect. 6 describes related work, mainly Natural Language Generation. Some figures on the size of the case study are given in Sect. 7, and we then conclude in Sect. 8.

2 Background

2.1 The Grammatical Framework

The GF Formalism. The Grammatical Framework (GF) is a formalism for defining grammars [3]. A GF grammar consists of one part which describes abstract syntax, and another part which describes concrete syntax. The abstract syntax part is formulated in a version of Martin-Löf's type theory [4], and can be seen as a description of how to construct abstract syntax trees. The concrete syntax then consists of *linearization* rules telling how to present these trees as expressions of a particular language. This is a distinguishing feature of GF as compared to many other grammar formalisms: grammars are written from the perspective of linearization rather than parsing. In fact, we can consider the GF formalism as a linearization (or generation) oriented typed functional language.

The concrete syntax is based on record types, strings and finite parameter types, enabling the representation of e.g. inflection tables and discontinuous constituents. A central notion in GF is *compositionality*: the linearization of a tree is always expressed in terms of the linearization of its subtrees, we have no access to the subtrees themselves in a linearization rule. This restriction is important for the implementation of GF.

By having multiple concrete syntaxes for the same abstract syntax we achieve *multilinguality*: we can present the same tree in several languages in parallel, and we can translate (within the language fragment described by the grammar) by parsing using one concrete syntax and linearizing with another. Compositionality imposes a restriction of structural similarity on the languages sharing the same abstract syntax, however, this restriction is to some degree countered by the expressiveness of the concrete syntax.

The GF System. The GF system [7] provides functionality such as parsing and linearization for grammars written in the GF formalism. The system also includes a syntax editor [8] in which the user can load a GF grammar and then edit the abstract syntax trees described by the grammar. The trees are all the time presented in the languages defined by the concrete syntaxes of the grammar. By

editing the abstract syntax tree and observing the results in a familiar language, a user can then interactively produce texts in foreign languages.

The GF Resource Grammar Library. An important part of the GF project is the resource grammar library [9], which provides an API of types and functions for common linguistic structures. There are resource grammars available for English, Finnish, French, German, Italian, Russian and Swedish, which to a large extent share the same interface.

Grammar Engineering. A typical GF application grammar describes a well-defined fragment of natural language for a restricted domain, e.g. in our case software specifications. The resource grammar library provides a division of labour: the author of an application grammar can be a domain expert, who does not need to be familiar with linguistic details. His or her task is to come up with an abstract syntax which models the domain, and to link the abstract syntax to concrete language by using the resource grammars. The linguistic expert is in turn responsible for the implementation of the resource grammars, where no knowledge of a particular domain is needed.

2.2 Formal Specifications and the KeY Project

The KeY Project. The KeY project [1] attempts to integrate formal software specification and verification into the industrial software engineering processes. The starting point is a commercial CASE (Computer Aided Software Engineering) tool, which is augmented by capabilities for formal specification and verification. The ultimate goal is to make the verification process transparent for the user with respect to the informal object-oriented model.

Formal and Informal Specifications. Formal methods require formal specifications, but in software engineering practice, informal specifications are commonly used. We cannot expect everyone who needs to deal with specifications—e.g. customers, managers, or software engineers—to master a formal notation (cf. [10] p. 131 “...most customers don’t understand formal specifications and are reluctant to accept it as a system contract”). This motivates the need for a systematic link between formal and informal specifications: to support authoring of specifications as well as synchronizing and maintaining formal and informal versions of specifications, and to present the specifications to different audiences using different levels of formality.

The Object Constraint Language. The Object Constraint Language (OCL) is a formal specification language used to specify precise requirements for object-oriented software systems [11]. It is a sub-standard of the Unified Modelling Language (UML) [12]. An OCL specification is always given in the context of some particular UML model,¹ and it consists of a boolean expression which

¹ In this paper, a *UML model* is simply a class diagram, containing classes, attributes, methods and associations. Side-effect free methods are called *queries*.

is used as an invariant of a class, or as a pre-condition or post-condition of a method. The attributes, queries and associations from the UML model, as well as a library of predefined types (e.g. integers, strings and collections) are available for constructing OCL expressions. For example, given a class `OwnerPIN`, with attributes `tryCounter` and `maxTries`, we can specify the requirement “the try counter is at most the maximum number of tries” using the OCL:

```
context OwnerPIN
inv: self.tryCounter <= self.maxTries
```

As we see in this example, each OCL expression is given in the context of a particular class, and the expression `self` refers to an instance of that class.

2.3 The Java Card API Specification

Java Card technology [5] allows software developers to write Java programs that run on smart cards and other devices with very limited memory and processing capabilities. The Java Card API (Application Programming Interface) is a set of library classes used in Java Card programs. It is a subset of the standard Java API and is specifically designed for smart card programming.

Due to the size and nature of the applications that use Java Card, formal methods could be useful in verifying the correctness of these programs. With this in mind, OCL constraints have been defined for the Java Card API in [13] (based on JML specifications in [14]). These OCL specifications provided the basis for a case study using our translation tool. The specifications of 37 Java classes were fully translated, and examples from this are used throughout this paper. Details on the case study are available in [15] and on the web [16].

3 Motivating Example

In this section we will take an OCL specification from the Java Card API case study (Fig. 1) and show how it gets translated into natural language. For comparison, we start with a translation produced with an earlier version of our system, Fig. 2. Then we discuss some possible improvements of this translation, which leads to the translation in Fig. 3, which is the output from the current system. The machinery behind the improvements will be explained later.

3.1 The OCL Specification

We consider the OCL specification for the method `check` of the class `OwnerPIN`. `OwnerPIN` stores the PIN code of a smart card, and keeps track of the maximal number of attempts allowed to present the correct PIN before the card is locked. The purpose of the method `check` is to compare a given PIN number with the PIN value in the `OwnerPIN` class itself. If they match and the PIN is not blocked, it sets the validated flag and resets the try counter to its maximum. If it does not match, it decrements the try counter and, if the counter has reached zero, blocks the PIN. The try counter is specified in the first element of

```

context OwnerPIN
def: let tryCounter = self.triesLeft->at(1)

context OwnerPIN::check(pin: Sequence(Integer),
    offset: Integer, length: Integer): Boolean
post: self.tryCounter = 0 implies result = false
post: (self.tryCounter > 0 and pin <> null and offset >= 0 and length >= 0
    and offset+length <= pin->size()
    and Util.arrayCompare(self.pin, 0, pin, offset, length) = 0
    ) implies (result = true and self.isValidated() and tryCounter = maxTries)
post: (self.tryCounter > 0 and not (pin <> null and offset >= 0 and length >= 0
    and offset+length <= pin->size()
    and Util.arrayCompare(self.pin, 0, pin, offset, length) = 0)
    ) implies (not self.isValidated() and self.tryCounter = tryCounter@pre-1 and
    (( not excThrown(java::lang::Exception) and result = false)
    or excThrown(java::lang::NullPointerException)
    or excThrown(java::lang::ArrayIndexOutOfBoundsException)))

```

Fig. 1. OCL specification from the Java Card API

the attribute `triesLeft`, which is an array. The validated flag can be accessed using the `isValidated()` method. The PIN comparison can be done using the `arrayCompare()` method which is defined in the `Util` class of the JavaCard API. Fig. 1 shows the OCL specification of `check` (including a definition of a helper attribute `tryCounter`).

3.2 A First Attempt

In Fig. 2 we show the translation of the OCL specification produced by an earlier version of our system. The English text is basically correct, but it is clumsy and very hard to read.

for the class `OwnerPIN` introduce the following definition : the `tryCounter` is defined as the element at index 1 of the `triesLeft` of the `ownerPIN` for the operation `check (pin : Seq(Integer) , offset : Integer , length : Integer) : Boolean` of the class `javacard::framework::OwnerPIN` the following holds : the following postconditions should hold : (*) if the `tryCounter` of the `ownerPIN` is equal to 0 , the result is equal to false (*) if the `tryCounter` of the `ownerPIN` is greater than 0 and `pin` is not equal to null and `offset` is at least 0 and `length` is at least 0 and `offset plus length` is at most the size of `pin` and the query `arrayCompare (the pin of the ownerPIN , 0 , pin , offset , length)` to `Util` is equal to 0 , the result is equal to true and the query `isValidated ()` holds for the `ownerPIN` and the `tryCounter` of the `ownerPIN` is equal to the `maxTries` of the `ownerPIN` (*) if the `tryCounter` of the `ownerPIN` is greater than 0 and it is not the case that `pin` is not equal to null and `offset` is at least 0 and `length` is at least 0 and `offset plus length` is at most the size of `pin` and the query `arrayCompare (the pin of the ownerPIN , 0 , pin , offset , length)` to `Util` is equal to 0 , it is not the case that the query `isValidated ()` holds for the `ownerPIN` and the `tryCounter` of the `ownerPIN` is equal to the `tryCounter` of the `ownerPIN` at the beginning of the Operation minus 1 and it is not the case that an exception is thrown and the result is equal to false or a `NullPointerException` is thrown or an `ArrayIndexOutOfBoundsException` is thrown

Fig. 2. Translation of OCL specification (before)

3.3 An Improved Translation

Fig. 3 shows an improved version of the translation: the output of the current version of our system. Below we go through the improvements made. Each one is quite simple in itself, but the end result is in our opinion a text of acceptable quality, which shows that our approach works for non-trivial specifications.

for the class **OwnerPIN** introduce the following definition :

- the try counter is defined as the element at index 1 of the `triesLeft` attribute

for the operation **check (pin : Sequence(Integer) , offset : Integer , length : Integer) : Boolean** of the class **javacard::framework::OwnerPIN** ,
the following post-conditions should hold :

- if the try counter is equal to 0 then this implies that the result is equal to false
- if the following conditions are true
 - the try counter is greater than 0
 - *pin* is not equal to null
 - *offset* is at least 0
 - *length* is at least 0
 - *offset* plus *length* is at most the size of *pin*
 - the query `arrayCompare (the pin , 0 , pin , offset , length)`¹ on `Util` is equal to 0
 then this implies that the following conditions are true
 - the result is equal to true
 - this owner PIN is validated
 - the try counter is equal to the maximum number of tries
- if the try counter is greater than 0 and at least one of the following conditions is not true
 - *pin* is not equal to null
 - *offset* is at least 0
 - *length* is at least 0
 - *offset* plus *length* is at most the size of *pin*
 - the query `arrayCompare (the pin , 0 , pin , offset , length)`² on `Util` is equal to 0
 then this implies that the following conditions are true
 - this owner PIN is not validated
 - the try counter is equal to the previous value of the try counter minus 1
 - at least one of the following conditions is true
 - * an exception is not thrown and the result is equal to false
 - * a null pointer exception is thrown
 - * an array index out of bounds exception is thrown

¹ Compares the specified source array, beginning at the specified position, with the destination array beginning at the specified position from left to right. A result of 0 indicates that the arrays are equal.

² Compares the specified source array, beginning at the specified position, with the destination array beginning at the specified position from left to right. A result of 0 indicates that the arrays are equal.

Fig. 3. Translation of OCL specification (after)

Formatting. Two of the most important problems in Fig. 2 are (1) the specification is just a big piece of text, where the structure is very hard to discern, and (2) it is hard or impossible to determine the scope of the and:s and or:s, which

makes the specification ambiguous. To address these problems we introduce formatting: line breaks are inserted, keywords are printed in bold and arguments to the method are italicized. Furthermore, lists of constraints, as well as sequences of and/or statements are made into bullet lists. The formatting consists of HTML or \LaTeX tags in the text, what we see in Fig. 3 is the \LaTeX version.

Negation. The scope of the negations is hard to determine, and negating sentences as in e.g. “it is not the case that an exception is thrown” is a clumsy construction. Using “an exception is not thrown” instead solves both these problems.

Making Use of the Context. In Fig. 2 the text “of the ownerPIN” is used very frequently. Since the specification is given as postconditions in the context of a method of the class `OwnerPIN`, we should be able to just leave out all occurrences of “of the ownerPIN”, resulting in a much less repetitive text. (In OCL we are allowed to do the same thing, by leaving out `self`.) This can be seen as a simple case of referring expressions generation [6].

Domain-Specific Vocabulary. Based on the type and capitalization of identifiers, we can improve the translation of domain-specific vocabulary. For instance, the class `OwnerPIN` can be automatically translated as “owner PIN”, instead of just “ownerPIN” as in Fig. 2.

The attributes `triesLeft` and `maxTries` are similarly translated as “the tries left” and “the max tries” by default. Although these translations are quite adequate, we can improve this even further by making manual changes. Thus, using our system we can by hand change the translation to “the triesLeft attribute” and “the maximum number of tries”, for these attributes.

Two methods are used in this OCL constraint: `isValidated` and `arrayCompare`. For `isValidated`, which returns a boolean, we introduce some simple heuristics which by default translates it to “. . . is validated” instead of “the query `isValidated()` holds”. The second method used, `arrayCompare`, gets linearized to “the query `arrayCompare (the pin, 0 , pin , offset , length)` on `Util`”. Unfortunately this method is not so easily translated into simple English. The task it carries out does not fit nicely as part of the translated constraint. To solve this problem, we use the ‘note’ facility provided in the grammar. We can by hand add a note for the method, and this will then be displayed as a tool-tip when HTML formatting is used or, as in this case, as a footnote when \LaTeX formatting is used (a current limitation is the needless duplication of footnotes).

4 System Overview

The system is built around a GF grammar for specifications: there is an abstract syntax giving rules for how to form abstract syntax trees of specifications, as well as three concrete syntaxes to present abstract syntax trees in OCL, English and German, respectively.

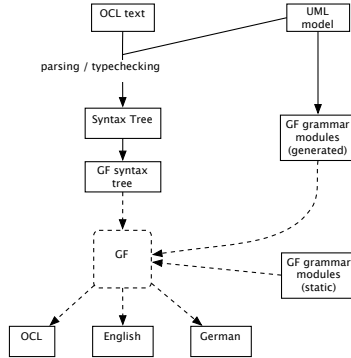


Fig. 4. From OCL to Natural Language

Given this grammar, the GF system provides us with a syntax editor where we can edit specifications in OCL, English and German in parallel. We also get parsers and linearizers for OCL and (fragments of) English and German, which we can use for translating e.g. OCL specifications into English, by first parsing OCL into an abstract syntax, and then linearizing the tree into English.

However, our system does not just consist of a GF grammar and the functionality provided directly by GF. What makes things more complex is that (1) parts of the grammar are dynamically generated depending on the context (the grammar is not closed), and (2) we use a separate program for parsing OCL specifications and (3) turning them into GF abstract syntax trees. These external programs (as well as the GF system itself) are implemented in the functional language Haskell. Fig. 4 shows the overall structure of the system.

There are two current prototypes of our system: one which allows syntax editing of specifications in OCL and English inside KeY, another which allows batch-translation from OCL to English (yet to be integrated more closely with KeY). The concrete German grammar has so far only been used for small examples [17]. The system is available for download [16].

Syntax editing of specifications is briefly described in [2], except for the integration with the KeY system. In this paper we focus on taking existing OCL specifications into English.

4.1 External Programs

Grammar Generation. An OCL specifications uses domain specific vocabulary as defined by a UML model. When a user adds e.g. a class or an attribute to the model, he also extends the language of specifications of that model. We therefore generate GF grammar modules from the UML model to dynamically extend the grammar with domain specific vocabulary. This is described in more detail in Sect. 5.2. The general idea of dynamically extending a grammar with user-defined concepts is also used in [18].

External Parser. Given an OCL specification and a UML model, the OCL is first parsed using a standard context free parser, and then type checked with respect to the UML model, resulting in an annotated syntax tree of the OCL specification. This step is described in detail in [19].

The original motivation for adding an external parser was that the parser derived by GF for our particular grammar had termination problems.² However, there are also more general reasons for using an external parser: (1) *Efficiency*: An external, context free parser for a formal language—in our case OCL—is more efficient than a parser derived from a GF grammar. (2) *Modularity*: The GF abstract syntax does not have to handle all particularities of OCL. For instance, OCL has various implicit forms which require disambiguation (see e.g. [19]). This can be done by the external parser and typechecker.

Transformation Into GF Trees. The annotated trees returned by the external parser are transformed into GF abstract syntax trees. The context free structure for most parts maps into the GF abstract syntax in a straightforward way. However, we also perform some structural transformations in order to improve the quality of the natural language, e.g. the transformation described in Sect. 5.3. These transformations could probably be avoided by extending the GF abstract and concrete syntax instead, but we believe that expressing the transformations in the Haskell programming language instead of the GF formalism is in this case a simpler and more modular solution.

5 Grammar Engineering

We start this section by giving a very brief introduction to our GF grammar, to give a general idea of what writing an application grammar for specifications amounts to (Sect. 5.1). We then explain how the improvements described in the motivating example section are implemented in the GF grammar. There is not enough room for describing everything, so we give two representative examples: dynamically extending the grammar with domain specific concepts (Sect. 5.2), and formatting (Sect. 5.3). Throughout this section, we make use excerpts from the grammars without explaining all details of the GF formalism.

5.1 An Application Grammar for Specifications

Representing Specifications: Abstract Syntax. In a typical GF application grammar, the abstract syntax part is used for defining a semantic domain, without any linguistic considerations. We define categories (types) and functions which gives the rules for how to form trees in these categories. In our case, we are

² As explained in [2], this is because our grammar makes use of dependent types in such a way that the derived GF parser, which in a first step disregards dependent types, contained cyclic rules. In the meantime, this problem has been given a general solution in [20] (the implementation of which is in progress).

interested in the domain of (OCL) specifications. We consider OCL specifications as expressions formed using the attributes and queries of the classes in the UML model (including the predefined OCL types). For each class in the UML model, there will be a corresponding GF function c as well as a (dependent) category **Instance** c representing expressions of that class. As an example, this is how the **size** query of the OCL library class **String** (which returns the length of a string as an integer) is represented in GF abstract syntax judgements:

```
cat Class; cat Instance (c:Class); fun StringC, IntegerC: Class;
fun size : Instance StringC -> Instance IntegerC;
```

This defines the GF function **size** as taking trees of type **Instance StringC** and returning something of type **Instance IntegerC**. Note that with the dependent category **Instance**, we have introduced type-checking into the grammar: only trees representing type correct specifications can be built. As already mentioned, this leads to complications for the GF derived parser, see [2] for an explanation of this.

There are many choices to be made on exactly how to model specifications in abstract syntax, most of which we do not discuss here. One example, however, is introducing a category **Sent** for representing sentences. It is for instance used for the equality operator, with which we can state that any two instances x and y of the same class c are equal:

```
cat Sent;
fun equal : (c:Class) -> (x,y : Instance c) -> Sent;
```

The introduction of **Sent** is motivated by the fact that in natural language, we want to distinguish between expressions and sentences. In OCL, however, there is no such distinction – sentences just correspond to an expression of boolean type. This is an example of an interlingua problem: if a semantic distinction is made in one language, it has to be introduced into the abstract syntax, even if it is not present in the other languages.

Using the Resource Grammars: Concrete Syntax. In the concrete syntax, we give linearization rules presenting abstract syntax trees in English and German (we will not discuss the concrete syntax for OCL). To each category C in the abstract syntax, we must associate a record type: the *linearization category* of C . For each function in the abstract syntax, we define a linearization rule which builds a record in the corresponding linearization category. For instance, we might start with the abstract category **Class**, to be treated like common noun phrases in concrete syntax:³

```
param Number = Sg | Pl;
lincat Class = {s : Number => Str};
lin IntegerC = {s = table {Sg => "integer"; Pl => "integers"}};
```

³ This example is simplified: in the real grammar, **Class** has a more complex linearization category which represents something more than just common noun phrases.

Here we define a parameter type for number. The linearization category of **Class** is a record type which has one field **s**, which is a string inflected in number. Then we define the linearization of **IntegerC** as the noun “integer” (in singular and plural form). To complete the concrete syntax, we would then have to go on defining **linCat:s** and **lin** rules for the rest of the abstract syntax types and functions, along with the required record and parameter types. However, we instead make use of the GF resource grammar library [17].

The resource grammars provide an API of linguistically motivated record and parameter types, along with utility functions to be used in linearization rules. For instance, there is a parameter type **Number**, as well as a record type **CN** for common noun phrases. Using the resource grammars raises the level of abstraction in the concrete syntax: instead of dealing directly with issues of e.g. inflection or word order, we can use the linguistic structures provided by the resource API. As long as we use only the API, all type-correct uses of the resource grammar preserve grammaticality. Since the API is available for seven languages including English and German, we can often reuse the same concrete syntax. Without explaining any details, an example of our use of the resource grammar is the following English linearization rule for the **size** method:

```
lin size x = DefOneNP (AppFun (funOfCN
                               (useN (nNonhuman "length")))) x);
```

The functions used on the right hand side in the linearization rule are part of the resource grammar API. The linearization of the tree **size x** using this rule will be “the length of *x*”. To provide the German linearization “die Länge von *x*”, the rule is almost the same:

```
lin size x = DefOneNP (AppFun (funVonCN (useN (nFrau "Länge"))))x);
```

5.2 Domain-Specific Vocabulary

In order to translate from OCL to English, the grammar needs to contain information about the UML model upon which the OCL is based. A grammar generation program therefore generates GF modules based on the UML model. The automatic generation does not always produce the most suitable translation, therefore it is also possible for the user to manually improve the translation by modifying the generated grammars, in particular the linearization rules in the concrete syntax.

To aid in the construction of the domain-specific concrete module, a resource module has been defined, which contains many operations that are useful when linearizing classes, attributes etc. We call this the API for Domain-specific Vocabulary. This API provides a layer of abstraction which hides some of the complexity of the rest of the grammar, making it easier to generate the linearizations for the domain entities. It also makes subsequent hand modifications possible without full knowledge about GF and the resource grammars.

Using the API. Although the API uses concepts taken from the OCL grammar and from the resource grammars, the interface provided is simple enough to not

require a deep knowledge of the underlying grammars. We will consider OCL classes as an example. The following operation is provided for constructing the linearization of a class (**ClassL** is the linearization category of **Class**):

```
oper mkClass: CN -> Str -> ClassL;
```

Classes are defined as consisting of a common noun phrase (**CN**) that corresponds to the class name as it will appear in natural text, and an identifier (a string), which is the actual name of the class in the UML diagram. The class identifier is used when formally specifying the class name.

Common ways of constructing the **CN** are included in the API, such as constructing a **CN** from a String, or adding an adjective to the **CN**. Irregular ways of constructing a **CN** can be found in the resource grammar modules. Take for example the class **OwnerPIN**, which is linearized using operations defined in the API.

```
lin OwnerPIN = mkClass (adjCN "owner" (strCN "PIN")) "OwnerPIN";
```

This will result in the class name being represented as a common noun phrase in natural text: “the maximum PIN size of the owner PIN is greater than 0” while the class identifier is used in a more formal setting: “for the class **OwnerPIN** the following invariants hold:”.

Grammar Generation. The grammar generator uses some heuristics to derive a reasonable linearization for a domain entity (a class, an attribute, a method or an association) from its name and type. Given a UML model, it produces an abstract syntax module with one function for each domain entity, and a concrete module with corresponding English linearizations. A concrete module with OCL linearizations is also generated. The concrete English module makes use the API for domain-specific vocabulary, and the resource grammars. Since GF supports separate compilation of modules, only these generated modules need to be recompiled whenever the UML model changes, not the whole grammar.

The heuristics is based on types, and on splitting an identifier into words based on capitalization. E.g., the identifier **OwnerPIN** is split into the strings “owner” and “PIN”. Since we also know the type, i.e. in this case that **OwnerPIN** is the name of class, we build a noun “owner PIN” as described just above. Another simple rule is special handling of boolean properties that start with “is”, e.g. **isValidated** becomes a sentence saying “... is validated”.

The heuristics for grammar generation obviously depends on the natural language used for identifiers, in this case English. A good heuristics for German would be more complex, e.g. it would require access to a lexicon for determining the gender of nouns. The heuristics also requires a consistent convention for word boundaries in identifiers (e.g. is it **isValidated** or **is_validated**?).

Modifying the Grammar. The generated grammar does not always succeed in producing the best translation. Using the API it is possible to make hand-modifications to the generated grammar without too much difficulty. For example, the attributes **triesLeft** and **maxTries** are translated as “the tries left” and “the max tries” by default using the generated judgements we see below.

```
lin maxTries = mkSimpleProperty (adjCN "max" (strCN "tries"));
lin triesLeft = mkSimpleProperty (adjCN "tries" (strCN "left"));
```

Although these translations are quite adequate, we can improve them by making manual changes to the generated grammar using some of the operations provided in the API. Thus, using the judgements below, we can construct the text “the triesLeft attribute” and “the maximum number of tries”, for these attributes.

```
lin maxTries = mkSimpleProperty (ofCN (adjCN "maximum"
                                     (strCN "number")) (strCN "tries"));
lin triesLeft = mkSimpleProperty (attrCN "triesLeft");
```

5.3 Grammar-Based Formatting

The use of formatting in the translated text has a dramatic effect on the readability of the output. As we see in the motivating example in Fig. 3, the formatting includes e.g. breaking the text into paragraphs, using different fonts for headings and argument variables, and presenting various structures in the form of bullet lists.

Most of this formatting is done completely on the level of concrete syntax: An interface module has been defined that contains operations required to perform formatting tasks, without specifying an implementation. This interface can then be implemented in many different ways using different instances. Currently three instances exist, allowing the possibility to have no formatting, HTML formatting or \LaTeX formatting. These instances do not define their own *pretty-printing* rules, instead they simply use formatting tags leaving the actual layout to be handled by the \LaTeX and HTML rendering engines. The linearization rules of the concrete syntax then makes use of the operations specified by the formatting interface.

Using Lists for Aggregation. There is one exception to the rule that all formatting is done just in concrete syntax: formatting lists of conjunctions and disjunctions as bullet lists. This requires changes also in the abstract syntax, as well as support from the external program which transforms the result of the context free OCL parser into GF abstract syntax.

To treat lists of conjunctions (the machinery for disjunctions is just the same) in a special way we simply introduce a new category **AndList** in the abstract syntax, along with functions for creating such lists, and converting them into sentences:

```
fun oneAnd : Sent -> Sent -> AndList;
fun consAnd : Sent -> AndList -> AndList;
fun andList2Sent : AndList -> Sent;
```

The base case **oneAnd** takes two sentences and builds an **AndList**, containing just one conjunction. The function **consAnd** prepends a sentence to an existing **AndList**. Once the list is built, **andList2Sent** allows us to consider it as a sentence.

A list containing just one conjunction, i.e. a tree **oneAnd** $x\ y$ would be linearized just as “ x and y ”, while using **consAnd** should result in a bullet list, saying “the following conditions are true: ...”. However, this is dealt with in the concrete syntax (which is omitted here), the abstract syntax just provides the required structure. In fact, we use the same kind of abstract syntax for sums and products, where we are not interested in formatting. In that case, the problem is to translate e.g. an OCL expression $2+3$ as “2 plus 3”, but $1+2+3$ as “the sum of 1, 2 and 3”.

When translating OCL to English, we must find OCL expressions where lists of conjunctions occur, and make sure that they are treated as **AndLists**. This can be seen as simple aggregation problem [6]. As mentioned above, this step is not performed inside the grammars, but in the transformation from context free OCL syntax trees (as returned by the external parser) to GF abstract syntax trees.

6 Related Work

Natural Language Generation (NLG) is described in [6] as producing understandable natural language text from a non-linguistic representation of information. This very general description also fits GF linearization: Linearization can be considered as a two-step procedure, where the linearization rules go from non-linguistic abstract syntax to linguistically motivated resource grammar constructions. The resource grammar implementation then takes the step to surface strings in natural language. However, while linearization is therefore clearly more than just linguistic realization (cf. the discussion in [6] on realization as the inverse of parsing), it is much simpler than a typical NLG system. Linearization rules (and the resource grammars) are written in GF concrete syntax, a restricted functional language, and linearization rules are always compositional. In contrast, [6] describes a typical NLG system architecture as a pipeline consisting of the phases text planning, sentence planning and linguistic realization, along with separate intermediate representation formats.

When using our system for translating OCL to English (as opposed to syntax editing), there is also the external OCL parser / typechecker and grammar generation, and the architecture is more similar to that of a compiler than a NLG system. We also do some transformations to the GF syntax trees in an external program. Some of these transformations could be described in terms of NLG concepts, e.g. aggregation (Sect. 5.3). They are also similar to some of the ideas in [21], which describes generation of natural language text from formal proofs as a process resembling code generation in a compiler.

7 The Case Study in Numbers

The Java Card API case study consists of OCL specifications of 37 classes, the word count of the English translation is close to 17000. The generated grammar modules of domain-specific vocabulary contain about 1100 concepts, i.e. 1100 abstract syntax functions, each one with a corresponding linearization rule. 361 of

these concepts are actually being used in the translated specifications. By-hand modifications were made to the linearization rules of 73 of these 361 concepts, i.e. 20% of the used domain-specific concepts needed modifications. 18 of these modifications are of a trivial nature and could probably be automated if one introduces domain-specific heuristics for grammar generation, which leaves 15% of the used domain-specific concepts that require non-trivial modifications.

8 Conclusion

We have presented a tool for translating formal OCL specifications into natural language based on GF grammars. By adding a domain-specific vocabulary API, formatting, and other stylistic improvements, we achieve a translation of a non-trivial case study of OCL specifications which is acceptable to a human reader. Relatively few by-hand modifications using the API were necessary; the modifications are made on the grammar level, but do not require linguistic or GF expertise. Although we add external programs, the compositional and declarative GF formalism remains the centre of our work: the external programs are used either to generate GF grammar modules, or to manipulate GF abstract syntax trees. The tool and the Java Card API case study are available on the web [16].

8.1 Future Work

Important lines of future work include: (1) Further improvements to the natural language, e.g. by more sophisticated use of aggregation and referring expressions generation. (2) A more formal evaluation of the quality of the generated natural language. (3) Integration into the KeY system, most importantly providing a user interface for manipulating domain-specific vocabulary, based on the API we have defined.

Acknowledgements

We thank Reiner Hähnle, Aarne Ranta and the anonymous referees for valuable suggestions on how to improve the paper.

References

1. Ahrendt, W., Baar, T., Beckert, B., Bubel, R., Giese, M., Hähnle, R., Menzel, W., Mostowski, W., Roth, A., Schlager, S., Schmitt, P.H.: The KeY tool. *Software and System Modeling* **4** (2005) 32–54
2. Hähnle, R., Johannisson, K., Ranta, A.: An authoring tool for informal and formal requirements specifications. In Kutsche, R.D., Weber, H., eds.: *Fundamental Approaches to Software Engineering*. Number 2306 in LNCS (2002)
3. Ranta, A.: Grammatical Framework: A Type-theoretical Grammar Formalism. *The Journal of Functional Programming* **14** (2004) 145–189

4. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis, Napoli (1984)
5. Sun Microsystems: Java card homepage (2004) <http://java.sun.com/products/javacard/>.
6. Reiter, E., Dale, R.: Building applied natural language generation systems. *Journal of Natural Language Engineering* **3** (1997) 57–87
7. Ranta, A.: Grammatical Framework homepage (2005) www.cs.chalmers.se/~aarne/GF.
8. Khegai, J., Nordström, B., Ranta, A.: Multilingual syntax editing in GF. In Gelbukh, A., ed.: *CICLing-2003*, Mexico City, Mexico. LNCS, Springer (2003)
9. Ranta, A.: The GF resource grammar library (2004) <http://www.cs.chalmers.se/~aarne/GF/lib/resource/>.
10. Sommerville, I.: *Software Engineering*. Seventh edn. Addison Wesley (2004)
11. The Object Management Group: Object constraint language specification (2004) <http://www.omg.org/docs/formal/03-03-13.pdf>.
12. The Object Management Group: Unified modelling language homepage (2004) <http://www.uml.org>.
13. Larsson, D., Mostowski, W.: Specifying Java Card API in OCL. In Schmitt, P.H., ed.: *OCL 2.0 Workshop at UML 2003*. Volume 102C of ENTCS., Elsevier (2004) 3–19
14. Meijer, H., Poll, E.: Towards a full formal specification of the Java Card API. In Attali, I., Jensen, T., eds.: *Smart Card Programming and Security*. Number 2140 in LNCS, Springer (2001) 165–178
15. Burke, D.A.: Improving the natural language translation of formal software specifications. Master's thesis, Chalmers University of Technology, SE-412 96 Göteborg, Sweden (2004)
16. Johannisson, K.: OCL to natural language tool homepage (2004) <http://www.cs.chalmers.se/~krijo/gfspec/>.
17. Daniels, H.J.: Eine deutsche Grammatik für OCL. Studienarbeit (2003) <http://www.cs.chalmers.se/~krijo/gfspec/>.
18. Hallgren, T., Ranta, A.: An extensible proof text editor. In Parigot, M., Voronkov, A., eds.: *Logic for Programming and Automated Reasoning, LPAR*. LNAI 1955, Springer (2000) 70–84
19. Johannisson, K.: Disambiguating implicit constructions in OCL (2004) Online proceedings of OCL and Model Driven Engineering Workshop at UML 2004, <http://www.cs.kent.ac.uk/projects/ocl/oclmdeuuml04/description.htm>.
20. Ljunglöf, P.: Expressivity and complexity of the Grammatical Framework. PhD thesis, Chalmers University of Technology, Göteborg University, SE-412 96 Göteborg, Sweden (2004)
21. Coscoy, Y., Kahn, G., Thery, L.: Extracting text from proofs. In Dezani-Ciancaglini, M., Plotkin, G., eds.: *Proc. Second Int. Conf. on Typed Lambda Calculi and Applications*. Volume 902 of LNCS. (1995) 109–123

On the Selective Lambek Calculus[★]

Marcelo da S. Corrêa¹ and E. Hermann Haeusler²

¹ Instituto de Matemática, Universidade Federal Fluminense,
Rua Mario S. Braga s/n, Niterói RJ 24020-140, Brazil
`ganmarc@vm.uff.br`,

`correa@informatik.uni-tuebingen.de`

² Departamento de Informática,
Pontifícia Universidade Católica do Rio de Janeiro,
Rua Marquês de São Vicente, 225,
Rio de Janeiro, RJ 22453-900, Brazil
`hermann@inf.puc-rio.br`

Abstract. An intermediate logical system in comparison with the non-associative (**NL**) and the associative Lambek Calculus(**L**) is obtained by extending their language by means of a so-called “combining permission”, which is used to regulate the introduction of a product formula in the scope of a derivation and to impose a control on applications of the association structural rules. A cut elimination theorem for such a system, so-called Selective Lambek Calculus, is presented.

1 Introduction

The variants of the Lambek Calculus (**L**), formerly *Calculus of Syntactic Types*[9], have been in evidence nowadays due to its many applications in linguistics in the scope of the type logic categorial grammars[15]. The non-associative Lambek Calculus(**NL**) [10], considered the pure logic of residuation [14], has been settled as the basic logic of a substructural hierarchy of logics concerning the resource sensitivity in the context of the Gentzen systems. While the Gentzen presentation for **NL** is characterized by the absence of structural rules, other systems can be obtained from it by including general structural rules for Associativity, as **L**, or for Permutation, as **NLP**, or both as **LP**[2], undermining *the sensitivity to aspects of resource structure* of **NL**(quoted from [8]).

There are approaches in which the resource sensitivity is partially undermined by considering structural modalities, as in the system **NL** \diamond [13], or combinators as in the *Structurally Free Logic*[7]. In such systems the structure of terms marked with a given structural modality or those ones affected by a combinator can be changed or they can be freely permuted with other term, as representing

[★] This work was Elaborated During a Post-Doctoral Staying of the First Author in the Wilhelm-Schickard-Institut, University of Tübingen and it was Partially Supported by the Brazilian Foundation CAPES (PROBRAL/CAPES/DAAD-175/04).

a special and particular property of such elements. In another direction, we propose to regain associativity adopting special association structural rules and performing a kind of contextual analysis in the sense that the structure of a term could be changed if the new structure “makes sense”, i.e., if it is actually possible to combine the considered expressions to form an acceptable one. This is captured by a notion of “combining permission” used to regulate the introduction of a product formula in the scope of a derivation and also to impose a control on the application of an association structural rule, by means of incorporating special premises into the rule for introducing the product operator at the right of a sequent and into the association structural rules. This approach was adopted by the authors in a first version of the system **Selective Lambek Calculus**[5], here denoted by **SL**. In this work, we change deeply the presentation of the rules in order to obtain a cut elimination theorem for **SL**.

2 Relaxing the Non-associative Lambek Calculus

The language \mathcal{F} of formulas (or types) of the non-associative Lambek Calculus **NL** is recursively obtained by closing a denumerable set \mathcal{A} of atomic formulas (or basic types) under the following binary connectives (or type forming operators): one binary (tensor) product \bullet and its left and right residual implications, denoted here by \leftarrow and \rightarrow , respectively: $\mathcal{F} ::= \mathcal{A} \mid \mathcal{F} \leftarrow \mathcal{F} \mid \mathcal{F} \bullet \mathcal{F} \mid \mathcal{F} \rightarrow \mathcal{F}$.

We shall consider that atomic formulas will be denoted by Latin capital letters and meta-variables over type formulas by Greek small letters.

In a Gentzen presentation of **NL**, a sequent consists of an antecedent (a non-empty rooted binary tree) and a succedent (a formula) separated by an operator \vdash , such that binary trees are written as terms formed from formulas and the structural operator (\cdot, \cdot) . The language \mathcal{T} of terms is recursively determined as $\mathcal{T} ::= \mathcal{F} \mid (\mathcal{T}, \mathcal{T})$.

The Greek capital letter Γ , with or without a subscript or a superscript, is used as meta-variable over terms. The notation $\Gamma[\Gamma_1]$ denotes a term Γ containing a distinguished occurrence of a sub-term Γ_1 and $\Gamma[\Gamma_2]$ is the result of replacing exactly such an occurrence of Γ_1 in Γ by Γ_2 .

In the presentation of the inference rules for **NL**, on the following, distinguished occurrences of subterms in upper and bottom sequents are assumed to occupy the same position in a term Γ , as indicated in [14].

A Gentzen presentation for the associative Lambek Calculus **L** can be obtained from **NL** by considering the following association structural rule, in which the double line indicates that it may be applied in both directions.

$$\frac{\Gamma[((\Gamma_1, \Gamma_2), \Gamma_3)] \vdash \delta}{\Gamma[(\Gamma_1, (\Gamma_2, \Gamma_3))] \vdash \delta} A$$

A cut elimination theorem can be proved for both systems[9, 1, 6]. Besides, they could be compared by means of the following characteristic theorems and derived rules of inference, as presented in [14] but which are related to some

$$\begin{array}{c}
\frac{}{\alpha \vdash \alpha} \text{ax} \qquad \frac{\Gamma' \vdash \alpha \quad \Gamma[\alpha] \vdash \gamma}{\Gamma[\Gamma'] \vdash \gamma} \text{cut} \\
\\
\frac{\Gamma_1 \vdash \alpha \quad \Gamma_2 \vdash \beta}{(\Gamma_1, \Gamma_2) \vdash \alpha \bullet \beta} \bullet \mathcal{R} \qquad \frac{\Gamma[(\alpha, \beta)] \vdash \gamma}{\Gamma[\alpha \bullet \beta] \vdash \gamma} \bullet \mathcal{L} \\
\\
\frac{(\alpha, \Gamma) \vdash \beta}{\Gamma \vdash \alpha \rightarrow \beta} \rightarrow \mathcal{R} \qquad \frac{\Gamma_1 \vdash \alpha \quad \Gamma[\beta] \vdash \gamma}{\Gamma[(\Gamma_1, \alpha \rightarrow \beta)] \vdash \gamma} \rightarrow \mathcal{L} \\
\\
\frac{(\Gamma, \alpha) \vdash \beta}{\Gamma \vdash \beta \leftarrow \alpha} \leftarrow \mathcal{R} \qquad \frac{\Gamma_1 \vdash \alpha \quad \Gamma[\beta] \vdash \gamma}{\Gamma[(\beta \leftarrow \alpha, \Gamma)] \vdash \gamma} \leftarrow \mathcal{L}
\end{array}$$

Fig. 1. A sequent system for **NL**

basic principles of Lambek[9]. In fact, all of the items (1) to (11) hold in **L**, but only (1) to (6) hold in **NL**.

- | | |
|-------------------------------|---|
| 1. Application: | $\alpha \bullet (\alpha \rightarrow \beta) \vdash \beta$ and $(\beta \leftarrow \alpha) \bullet \alpha \vdash \beta$ |
| 2. Co-application: | $\alpha \vdash \beta \rightarrow (\beta \bullet \alpha)$ $\alpha \vdash (\alpha \bullet \beta) \leftarrow \beta$ |
| 3. Monotonicity: | If $\alpha \vdash \beta$ and $\delta \vdash \gamma$, then $\alpha \bullet \delta \vdash \beta \bullet \gamma$ |
| 4. Isotonicity: | If $\alpha \vdash \beta$, then $\gamma \rightarrow \alpha \vdash \gamma \rightarrow \beta$
and $\alpha \leftarrow \gamma \vdash \beta \leftarrow \gamma$ |
| 5. Antitonicity: | If $\alpha \vdash \beta$, then $\beta \rightarrow \gamma \vdash \alpha \rightarrow \gamma$
and $\gamma \leftarrow \beta \vdash \gamma \leftarrow \alpha$ |
| 6. Lifting: | $\alpha \vdash \beta \leftarrow (\alpha \rightarrow \beta)$ and
$\alpha \vdash (\beta \leftarrow \alpha) \rightarrow \beta$ |
| 7. Geach (main functor): | $\alpha \rightarrow \beta \vdash (\gamma \rightarrow \alpha) \rightarrow (\gamma \rightarrow \beta)$ and
$\beta \leftarrow \alpha \vdash (\beta \leftarrow \gamma) \leftarrow (\alpha \leftarrow \gamma)$ |
| 8. Geach (secondary functor): | $\alpha \rightarrow \beta \vdash (\alpha \rightarrow \gamma) \leftarrow (\beta \rightarrow \gamma)$ and
$\beta \leftarrow \alpha \vdash (\gamma \leftarrow \beta) \rightarrow (\gamma \leftarrow \alpha)$ |
| 9. Composition: | $(\alpha \rightarrow \beta) \bullet (\beta \rightarrow \gamma) \vdash \alpha \rightarrow \gamma$ and
$(\gamma \leftarrow \beta) \bullet (\beta \leftarrow \alpha) \vdash \gamma \leftarrow \alpha$ |
| 10. Restructuring: | $(\alpha \rightarrow \beta) \leftarrow \gamma \vdash \alpha \rightarrow (\beta \leftarrow \gamma)$ and
$\alpha \rightarrow (\beta \leftarrow \gamma) \vdash (\alpha \rightarrow \beta) \leftarrow \gamma$ |
| 11. (De)Currying: | $(\alpha \bullet \beta) \rightarrow \gamma \vdash \vdash \beta \rightarrow (\alpha \rightarrow \gamma)$ and
$\gamma \leftarrow (\alpha \bullet \beta) \vdash \vdash (\gamma \leftarrow \beta) \leftarrow \alpha$. |

The derivations for the items (7) to (11) are usually based on the general associativity of the operator “ \bullet ,” that corresponds also to a structural postulate of associativity for \bullet [14, 9]. However, we could prove the properties (7) to (10) if we add two extra rules, for introducing the residual implications in the left side of a sequent, to the collection of rules of **NL**, as follows.

$$\frac{\Gamma_1 \vdash \alpha \quad \Gamma[(\beta, \Gamma_2)] \vdash \gamma}{\Gamma[(\Gamma_1, (\alpha \rightarrow \beta, \Gamma_2))] \vdash \gamma} \rightarrow \mathcal{L}^* \qquad \frac{\Gamma_1 \vdash \alpha \quad \Gamma[(\Gamma_2, \beta)] \vdash \gamma}{\Gamma[(((\Gamma_2, \beta \leftarrow \alpha), \Gamma_1))] \vdash \gamma} \leftarrow \mathcal{L}^*$$

This would demand to assign the type $s \leftarrow s$ to the expression “Poor”, which could suggest that an adjective affects a sentence instead of a noun. However, a derivation of the sequent $(A \leftarrow A, (A, A \rightarrow B)) \vdash B$ can be obtained similarly to such a derivation presented above to one case of the (De)Currying property.

3 Combining Permissions and the Selective Lambek Calculus

In order to obtain a proper control of how the structure of a term could be changed, we propose to consider special association structural rules with extra premises that it will restrict its applications to some desired cases, including those related to the Geach, Composition and Restructuring properties but avoiding the (De)Currying ones according to the motivation problem presented above. Such a control will be determined by considering a so-called *combining permission*, which is a concept motivated by the notion of *sequencing arrows* used to obtain a categorical characterization for a non-commutative tensor product [4] and which it was introduced in a first presentation of the **Selective Lambek Calculus** [5]. Intuitively, the inference of a combining permission between to formulas α and β , denoted by $\alpha \nabla \beta$, justifies the formation of a product formula $\alpha \bullet \beta$ in the scope of a derivation.

The sequent system for the Selective Lambek Calculus, here denoted by **SL**, consists of a collection of logical rules, two association structural rules and a collection of rules for introducing combining permissions. In this work, the presentation of the rules is deeply changed to obtain a cut elimination theorem for this formulation of the system.

The usage of combining permissions to allow, or do not, a change in the structure of a term is based on the fact that each occurrence of the operator “,” must be properly converted to an occurrence of the connective \bullet . Hence, if one would like to transform the term $((\alpha, \beta), \gamma)$ to a term $(\alpha, (\beta, \gamma))$, for instance, it must be obtained combining permissions $\beta \nabla \gamma$ and $\alpha \nabla (\beta \bullet \gamma)$, since this corresponds to transform the formula $(\alpha \bullet \beta) \bullet \gamma$ to the formula $\alpha \bullet (\beta \bullet \gamma)$. In fact, this corresponds that there are combining permissions that support the formation of the new formula. The notion of combining permission is generalized in this study to allow that the operator ∇ could be applied to terms, as in $\Gamma_1 \nabla \Gamma_2$, more than just to formulas.

Moreover, while having a combining permission between formulas α and $\alpha \rightarrow \beta$, for instance, is exactly what it is expected, a so-called *basic combining permission* relating two atomic formulas (that represent basic types as n, np, s, \dots) could not be very reasonable in the linguistic sense, but it will play a technical role in our approach in such a way that a derivation will be decorated with the basic combining permissions needed to infer the combining permissions that will justify the occurrences of the connective \bullet . For this reason a sequent is structured as

$$[\Delta] \Gamma \vdash \gamma,$$

such that:

- The term Γ is called the *hypothesis context* of the sequent;
- Δ is a set of basic combining permissions and it is called the *permission context*. Notation:
 - ★ $[]$ denotes the empty set of basic combining permission, and
 - ★ $[\Delta_1, \Delta_2, \dots, \Delta_n]$ denotes the set $\Delta_1 \cup \Delta_2 \cup \dots \cup \Delta_n$ of basic combining permissions.

The extra premisses of the association rules will consist of so-called *auxiliary sequents*. An auxiliary sequent is structured as

$$\Delta \Rightarrow \Gamma_1 \nabla \Gamma_2,$$

considering that Δ contains the basic combining permissions needed to infer $\Gamma_1 \nabla \Gamma_2$. In the case that Δ is the empty set we denote an auxiliary sequent just as $\Rightarrow \Gamma_1 \nabla \Gamma_2$.

Hence, for instance, one of the association structural rules of **SL** can be formulated as follows:

$$\frac{[\Delta_1] \Gamma[(\Gamma_1, \Gamma_2), \Gamma_3] \vdash \delta \quad \Delta_2 \Rightarrow \Gamma_2 \nabla \Gamma_3 \quad \Delta_3 \Rightarrow \Gamma_1 \nabla (\Gamma_2, \Gamma_3)}{[\Delta_1, \Delta_2, \Delta_3] \Gamma[(\Gamma_1, (\Gamma_2, \Gamma_3))] \vdash \delta} \text{ ASS1}$$

For introducing a formula $\alpha \bullet \beta$ in the right side of a sequent it is also demanded that a combination permission $\alpha \nabla \beta$ must be inferred.

$$\frac{[\Delta_1] \Gamma_1 \vdash \alpha \quad [\Delta_2] \Gamma_2 \vdash \beta \quad \Delta_3 \Rightarrow \alpha \nabla \beta}{[\Delta_1, \Delta_2, \Delta_3] (\Gamma_1, \Gamma_2) \vdash \alpha \bullet \beta} \bullet \mathcal{R}$$

One can think about including a fourth premise in such a rule related to the introduction of a combining permission $\Gamma_1 \nabla \Gamma_2$ between the hypothesis contexts Γ_1 e Γ_2 of its respective upper sequents to turn explicit the correspondence between the operator “,” and the connective \bullet . However, this is not necessary since, by means of the Theorem1 presented in the next section, we assure that if there is a derivation of a sequent, its permission context contains all basic combining permissions needed to infer the combining permissions related to any occurrence of the operator “,” in such a derivation. This will be proved even by considering the rule $\bullet \mathcal{L}$ for introducing a formula $\alpha \bullet \beta$ in the hypothesis context similar to that one adopted in **NL**, i.e, without including an extra premise related to the introduction of a combining permission $\alpha \nabla \beta$.

The cut rule and the rules for introducing the residual implications also differ from those ones adopted in **NL** only by the permission contexts that are incorporated into the sequents. However, the axiom is formulated exactly to atomics formulas, since every occurrence of connective \bullet on the right side of a sequent must be justified by the introduction of a combining permission.

Let us consider here rules $\rightarrow \mathcal{L}^*$ and $\leftarrow \mathcal{L}^*$ similar to those ones presented in the previous section, obtained just by incorporating the permission contexts into the sequents. To assure that these rules are derived rules in **SL**, we must obtain combining permissions related to the new occurrences of the operator “,” in its respective bottom sequents in order to apply one of the association structural

$$\begin{array}{c}
\frac{}{[] \alpha \vdash \alpha} ax, \text{ if } \alpha \text{ is an atomic formula} \quad \frac{[\Delta_1] \Gamma' \vdash \alpha \quad [\Delta_2] \Gamma[\alpha] \vdash \beta}{[\Delta_1, \Delta_2] \Gamma[\Gamma'] \vdash \beta} cut \\
\\
\frac{[\Delta_1] \Gamma_1 \vdash \alpha \quad [\Delta_2] \Gamma_2 \vdash \beta \quad \Delta_3 \Rightarrow \alpha \nabla \beta}{[\Delta_1, \Delta_2, \Delta_3] (\Gamma_1, \Gamma_2) \vdash \alpha \bullet \beta} \bullet \mathcal{R} \quad \frac{[\Delta] \Gamma[(\alpha, \beta)] \vdash \gamma}{[\Delta] \Gamma[\alpha \bullet \beta] \vdash \gamma} \bullet \mathcal{L} \\
\\
\frac{[\Delta] (\alpha, \Gamma) \vdash \beta}{[\Delta] \Gamma \vdash \alpha \rightarrow \beta} \rightarrow \mathcal{R} \quad \frac{[\Delta_1] \Gamma_1 \vdash \alpha \quad [\Delta_2] \Gamma[\beta] \vdash \gamma}{[\Delta_1, \Delta_2] \Gamma[(\Gamma_1, \alpha \rightarrow \beta)] \vdash \gamma} \rightarrow \mathcal{L} \\
\\
\frac{[\Delta] (\Gamma, \alpha) \vdash \beta}{[\Delta] \Gamma \vdash \beta \leftarrow \alpha} \leftarrow \mathcal{R} \quad \frac{[\Delta_1] \Gamma_1 \vdash \alpha \quad [\Delta_2] \Gamma[\beta] \vdash \gamma}{[\Delta_1, \Delta_2] \Gamma[(\beta \leftarrow \alpha, \Gamma_1)] \vdash \gamma} \leftarrow \mathcal{L} \\
\\
\frac{[\Delta_1] \Gamma[(\Gamma_1, \Gamma_2), \Gamma_3] \vdash \delta \quad \Delta_2 \Rightarrow \Gamma_2 \nabla \Gamma_3 \quad \Delta_3 \Rightarrow \Gamma_1 \nabla (\Gamma_2, \Gamma_3)}{[\Delta_1, \Delta_2, \Delta_3] \Gamma[(\Gamma_1, (\Gamma_2, \Gamma_3))] \vdash \delta} ASS1 \\
\\
\frac{[\Delta_1] \Gamma[(\Gamma_1, (\Gamma_2, \Gamma_3))] \vdash \delta \quad \Delta_2 \Rightarrow \Gamma_1 \nabla \Gamma_2 \quad \Delta_3 \Rightarrow (\Gamma_1, \Gamma_2) \nabla \Gamma_3}{[\Delta_1, \Delta_2, \Delta_3] \Gamma[(\Gamma_1, \Gamma_2), \Gamma_3] \vdash \delta} ASS2
\end{array}$$

Fig. 2. Collection of logical and structural rules for **SL**

rules ASS1 or ASS2. In fact, part of the rules for introducing combining permissions, depicted in Figure 3, can be motivated by trying to fulfill these requirements. However, such a property will just be stated as the Corollary8 in the next section, since it depends on the Theorem1 as it explained on the following.

$$\frac{[\Delta_1] \Gamma_1 \vdash \alpha \quad [\Delta_2] \Gamma[(\beta, \Gamma_2)] \vdash \gamma}{[\Delta_1, \Delta_2] \Gamma[(\Gamma_1, (\alpha \rightarrow \beta, \Gamma_2))] \vdash \gamma} \rightarrow \mathcal{L}^* \quad \frac{[\Delta_1] \Gamma_1 \vdash \alpha \quad [\Delta_2] \Gamma[(\Gamma_2, \beta)] \vdash \gamma}{[\Delta_1, \Delta_2] \Gamma[(\Gamma_2, \beta \leftarrow \alpha, \Gamma_1)] \vdash \gamma} \leftarrow \mathcal{L}^*$$

For instance, considering the rule $\rightarrow \mathcal{L}^*$, we would like to have the following derivation π :

$$\pi \equiv \frac{\frac{[\Delta_1] \Gamma_1 \vdash \alpha \quad [\Delta_2] \Gamma[(\beta, \Gamma_2)] \vdash \gamma}{[\Delta_1, \Delta_2] \Gamma[(\Gamma_1, \alpha \rightarrow \beta), \Gamma_2] \vdash \gamma} \rightarrow \mathcal{L} \quad \begin{array}{c} P_1 \\ \Delta'_2 \Rightarrow (\alpha \rightarrow \beta) \nabla \Gamma_2 \end{array} \quad \begin{array}{c} P_2 \\ \Delta_1 \Rightarrow \Gamma_1 \nabla (\alpha \rightarrow \beta, \Gamma_2) \end{array}}{[\Delta_1, \Delta_2, \Delta'_2] \Gamma[(\Gamma_1, (\alpha \rightarrow \beta, \Gamma_2))] \vdash \gamma} ASS1$$

such that $\Delta_1 \cup \Delta_2 \cup \Delta'_2 = \Delta_1 \cup \Delta_2$, since $\Delta'_2 \subset \Delta_2$ as it will be noticed below, and:

- P_1 and P_2 are called *auxiliary derivations*. In general, an auxiliary derivation consists on applications of the rules for introducing combining permission, presented in the Figure3, possibly having derivations of sequents as sub-derivations.
- The rules for introducing combining permissions $\mathcal{S}4$ and $\mathcal{S}12$ are mainly motivated to be used in an auxiliary derivation as the following one:

$$P_2 \equiv \frac{[\Delta_1] \Gamma_1 \vdash \alpha \quad \frac{}{\Rightarrow \alpha \nabla (\alpha \rightarrow \beta, \Gamma_2)} \mathcal{S}4}{\Delta_1 \Rightarrow \Gamma_1 \nabla (\alpha \rightarrow \beta, \Gamma_2)} \mathcal{S}12$$

The existence of a combining permission like $\alpha \nabla (\alpha \rightarrow \beta, \Gamma_2)$ or $\alpha \nabla (\alpha \rightarrow \beta)$ introduced by the rules $\mathcal{S}4$ or $\mathcal{S}2$, respectively, are accepted as primitive

elements, since they are quite natural in the linguistic sense. These rules, and similarly the rules $\mathcal{S}3$ and $\mathcal{S}5$, are special kinds of axioms related to the process of constructing auxiliary derivations.

- The rule $\mathcal{S}6$ captures the idea that what can be combined at right of $(\alpha \rightarrow \beta)$ is the same that can be combined at right of β , since $(\alpha \rightarrow \beta)$ combined at right of α produces β . Thus, it is meant to be used exactly to obtain an auxiliary derivation as P_1 :

$$P_1 \equiv \frac{\Delta'_2 \Rightarrow \beta \nabla \Gamma_2}{\Delta'_2 \Rightarrow (\alpha \rightarrow \beta) \nabla \Gamma_2} \overset{P_3}{\mathcal{S}6}$$

In fact, the existence of such an auxiliary derivation P_1 depends on the existence of an auxiliary derivation P_3 of $\Delta'_2 \Rightarrow \beta \nabla \Gamma_2$ for some $\Delta'_2 \subset \Delta_2$, what is a consequence of the Theorem1 related to the derivation π_2 of the sequent $[\Delta_2] \Gamma[(\beta, \Gamma_2)] \vdash \gamma$.

A derivation π' can be obtained by applying the rules $\leftarrow \mathcal{L}$, $\text{ASS}2$, $\mathcal{S}5$, $\mathcal{S}13$ and $\mathcal{S}7$, to show that $\leftarrow \mathcal{L}^*$ is also a derived rule in **SL**.

To motivate the remaining rules for introducing combining permissions ($\mathcal{S}8$, $\mathcal{S}9$, $\mathcal{S}10$, $\mathcal{S}11$ and $\mathcal{S}1$), let us consider the following auxiliary derivations that introduced the combining permissions $(A \bullet B) \nabla C$ and $A \nabla (B \bullet C)$, for atomic formulas A , B and C .

$$\frac{\frac{B \nabla C \Rightarrow B \nabla C}{B \nabla C \Rightarrow (A, B) \nabla C} \mathcal{S}10}{B \nabla C \Rightarrow (A \bullet B) \nabla C} \mathcal{S}8 \qquad \frac{\frac{A \nabla B \Rightarrow A \nabla B}{A \nabla B \Rightarrow A \nabla (B, C)} \mathcal{S}11}{A \nabla B \Rightarrow A \nabla (B \bullet C)} \mathcal{S}9$$

The rules $\mathcal{S}8$ and $\mathcal{S}9$ just allow to convert the operator “,” to the connective \bullet , without any restriction. The rule $\mathcal{S}10$ capture the view that if it does not exist any occurrence of the residual implications in the term at left, its structure can be broken (looking upwards) in a such way that the introduction of the considered combining permission will depend on the introduction of a combining permission between its right part and the original right term. The rule $\mathcal{S}11$ is dual to $\mathcal{S}10$.

The rule $\mathcal{S}1$, another kind of axiom, is meant to represent that a combining permission between atomic formulas is considered as an extra information, since it is stored at the left side of the auxiliary sequent. This is also done to allow that the basic combining permission that represents an important information about the possibility of combining two complex terms can be passed till the bottom auxiliary sequent.

The rules for introducing combining permissions are drawn to allow that derivations for all of the properties (1) to (10), presented in the previous section, can be constructed. But considering just the cases in which the considered formulas could be properly formed, i.e., it must be possible to obtain combining permissions related to the occurrences of the connective \bullet .

Notice that, as the expression reveals, an auxiliary derivation is just a special sub-derivation in the scope of a derivation of a particular sequent, having no isolated application. On the following, we present some examples to illustrate the construction of derivations and their auxiliary derivations in **SL**.

$$\begin{array}{c}
\overline{\alpha \nabla \beta \Rightarrow \alpha \nabla \beta} \text{ S1, if } \alpha \text{ and } \beta \text{ are atomic formulae.} \\
\\
\overline{\Rightarrow \alpha \nabla (\alpha \rightarrow \beta)} \text{ S2} \qquad \overline{\Rightarrow (\beta \leftarrow \alpha) \nabla \alpha} \text{ S3} \\
\overline{\Rightarrow \alpha \nabla ((\alpha \rightarrow \beta), \Gamma)} \text{ S4} \qquad \overline{\Rightarrow (\Gamma, (\beta \leftarrow \alpha)) \nabla \alpha} \text{ S5} \\
\frac{\Delta \Rightarrow \beta \nabla \Gamma}{\Delta \Rightarrow (\alpha \rightarrow \beta) \nabla \Gamma} \text{ S6} \qquad \frac{\Delta \Rightarrow \Gamma \nabla \beta}{\Delta \Rightarrow \Gamma \nabla (\beta \leftarrow \alpha)} \text{ S7} \\
\frac{\Delta \Rightarrow (\alpha, \beta) \nabla \Gamma}{\Delta \Rightarrow (\alpha \bullet \beta) \nabla \Gamma} \text{ S8} \qquad \frac{\Delta \Rightarrow \Gamma \nabla (\alpha, \beta)}{\Delta \Rightarrow \Gamma \nabla (\alpha \bullet \beta)} \text{ S9} \\
\frac{\Delta \Rightarrow \Gamma_2 \nabla \Gamma}{\Delta \Rightarrow (\Gamma_1, \Gamma_2) \nabla \Gamma} \text{ S10 (*)} \qquad \frac{\Delta \Rightarrow \Gamma \nabla \Gamma_1}{\Delta \Rightarrow \Gamma \nabla (\Gamma_1, \Gamma_2)} \text{ S11 (*)} \\
\\
\frac{[\Delta'] \Gamma' \vdash \alpha \quad \Delta \Rightarrow \alpha \nabla \Gamma}{\Delta', \Delta \Rightarrow \Gamma' \nabla \Gamma} \text{ S12, if } \Gamma' \not\models \alpha \\
\frac{[\Delta'] \Gamma' \vdash \alpha \quad \Delta \Rightarrow \Gamma \nabla \alpha}{\Delta', \Delta \Rightarrow \Gamma \nabla \Gamma'} \text{ S13, if } \Gamma' \not\models \alpha
\end{array}$$

(*) Both Γ_1 and Γ_2 cannot contain any occurrence of the connectives \leftarrow and \rightarrow .

Fig. 3. Rules for Introducing Combining Permissions of **SL**

(i) Restructuring property for atomic formulas:

$$\frac{\frac{[\] ((A, (A \rightarrow B) \leftarrow C), C) \vdash B}{[\] (A, (A \rightarrow B) \leftarrow C) \vdash B \leftarrow C} \leftarrow \mathcal{R} \quad \pi'}{[\] (A \rightarrow B) \leftarrow C \vdash A \rightarrow (B \leftarrow C)} \rightarrow \mathcal{R}$$

such that the sub-derivation π' can be obtained in the two following ways:

- Applying the rules ASS2 and $\leftarrow \mathcal{L}$:

$$\frac{\frac{\frac{[\] A \vdash A \quad [\] B \vdash B}{[\] C \vdash C \quad [\] A, A \rightarrow B \vdash B} \rightarrow \mathcal{L} \quad \frac{\overline{\Rightarrow A \nabla (A \rightarrow B)}}{\Rightarrow A \nabla (A \rightarrow B) \leftarrow C} \text{ S2} \quad \frac{\overline{\Rightarrow (A, (A \rightarrow B) \leftarrow C) \nabla C}}{\Rightarrow (A, (A \rightarrow B) \leftarrow C) \nabla C} \text{ S5}}{[\] ((A, (A \rightarrow B) \leftarrow C), C) \vdash B} \leftarrow \mathcal{L} \quad \text{S7} \quad \text{S5} \quad \text{ASS2}$$

- Applying the rule $\leftarrow \mathcal{L}^*$:

$$\frac{\frac{[\] A \vdash A \quad [\] B \vdash B}{[\] C \vdash C \quad [\] A, A \rightarrow B \vdash B} \rightarrow \mathcal{L}}{[\] ((A, (A \rightarrow B) \leftarrow C), C) \vdash B} \leftarrow \mathcal{L}^*$$

(ii) $[A \nabla B, B \nabla C] (A \bullet B) \bullet C \vdash A \bullet (B \bullet C)$

$$\frac{\frac{[A \nabla B, B \nabla C] (A, (B, C)) \vdash A \bullet (B \bullet C) \quad \frac{\overline{B \nabla C \Rightarrow B \nabla C}}{B \nabla C \Rightarrow (A, B) \nabla C} \text{ S1} \quad \frac{\overline{A \nabla B \Rightarrow A \nabla B}}{A \nabla B \Rightarrow (A, B) \nabla C} \text{ S10}}{[A \nabla B, B \nabla C] ((A, B), C) \vdash A \bullet (B \bullet C)} \text{ ASS2} \quad \bullet \mathcal{L} \\
\frac{[A \nabla B, B \nabla C] (A, B) \bullet C \vdash A \bullet (B \bullet C)}{[A \nabla B, B \nabla C] (A \bullet B) \bullet C \vdash A \bullet (B \bullet C)} \bullet \mathcal{L}$$

Recalling that for any derivation of a sequent there must be a cut-free derivation of the considered sequent, as it will be assured by the Theorem 9, we conjecture that it is not possible to obtain a derivation of the considered property. Similarly, it would be not possible to obtain in **SL** a derivation of the sequent related to the motivation example presented in the section 2:

$$[\Delta](A \leftarrow A) \bullet (A \bullet (A \rightarrow B)) \vdash B$$

for any set of basic combining permissions Δ , since one cannot introduce a combining permission $(A \leftarrow A) \nabla (A, (A \rightarrow B))$. Neither a derivation of a sequent

$$[\Delta](A \leftarrow A) \bullet (A \bullet (A \rightarrow B)) \vdash (A \leftarrow A) \bullet (A \bullet (A \rightarrow B)).$$

Hence, the derivability notion of **SL** is not reflexive, which also points out the selective character intended in our approach.

4 Properties

In this section, we present the concepts and results needed to assure that any occurrence of the operator “,” and the connective \bullet in a derivation are justified by a combining permission and we also present a cut elimination theorem for **SL**.

A derivation is **cut-free** if it does not contain any application of the cut rule. Hence, an auxiliary derivation P is said to be **cut-free** if P contains only cut-free derivations of the sequents that occur as premisses of applications of the rules $\mathcal{S}12$ and $\mathcal{S}13$.

We consider that a hypothesis context of a derivable sequent is *justified* by the set Δ of those basic combining permissions needed to obtain an auxiliary derivation of a combining permission related to each occurrence of the operator “,” in such a hypothesis context. In the particular case in which a hypothesis context consists of a single formula, we should just consider that Δ is the empty set. We shall say that a hypothesis context is **cut-free justified** by a set of basic combining permissions if there are cut-free auxiliary derivations that introduce the required combined permission. These concepts are recursively defined in the following:

Definition 0: Let α be a formula, let Γ , Γ_1 and Γ_2 be hypothesis contexts and Δ a permission context.

Γ is (cut-free) justified by Δ iff:

- $\Gamma \equiv \alpha$ and $\Delta \equiv \emptyset$; or
- $\Gamma \equiv (\Gamma_1, \Gamma_2)$ and there exist permission contexts Δ_1 , Δ_2 and Δ_3 , such that
 - $\Delta \equiv \Delta_1 \cup \Delta_2 \cup \Delta_3$,
 - Γ_1 is (cut-free) justified by Δ_1 ,
 - Γ_2 is (cut-free) justified by Δ_2 and
 - there exists a (cut-free) auxiliary derivation of the auxiliary sequent $\Delta_3 \Rightarrow \Gamma_1 \nabla \Gamma_2$.

Some rules for introducing connectives, as $\bullet \mathcal{R}$, $\leftarrow \mathcal{L}$, and $\rightarrow \mathcal{L}$ and even the cut rule generate new occurrences of the “,” operator in the left context of their bottom sequents without verifying if there exists a combining permission related to such occurrences. We show on the sequel that this is actually unnecessary,

by proving that the hypothesis context of any derivable sequent is justified by a subset of its respective permission context.

Theorem 1: Let Γ be a hypothesis context, Δ a permission context and γ a formula.

If there is a cut-free derivation of the sequent $[\Delta] \Gamma \vdash \gamma$, then:

- i) Γ is cut-free justified by a set of basic combining permissions $\Delta' \subset \Delta$, and
- ii) there is a cut-free auxiliary derivation of an auxiliary sequent $\Delta'' \Rightarrow \alpha_1 \nabla \alpha_2$, for some $\Delta'' \subset \Delta$, for any subformula $\alpha_1 \bullet \alpha_2$ of γ and of any formula occurrence of Γ .

A proof of the Theorem1 is done by induction on the number of inferences of a derivation, applying the Lemma4, the Corollary5 and the Lemma6 on the following. Special attention must be paid to the cases in which the bottom sequent of a derivation is obtained by an application of $\bullet\mathcal{L}$, $\leftarrow\mathcal{L}$, $\rightarrow\mathcal{L}$, ASS1 and ASS2, since new occurrences of the operator “ ∇ ” are generated in the left context of a sequent. We consider on the following the case in which the rule $\bullet\mathcal{R}$ is applied.

$$\pi \equiv \frac{\frac{[\Delta_1] \Gamma_1 \vdash \alpha \quad [\Delta_2] \Gamma_2 \vdash \beta}{[\Delta_1, \Delta_2, \Delta_3] (\Gamma_1, \Gamma_2) \vdash \alpha \bullet \beta} \quad \Delta_3 \Rightarrow \alpha \nabla \beta}{[\Delta_1, \Delta_2, \Delta_3] (\Gamma_1, \Gamma_2) \vdash \alpha \bullet \beta} \bullet\mathcal{R}$$

By induction hypothesis, we have that the context Γ_1 is cut-free justified by some subset Δ'_1 of Δ_1 , since there is a cut-free derivation π_1 of $[\Delta_1]$. Considering also that the context (α, β) is cut-free justified by Δ_3 , the Lemma6 ensures that the context (Γ_1, β) is cut-free justified by $\Delta'_1 \cup \Delta_3$.

By induction hypothesis, the context Γ_2 is cut-free justified by some subset Δ'_2 of Δ_2 , considering the cut-free derivation π_2 of $[\Delta_2] \Gamma_2 \vdash \beta$. Therefore, by Lemma6, the context (Γ_1, Γ_2) is cut-free justified by $\Delta'_1 \cup \Delta'_2 \cup \Delta_3$, which is a subset of $\Delta_1 \cup \Delta_2 \cup \Delta_3$.

Notice that there is no new formula occurrences in the hypothesis context Γ_1, Γ_2 in comparison with the hypothesis contexts of the upper sequents of $\bullet\mathcal{R}$. Hence, by induction hypothesis, we assure the second property holds for the hypothesis context of the bottom sequent of π .

To prove the cases in which the association structural rules are applied, we must also consider an operator Φ , which generates a formula from a hypothesis context by replacing occurrences of “ ∇ ” with the connective \bullet , similar to the operator $()^\circ$ adopted by Moortgat in [14].

Definition 2: Let α be a formula and let Γ, Γ_1 and Γ_2 be non-empty hypothesis contexts Φ is a mapping from the collection of hypothesis contexts into the collection of formulae, such that:

- if $\Gamma \equiv \alpha$, then, $\Phi(\Gamma) = \alpha$;
- if $\Gamma \equiv (\Gamma_1, \Gamma_2)$, then, $\Phi(\Gamma) = \Phi(\Gamma_1) \bullet \Phi(\Gamma_2)$.

Lemma 3: Let Δ be a permission context and Γ_1 and Γ_2 hypothesis contexts. If there is a (cut-free) auxiliary derivation of the auxiliary sequent

$$\Delta \Rightarrow \Gamma_1 \nabla \Gamma_2,$$

then there are (cut-free) auxiliary derivations of the following auxiliary sequents

$$\Delta \Rightarrow \Gamma_1 \nabla \Phi(\Gamma_2) \quad \text{and} \quad \Delta \Rightarrow \Phi(\Gamma_1) \nabla \Gamma_2.$$

We can prove the Lemma 3 by a simple induction on the number of inferences of an (arbitrary) auxiliary derivation. In the induction step, the desired (cut-free) auxiliary derivations are obtained from an assumed (cut-free) auxiliary derivation of the auxiliary sequent mentioned in the antecedent by applying the rules $\mathcal{S}8$ and $\mathcal{S}9$, respectively. Notice that, in the cases in which the rules $\mathcal{S}12$ or $\mathcal{S}13$ are applied, we could obtain a (cut-free) derivation of a sequent $[\Delta'] \Phi(\Gamma') \vdash \alpha$ from a given (cut-free) derivation of $[\Delta'] \Gamma' \vdash \alpha$ by successive applications of the rule $\bullet\mathcal{L}$.

Lemma 4: Let Γ be a hypothesis context, Δ^* be a set of basic combining permissions and γ a formula. For any (cut-free) derivation π of a sequent

$$[\Delta^*] \Gamma \vdash \gamma,$$

if Γ is (cut-free) justified by a set of basic combining permissions $\Delta \subset \Delta^*$, and there is a (cut-free) auxiliary derivation of an auxiliary sequent $\Delta^{**} \Rightarrow \alpha_1 \nabla \alpha_2$, for some $\Delta^{**} \subset \Delta^*$, for any subformula $\alpha_1 \bullet \alpha_2$ of any formula occurrence in Γ , then:

- (i) there is a (cut-free) derivation π' of a sequent $[\Delta'] \gamma \vdash \gamma$, for some $\Delta' \subset \Delta$;
- (ii) there is a (cut-free) derivation π'' of the sequent $[\Delta''] \delta \vdash \delta$, for some $\Delta' \subset \Delta$, for any formula δ which occurs in Γ .

This lemma is proved by induction on the length of a derivation.

Corollary 5: Let Γ be a hypothesis context, Δ^* be a set of basic combining permissions and γ a formula. For any (cut-free) derivation π of a sequent $[\Delta^*] \Gamma \vdash \gamma$, if Γ is (cut-free) justified by a set of basic combining permissions $\Delta \subset \Delta^*$, and there is a (cut-free) auxiliary derivation of an auxiliary sequent $\Delta^{**} \Rightarrow \alpha_1 \nabla \alpha_2$, for some $\Delta^{**} \subset \Delta^*$, for any subformula $\alpha_1 \bullet \alpha_2$ of any formula occurrence in Γ , then there is a (cut-free) derivation π' of the sequent $[\Delta'] \Gamma' \vdash \Phi(\Gamma')$, for some $\Delta' \subset \Delta$, for any subcontext Γ' of Γ .

A proof of this result follows from the Lemma4, since each subcontext Γ' is (cut-free) justified by a subset Δ' of Δ and each formula occurrence δ of Γ' also occurs in Γ . Then, there is a (cut-free) auxiliary derivation of a combining permission related to each occurrence of the operator “ ∇ ” in Γ' and there is a (cut-free) derivation of a sequent $[\Delta''] \delta \vdash \delta$ for each formula occurrence δ of Γ' and $\Delta'' \subset \Delta'$. Therefore a (cut-free) derivation of $[\Delta'] \Gamma' \vdash \Phi(\Gamma')$ is obtained by successive applications of $\bullet\mathcal{R}$.

Lemma 6: Let Γ be a hypothesis context, Δ^* be a set of basic combining permissions and γ a formula. Let π be a (cut-free) derivation of a sequent

$[\Delta^*] \Gamma \vdash \gamma$, such that Γ is (cut-free) justified by a set of basic combining permissions $\Delta \subset \Delta^*$, and there is a (cut-free) auxiliary derivation of an auxiliary sequent $\Delta^{**} \Rightarrow \alpha_1 \nabla \alpha_2$, for some $\Delta^{**} \subset \Delta^*$, for any subformula $\alpha_1 \bullet \alpha_2$ of any formula occurrence in Γ .

If there exist hypothesis contexts Γ' and Γ'' and permission contexts Δ' and Δ'' , such that:

- Γ' occurs in Γ ,
- Γ' is (cut-free) justified by $\Delta' \subset \Delta$,
- Γ'' is (cut-free) justified by Δ'' , and
- there is a derivation of the sequent $[\Delta', \Delta''] \Gamma'' \vdash \Phi(\Gamma')$,

then

- (i) the hypothesis context $\Gamma[\Gamma'']$, which is obtained from Γ by replacing exactly one occurrence of Γ' with Γ'' , is (cut-free) justified by some $\Delta''' \subset \Delta \cup \Delta''$, and
- (ii) For any subcontext Γ^* of Γ , including Γ itself, there is a (cut-free) derivation of the sequent $[\Delta'''] \Gamma^*[\Gamma''] \vdash \Phi(\Gamma^*[\Gamma'])$ for some $\Delta''' \subset \Delta \cup \Delta''$.

A proof for this lemma is done by structural induction on a hypothesis context Γ , applying the Lemma3 and the rules $\mathcal{S}12$ and $\mathcal{S}13$ to obtain the required combined permissions and the Corollary6 to obtain the sub-derivations necessary to prove the item (ii).

As a consequence of the Theorem1, we have that the hypothesis context of any derivable sequent is justified by a subset of the respective permission context.

Corollary 7: Let Γ be a hypothesis context, Δ a permission context and γ a formula.

If there is a derivation of the sequent $[\Delta] \Gamma \vdash \gamma$, then:

- i) Γ is justified by a set of basic combining permissions $\Delta' \subset \Delta$, and
- ii) there is an auxiliary derivation of an auxiliary sequent $\Delta'' \Rightarrow \alpha_1 \nabla \alpha_2$, for some $\Delta'' \subset \Delta$, for any subformula $\alpha_1 \bullet \alpha_2$ of γ and of any formula occurrence of Γ .

A proof of this can be obtained directly from the proof of the Theorem1 by just considering (arbitrary) derivations and auxiliary derivations and including a case in the induction step in which the derivation is obtained by an application of the cut rule.

As it was pointed out in the previous section, we can now assure that the rules $\rightarrow \mathcal{L}^*$ and $\leftarrow \mathcal{L}^*$ are derived rules in **SL**.

Corollary 8: The rules $\rightarrow \mathcal{L}^*$ and $\leftarrow \mathcal{L}^*$ are derived from $\rightarrow \mathcal{L}$ and $\leftarrow \mathcal{L}$ and the association structural rules ASS1 and ASS2 in **SL**.

We have obtained a Gentzen's style proof of the cut-elimination theorem for the considered system, in accordance with the presentation of Takeuti[17].

Theorem 9 (Cut Elimination:) If there is a derivation of a sequent S , then there is a cut-free derivation of S .

The proof of this theorem is a consequence of Lemma10, presented on the following, and could be done by induction on the number of applications of the cut rule in a derivation of a sequent.

Lemma 10: If π is a proof of a sequent $[\Delta] \Gamma \vdash \gamma$ which contains only one application of the cut rule occurring as its last inference, then there is a cut-free derivation of such a sequent.

This lemma is proved by on a double induction on a measure of complexity of a derivation based on the *grade* of the cut formula and the notion of *rank* of a derivation, as presented in [17]. We present on the following the part of the proof that deals with a case in which the rank of the considered derivation π is greater than 2, since the right upper sequent of the cut rule is a lower sequent of the structural associative rule ASS1. Considering the case in which the occurrence of the cut-formula α belongs to Γ_1 , we have that the derivation π must be as follows:

$$\frac{[\Delta'] \overset{\pi_1}{\Gamma'} \vdash \alpha \quad [\Delta_1, \Delta_2, \Delta_3] \overset{\pi_2}{\Gamma}[(\Gamma_1[\alpha], (F_2, F_3))] \vdash \gamma}{[\Delta_1, \Delta_2, \Delta_3, \Delta'] \Gamma[(\Gamma_1[\Gamma'], (F_2, F_3))] \vdash \gamma} \text{ cut}$$

such that

$$\pi_2 \equiv \frac{[\Delta_1] \Gamma[(\Gamma_1[\alpha], F_2), F_3] \vdash \gamma \quad \overset{P_1}{\Delta_2 \Rightarrow F_2 \nabla F_3} \quad \overset{P_2}{\Delta_3 \Rightarrow \Gamma_1[\alpha] \nabla (F_2, F_3)}}{[\Delta_1, \Delta_2, \Delta_3] \Gamma[(\Gamma_1[\alpha], (F_2, F_3))] \vdash \gamma} \text{ ASS1}$$

Then, we obtain the following derivation:

$$\pi_3 \equiv \frac{[\Delta'] \overset{\pi_1}{\Gamma'} \vdash \alpha \quad [\Delta_1] \overset{\pi_2}{\Gamma}[(\Gamma_1[\alpha], F_2), F_3] \vdash \gamma}{[\Delta_1, \Delta'] \Gamma[(\Gamma_1[\Gamma'], F_2), F_3] \vdash \gamma} \text{ cut}$$

Notice that $\nu(\pi_3) < \nu(\pi)$, since $\text{rank}(\pi_3) < \text{rank}(\pi)$. Then, by the considered induction hypothesis, there is a cut-free derivation π_4 of the sequent

$$[\Delta_1, \Delta'] \Gamma[(\Gamma_1[\Gamma'], F_2), F_3] \vdash \gamma.$$

Regarding to the cut-free derivation π'_2 , by Theorem1, $\Gamma[(\Gamma_1[\alpha], F_2), F_3]$ is cut-free justified by a set of basic combining permission $\Delta'_1 \subset \Delta_1$ and there is a cut-free auxiliary derivation of an auxiliary sequent $\Delta''_1 \Rightarrow \alpha_1 \nabla \alpha_2$, for some $\Delta''_1 \subset \Delta_1$, for any subformula $\alpha_1 \bullet \alpha_2$ of any formula occurrence in $\Gamma[(\Gamma_1[\alpha], F_2), F_3]$.

Then, the sub-context Γ_1 is cut-free justified by some $\Delta'''_1 \subset \Delta'_1$.

Considering the cut-free derivation π_1 , the Theorem1 ensures that Γ' is cut-free justified by the some $\Delta'' \subset \Delta'$.

Then, by Lemma6, there is a cut-free derivation π_5 of $[\Delta'''] \Gamma_1[\Gamma'] \vdash \Phi(\Gamma_1[\alpha])$ for some $\Delta''' \subset \Delta'_1 \cup \Delta''$.

By Lemma3 and regarding that P_2 is a cut-free auxiliary derivation of the auxiliary sequent $\Delta_3 \Rightarrow \Gamma_1[\alpha] \nabla (F_2, F_3)$, there is a cut-free auxiliary derivation P_3 of $\Delta_3 \Rightarrow \Phi(\Gamma_1[\alpha]) \nabla (F_2, F_3)$.

Then, we obtain the following cut-free auxiliary derivation:

$$P_4 \equiv \frac{[\Delta'''] \Gamma_1[\Gamma'] \vdash \Phi(\Gamma_1[\alpha]) \quad \Delta_3 \Rightarrow \Phi(\Gamma_1[\alpha]) \nabla (\Gamma_2, \Gamma_3)}{\Delta_3 \Rightarrow \Gamma_1[\Gamma'] \nabla (\Gamma_2, \Gamma_3)} \text{S12}$$

Hence, we obtain the following cut-free derivation:

$$\frac{[\Delta_1, \Delta'] \Gamma[(\Gamma_1[\Gamma'], \Gamma_2), \Gamma_3] \vdash \gamma \quad \Delta_2 \Rightarrow \Gamma_2 \nabla \Gamma_3 \quad \Delta_3 \Rightarrow \Gamma_1[\Gamma'] \nabla (\Gamma_2, \Gamma_3)}{[\Delta_1, \Delta_2, \Delta_3, \Delta'] \Gamma[(\Gamma_1[\Gamma'], (\Gamma_2, \Gamma_3))] \vdash \gamma} \text{ASS1}$$

5 Conclusion and Future Works

We have presented a framework in which intermediate systems between the non-associative and the associative Lambek Calculus can be formulated, making possible to consider questions about the non-associativity of the operator \bullet (and the operation of combining expression represented by it) in a more sensible way. Instead of consider unary modalities, as in [14], to recover the capability to change the association in a term, we adopted a notion of combining permission and performed a kind of contextual analysis to deal with such a problem.

However, the rules for introducing combining permissions $\mathcal{S}12$ and $\mathcal{S}13$ cause some troubles with regard to the decidability of the system, since they call on a formula that could not occur in the considered bottom auxiliary sequent. As a further work, we wonder about restricting these rules by imposing that the new formula α would have at most the same atomic formula occurrences as the hypothesis context Γ' of the left upper sequent and also that α would have a lesser complexity than $\Phi(\Gamma')$. Other future work consists on investigating what class of language could be described by the system **SL**, verifying if this approach of adopting combining permissions actually make it possible to generate a grammar that could recognize some class or sub-class of the context-sensitive languages, instead of context-free languages as **NL** and **L** [11, 12, 16].

Acknowledgments. We are grateful to Prof. P. Schroeder-Heister, Michael Arndt and Ernst Zimmermann for their helpful comments and suggestions. We also would like to thank the anonymous referees for their fair criticisms.

References

1. Abrusci, V.M.: *A Comparison between Lambek Syntactic Calculus and Intuitionistic Linear Propositional Logic*. Zeitschr. f. Math. Logik und Grundlagen d. Math. 36 (1990)11-15.
2. Benthem, J. van. *The semantics of variety in categorial grammar*. Report 83-29, Simon Fraser University, Burnaby (B.C) Canada (1983). (Revised version in [3]).
3. Buszkowski, W., Marciszewski, W and Benthem, J. van (eds): *Categorial Grammar*. John Benjamins. Amsterdam (1988).
4. Corrêa, M. Da S. & Haeusler, E. H.: *A concrete categorial model for the Lambek Calculus*. Mathematical Logic Quarterly (Formerly Zeitschr. f. Math. Logik und Grundlagen d. Math.) 43 (1997) 49-59.

5. Corrêa, M. Da S. & Haeusler, E. H.: *Selective Lambek Syntactic Calculus*. Proceedings of the 5th Workshop on Logic, Language, Information and Computation. São Paulo, July (1998) 39-46. Abstract appeared in the Conference Report published in Vol 6, N. 6 (Nov. 1998) of the Logic Journal of the IGPL.
6. Došen, K.: *Sequent systems and groupoid models*. Studia Logica 47, 353-385, 1988, Studia Logica 48 (1989) 41-65.
7. Dunn, J.M, Meyer, R. K.: *Combinators and structurally free logic*. Logic Journal of the IGPL Vol. 5 No. 4 (1997) 505-537.
8. Heple, M. *A general framework for hybrid substructural categorial logics*. Ms.IRCS Penn, Available as IRCS Report 94-14.
9. Lambek, J.: *The Mathematics of Sentence Structure*. American Math. Monthly 65 (1958) 154-169.
10. Lambek, J.: *On the calculus of syntactic types*. In: Structure of Language and its Mathematical Aspects. Proceedings of 12th Symp. Appl.Math., American Math. Soc., Providence, R.I. (1961) 166-178.
11. Kandulski, M.: *The equivalence of nonassociative Lambek Categorical Grammars and context-free grammars*. Zeitschrift fr Mathematische Logik und Grundlagen der Mathematik 34 (1988) 41-52.
12. Kandulski, M.: *On commutative and nonassociative syntactic calculi and categorial grammars*. Mathematical Logic Quarterly 65 (1995) 217-235.
13. Moortgat, M.: *Multimodal linguistic inference*. Journal of Logic, Language and Information 5 (1996) 349-385.
14. Moortgat, M.: *Categorial Type Logics*. In: J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*. North Holland Elsevier. Amsterdam (1997) chapter 2.
15. Morrill, G.: *Type Logical Grammar. Categorial Logic of Signs*. Kuwer, Dordrecht (1994).
16. Pentus, M.: *Lambek grammars are context-free*. In Proceedings of 8th Annual IEEE Symposium on Logic in Computer Science, New York:IEEE (1993).
17. Takeuti, G.: *Proof Theory*. North Holland. Amsterdam (1975).

Grammatical Development with XMG

Benoit Crabb

Loria, B.P.239, F-54506 Vandoeuvre-ls-Nancy, France
crabbe@loria.fr

Abstract. This paper is dedicated to the compact representation of Tree Adjoining Grammars. We provide a methodology for grammatical development with eXtensible MetaGrammar (XMG). The provided methodology has been set up together with the development of a large French TAG. Furthermore the grammatical representation language and the assorted development methodology presented can be reused for grammatical development with other strongly lexicalised syntactic formalisms.

XMG is an interpreter for a grammatical representation language designed in the spirit of PATR II. It has been specifically designed to ease the practical development of strongly lexicalised grammars. A first presentation of the language is given by [1] and a description of the assorted interpreter is given by [2].

Here, we are concerned with providing a grammatical development methodology for Tree Adjoining Grammars (TAG [3]) using XMG. This methodology has been applied to the actual development of an open source French TAG. We begin by motivating the language used: section 1 recalls the motivation of lexicalised syntactic formalisms and section 2 indicates the kind of generalisations we want to capture in grammatical descriptions. Section 3 introduces the actual language used for grammatical description. Section 4 provides a methodology for large grammatical development with that language. Finally Section 5 draws a short comparison with related proposals.

1 Motivation: Generalisations in the Lexicon

TAG is used in its lexicalised version (LTAG) for the representation of natural languages. In a lexicalised TAG, each grammatical unit is an elementary tree anchored by at least one word, the anchor. An anchor is a leaf node of an elementary tree.

Though strong lexicalism has well known advantages: on linguistic grounds, it allows to account for lexical exceptions or multiword expressions; on computational grounds it allow to select the only subset of the grammar required to parse a given sentence, thus improving parsing efficiency.

However in practice significant problems come from the lack of generalisations in grammatical representation. This lack of generalisation entails difficulties for

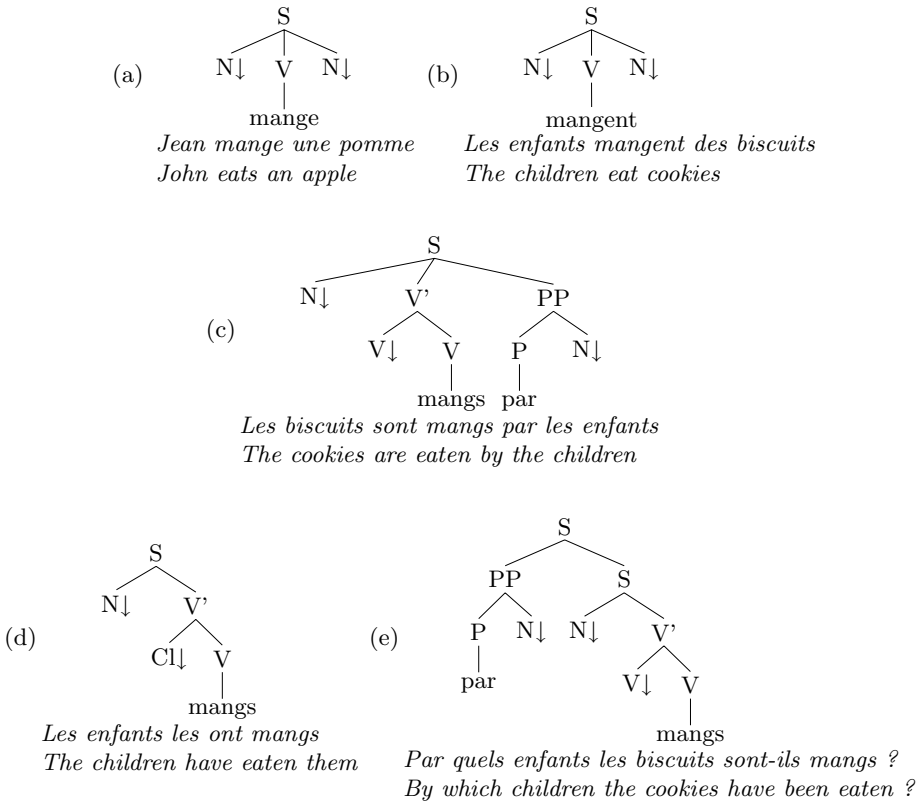


Fig. 1. Some alternative entries for transitive *manger* (to eat)

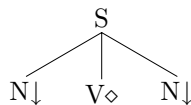
grammatical development and grammatical maintenance. Any substantial modification in a large sized grammar such as the representation of subject-verb agreement requires to modify an important number of independently described units.

Strictly speaking, in a lexicalised context, the lexicon for a tree adjoining grammar is an enumeration of independently described trees. In Figure 1 we illustrate this by providing some sample entries for the transitive French verb *manger* (to eat). Each sample tree is glossed with a sample sentence illustrating the context represented by that tree¹. These trees represent some sample alternative contexts in which *manger* can occur in French. Trees (a, b) illustrate the need for two (or more) different elementary trees for handling morphological alternatives, (c) illustrates a diathesis alternate realisation (passive) of the predicate and finally (d,e) illustrate alternative realisations of the predicate's

¹ In the grammar we have implemented there are actually 153 trees related to the transitive verb *manger*.

arguments : cliticisation of the object (d) and a questioned realisation of the By Object of the passive variant (e).

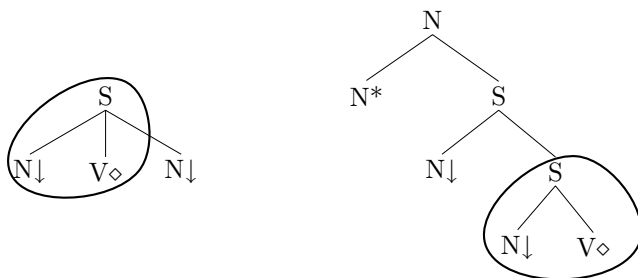
In the remaining of the paper we will consider only tree schemata as the units of interest, as it is usually the case in TAG implementations (see e.g. [4]). A tree schema is an elementary tree where the lexical item (the anchor) is left underspecified (we use the \diamond notation to indicate this). Actual elementary trees are generated on the fly by the parser. Such tree schemata are grammatical units among which we can identify generalisations. Using these generalisations, we shall show how to take advantage of them to ease the development of a computational grammar.



2 Two Kinds of Grammatical Generalisations

Methodologically we want to capture two kinds of generalisations : structure sharing on one hand and alternatives on the second hand.

Structure Sharing. Structure sharing is usually well understood. These generalisations aim at factoring out the common structures. The two following trees illustrate this:

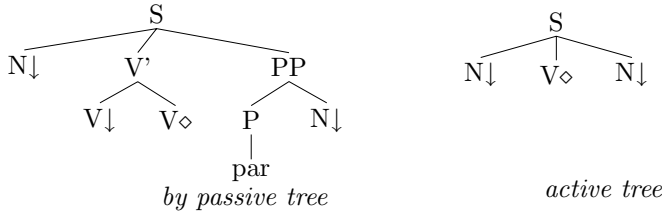


Jean mange une pomme
John eats an apple

La pomme que Jean mange
The apple John eats

Where the encircled subtrees represent the realisation of a subject, which is a common substructure shared by both units.

Alternatives. Alternatives is a specificity of the lexicon which has been sometimes ignored or left unclear in recent TAG literature [5, 6]. The active/passive alternative is an instance of such generalisations. Historically, the expression of such alternatives in the lexicon is initially due to [7] who argues in favor of the expression of a counterpart to some transformations of the Generative Grammar in the lexicon. In a language of grammatical representation we have to capture the fact that the two following trees are active/passive alternatives of a same predicate argument structure without focusing whether these trees actually share some common substructure.



The two methodological requirements we put forward are reflected in lexical rule based systems. [8] has in some respects ported the proposal of [9] to TAG. He uses an inheritance hierarchy to express structure sharing and lexical rules (called metarules) to express alternatives.

It is worth noting that alternatives have a special status because these generalisations contribute to describe *sets* of related grammatical units (such as an active-passive alternative). For instance an `ACTIVETOPASSIVE` lexical rule outputs a new passive tree provided a base active tree, which makes a set of two trees. Alternatives are an important point in grammatical representation: indeed a tree family in a TAG is a set of trees that represent alternative realisations of a given predicate argument structure.

3 The Language and Its Intuition

The language of grammatical description presented in this section aims at taking advantage of the two kinds of generalisations we have identified to ease grammatical development². Its original motivation is to provide an alternative grammatical representation language to a lexical rule based one. This motivation comes from the fact that we do not want to use lexical rules in grammatical development for TAG in order to avoid rule sequencing problems. The sequencing problems are increased in TAG since the number of lexical rules required is more much than in a phrase structure grammar lexical description. Indeed, since TAG has no independent context-free rules, their counterparts are expressed in the lexicon.

The language crucially relies on the notion of *class* (or template). A class is a *description* of partial grammatical structures. With TAG such a description is a tree description possibly augmented with feature structures³. Using classes allows giving a name to such a description that can be reused to stand for this description in any grammatical description. These classes allow us to capture grammatical generalisations.

The descriptions that can be expressed in a class follow the abstract syntax given here:

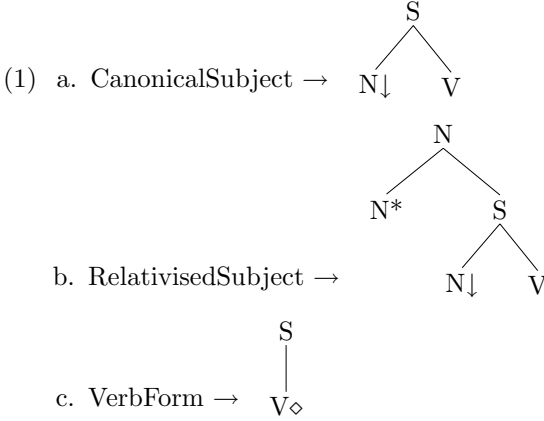
$$\begin{array}{lcl}
 \textit{Class} & ::= & \textit{Name} \quad \rightarrow \quad \textit{Goal} \\
 \textit{Goal} & ::= & \phi \quad | \quad \textit{Name} \quad | \quad \textit{Goal} \vee \textit{Goal} \quad | \quad \textit{Goal} \wedge \textit{Goal}
 \end{array}$$

² A formal presentation of the language is detailed in [1].

³ We do not detail the use of feature structures in this paper.

Essentially, a class associates a name to a description, called the Goal. The Goal is one of the following:

- An actual description of a grammatical structure (ϕ). Using TAG, we associate a name to a tree description. The following classes exemplify this:



The tree descriptions are expressed in a common tree description language detailed by [1]. We use common graphical notations to represent the formulae of the tree description language⁴. A tree description is interpreted as a finite first order tree.

- A name of a class otherwise defined. Thus in the following description, the class ActiveVerbForm reuses the description associated with the class VerbForm:

(2) ActiveVerbForm \rightarrow VerbForm
- A disjunction or choice of descriptions. The following class associates the name Subject with an alternative: a subject is either (in this example) a canonical or a relativised subject

(3) Subject \rightarrow CanonicalSubject \vee RelativisedSubject
- A conjunction of descriptions. Finally the language allows us to express conjunctions of descriptions. A conjunction of description is understood as the conjunction of two tree descriptions where node names of each conjunct are renamed.

(4) IntransitiveVerb \rightarrow Subject \wedge ActiveVerbForm

Interpretation of the Language. This grammatical description language is interpreted by [1] as a logic program of the DCG paradigm [10] where the terminals of the language are tree descriptions and where the conjunction of descriptions

⁴ Straight lines indicate immediate dominance. Linear precedence is indicated by the symbol \prec^+ and adjacency or immediate precedence is not indicated by any special symbol. Trees are decorated with node labels. We leave implicit node names who play no significative role in our discussion.

is a counterpart for concatenation. The language of the DCG is a tree language, that of the grammatical units of the grammar. We further require the number of grammatical units to be a finite set. Therefore the DCG is restricted to be non recursive (directly or indirectly). Given an axiom, an interpreter for this language generates the whole language of the grammar⁵.

In the context of our current example, given the axiom *IntransitiveVerb*, an interpreter outputs the results as depicted in Figure 2. This picture shows on the left the conjunction of descriptions used to output the models depicted on the right of the arrow. In other words the interpreter builds a *set* of trees, each of them is an alternative realisation of an *Intransitive* context.

As illustrated in figure 2, conjunction is regarded as a conjunction of tree descriptions yielding a new description whose class of admissible models is a class of finite first order trees, the minimal models [1].

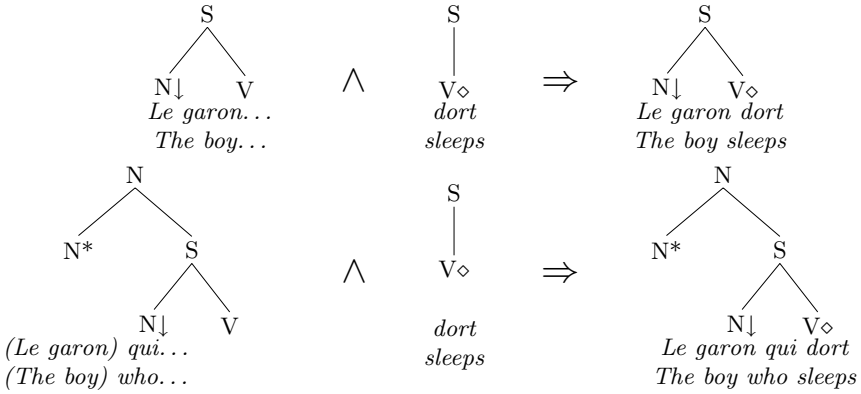


Fig. 2. Tree Generation for *IntransitiveVerb*

Inheritance Hierarchies. After [9], work on grammatical organisation often relies on inheritance hierarchies. Though this notion is not central to our language, we can easily interpret classes as being organized in inheritance hierarchies. We illustrate this with the following example

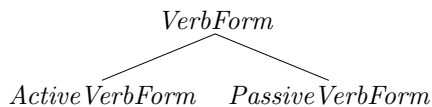
- (5) a. $\text{ActiveVerbForm} \rightarrow \text{VerbForm} \wedge \phi$
 b. $\text{PassiveVerbForm} \rightarrow \text{VerbForm} \wedge \psi$

where a class C_1 whose description uses a class named C_2 is interpreted as inheriting information from C_2 ⁶. Thus both *ActiveVerbForm* and *PassiveVerbForm*

⁵ [2] describes an actual implementation of such an interpreter. The interpretation of the grammatical description language as a logic program enables them to implement it following the model of a PROLOG interpreter, a Warren Abstract Machine [11].

⁶ ϕ and ψ stand for the additional description specified in the inheriting class which was not necessary to this paper.

are interpreted as inheriting information from *VerbForm*. That may be graphically represented as depicted above. It is worth being noticed that this inheritance notation should not be confused with HPSG ,



type-inheritance relations. The latter are a part of the HPSG theory where ours are just a graphical notation allowing to illustrate the organisation of classes and play much the same role as macros.

Name Spaces. Finally, we have also found it convenient to allow an explicit management of name spaces. Each class defines its own name space: each identifier (e.g. a node identifier in a tree description) is local to the class where it is declared. Nonetheless, in the context of inheritance, we have found it convenient to allow a given class to explicitly import the name space explicitly exported by its superclass.

4 Overall Methodology

In this section we provide the methodology we used for the development of the verbal part of a French TAG grammar. This methodology takes most of its inspiration from the three dimensional methodology given by [5, 6]. The description works by describing fragments of trees. A fragment represents either a realisation of an argument of the predicate or a realisation of the predicate itself. Intuitively the trees are described by combining one or more fragments representing arguments with a fragment representing the predicate form. This generalises the preliminary example given in section 3 where the classes *CanonicalSubject* and *RelativisedSubject* describe realisation of an argument and where the class *VerbForm* describes the realisation of the predicate itself.

We work with four levels of abstractions: the first aims at defining and organising classes describing tree fragments (Section 4.1). The second aims at grouping descriptions into syntactic functions (Section 4.2). The third aims at describing verbal diathesis alternatives (Section 4.3) while the fourth aims at capturing the notion of tree family (Section 4.4).

In the rest of this section, we illustrate our methodology by providing some sample classes that will allow us to generate some representative alternatives of a ditransitive family. Finally we finish this section by reporting the results of scaling up the given methodology in the context of development of a large French grammar.

4.1 Tree Fragments

Methodologically, this level of abstraction is only concerned with capturing structure sharing. The building blocks of the grammatical description are tree fragments. We mainly associate names to tree descriptions and organize them within

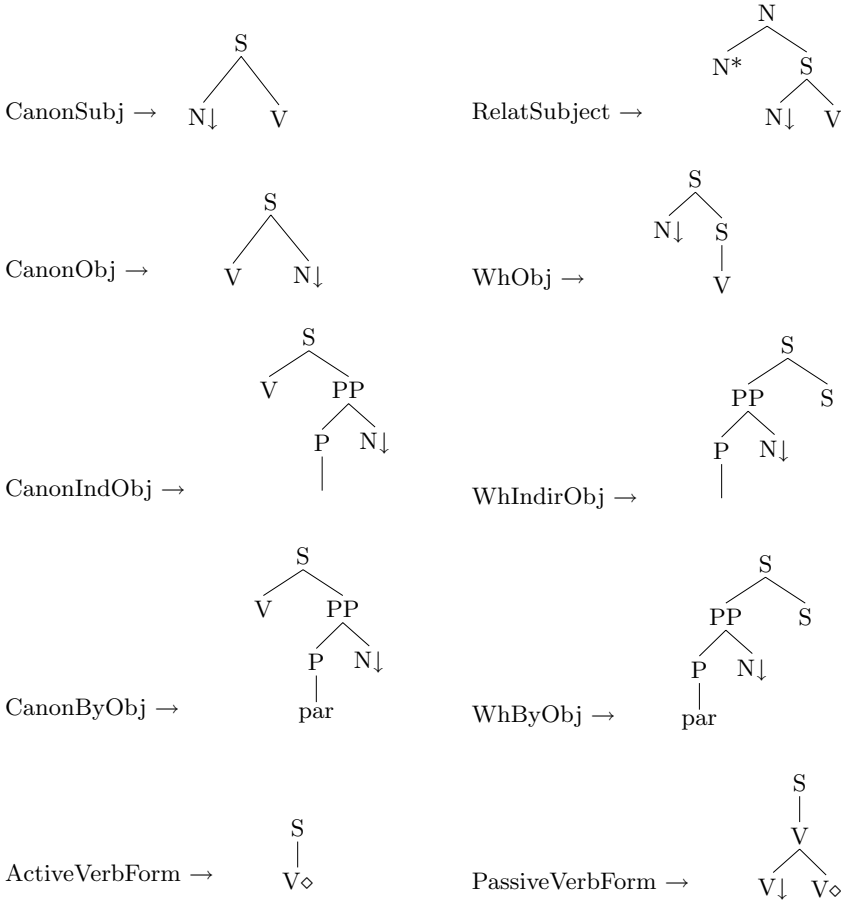


Fig. 3. Elementary tree fragments used as building blocks of the grammar

an inheritance hierarchy. Figure 3 provides sample classes describing tree fragments. These fragments represent different possible constructions of French verbal dependants in a TAG.

To further factorise information we organise the fragments in an inheritance hierarchy. Figure 4 provides a graphical representation of this hierarchy following the conventions introduced in section 3. This hierarchy illustrates that verbal arguments described in Figure 3 break in four categories. First, the canonical complements are those arguments realised after the verb. The canonical object is a noun and the prepositional complements are introduced by the prepositions for the canonical indirect object and *par* for the canonical by object. Second the canonical subject is a noun realised in front of the verb. Third, Wh arguments (or questioned arguments) are realised in front of a sentence headed by a verb and may possibly be realised at an unbounded distance of the predicate. Wh object is an extracted noun and questioned prepositional objects are extracted

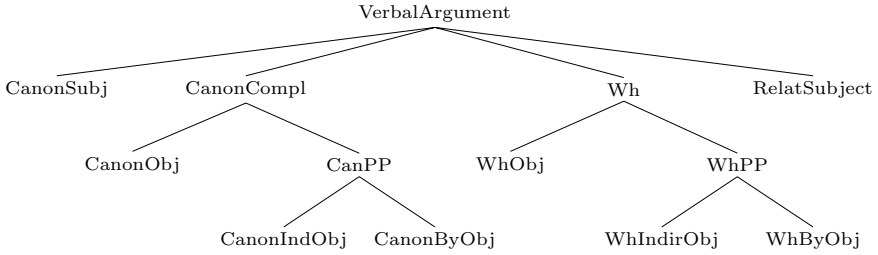


Fig. 4. Organisation of elementary fragments in an inheritance hierarchy

prepositional phrases that are introduced by a specific preposition. Fourth the relativised subject represents a relative pronoun realised in front of the sentence. Extracted subjects in French cannot be realised at an unbounded distance of the predicate.

Finally, classes in the hierarchy are specified in a way that each class has access to the identifiers declared by its immediate or non immediate superclasses.

4.2 Syntactic Functions

This second level of abstraction is mainly concerned with capturing alternatives. We want to capture generalisations that are syntactic functions. To do this, we take advantage of the tree fragments described in the previous section. We can reuse them by manipulating the name of the classes where they are defined.

Syntactic functions are understood here as notions that allow us to characterise verbal dependants by abstracting over the problem of word ordering. Along this line of thought, each syntactic function is a name associated to a set of alternative realisations in syntax as illustrated by the classes defined below:

- (6) a. $\text{Subject} \rightarrow \text{CanonicalSubject} \vee \text{RelatSubject}$
- b. $\text{Object} \rightarrow \text{CanonicalObject} \vee \text{WhObject}$
- c. $\text{ByObject} \rightarrow \text{CanonicalByObject} \vee \text{WhByObject}$
- d. $\text{IndirectObject} \rightarrow \text{CanonicalIndirObject} \vee \text{WhIndirObject}$

We define a subject as an abstraction for talking about a dependant that can be either realised in front of the verb in a canonical position (represented here by the class CanonicalSubject) or in front of the verb as a relative pronoun that cannot be realised at an unbounded distance from the predicate (class RelatSubject). Thus, the subject class we have provided here allows to characterise contexts such as these:

- (7) a. **Jean** mange (canonical subject)
- John** eats
- b. Le garon **qui** mange (relativised subject)
- The boy **who** eats

As a matter of illustration, the indirect object (class `IndirectObject`) is a function that abstracts over the realisation of an argument introduced by the preposition at the right of the verb (`CanonicalIndirObject`) or realised in extracted position (possibly realised at an unbounded distance from the predicate) as illustrated by the following examples:

- (8) a. Jean parle **Marie** (canonical indirect object)
 John talks **to Mary**
 b. **A qui** Jean parle-t-il ? (wh indirect object)
 To whom does John talk ?
 c. **A qui** Pierre croit-il que Jean parle ? (wh indirect object)
 To whom does Peter think that John talks ?

To sum up, the overall methodology we used for describing syntactic functions is compatible with the informal classification of French syntactic functions made by [12]. Each syntactic function is associated to a set of possible syntactic constructions. The set or system of syntactic functions for French is then defined in a way that there are no two syntactic functions associated with the same set of constructions.

4.3 Diathesis Alternations

In this third level, we take advantage of the abstractions captured so far to represent diathesis alternations. Again we are interested here in describing alternatives. Diathesis alternations are those alternations of mapping between arguments and syntactic functions, as for instance the active/passive alternation. In a diathesis alternation the actual form of the verb constrains the way arguments of the predicate are realised in syntax. Thus, in the following example,

- (9) a. Jean **envoie** une lettre
 John sends a letter
 b. Une lettre **est envoyée** par Jean
 A letter is sent by John

It is considered that both (9a) and (9b) are alternative realisations of a predicate argument structure such as *send(John, a letter)*. Diathesis alternations capture the fact that if the verb is at the active form, then the first predicative argument, *John*, behaves as a subject and the second predicative argument, *a letter*, behaves as an object. If the verb is in the passive form then the first predicative argument, *John*, behaves as a by Object (or agentive complement) and the second predicative argument, *a letter*, behaves as the subject. We can represent these facts in our language by defining the following class:

- (10) (TransitiveAlternation \rightarrow
 Subject \wedge ActiveVerbForm \wedge Object)
 \vee (Subject \wedge PassiveVerbForm \wedge ByObject)

Finally let us note that a traditional case of “erasing”, such as the agentless passive (or passive without agent) can be expressed in our language by adding an additional alternative where the By Object or agentive complement is not expressed. Thus (11) is an augmentation of (10) where we have added the agentless passive alternative (indicated in bold face).

- (11) TransitiveAlternation \rightarrow
 (Subject \wedge ActiveVerbForm \wedge Object)
 \vee (Subject \wedge PassiveVerbForm \wedge ByObject)
 \vee (**Subject** \wedge **PassiveVerbForm**)

This methodology can be further augmented to implement an actual linking along the lines of [13]. For the so-called erasing cases, one can map the “erased” predicative argument to an empty realisation in syntax. We refer the reader to [14] for further details.

4.4 Tree Families

Finally, we can capture tree families. In the TAG literature, a tree family is a set of trees that represent alternative realisations of a predicate argument structure. In the line of the example we have developped so far, we can easily describe a family of trees representing a ditransitive context. More precisely we can define a family where trees are variant realisations of a predicate argument structure with a nominal subject, a nominal object and an indirect nominal object as follows:

- (12) DitransitiveFamily \rightarrow TransitiveDiathesis \wedge Indirectobject

The trees generated for such a family will, among others, handle contexts such as these:

- (13) a. Jean offre des fleurs Marie
 John offers flowers to Mary
 b. A quelle fille Jean offre-t-il des fleurs ?
 To which girl does John offer flowers ?
 c. Le garon qui offre des fleurs Marie
 The boy who offers flowers to Mary
 d. Quelles fleurs le garon offre-t-il Marie ?
 Which flowers does the boy offer to Mary ?
 e. Les fleurs sont offertes par Jean Marie
 The flowers are offered by John to Mary
 f. Par quel garon les fleurs sont-elles offertes Marie ?
 By which boy the flowers are offered to Mary

All these sentences are seen as alternatives of the prototypical sentence (12). (12) illustrates an alternative (questioned) realisation of the Indirect Object, (12) and (12) respectively illustrate the realisation of a relativised subject and of a questioned object.(12) illustrates that the family handles passive alternatives thanks to the use of the TransitiveDiathesis class, and finally we show with

(12) that the agentive complement of the passive may be realised as well in a questioned position.

It is straightforward to extend the grammar by introducing new families. For instance, one can introduce a transitive family, that is a set of trees that are alternatives of predicate with a nominal subject and of a nominal object with the class definition (12) or an intransitive family (alternatives of a predicate with a nominal subject) with the definition (12).

- (14) a. `IntransitiveFamily` \rightarrow `Subject` \wedge `ActiveVerbForm`
 b. `TransitiveFamily` \rightarrow `TransitiveDiathesis`

Grammatical Generation. The last step of our presentation concerns the generation of an actual grammar with a given grammatical description. Recall that generating a grammar is understood as the generation of the set of words of the language of a grammar of the DCG paradigm, each of these words is a conjunction of tree descriptions licensed by the grammatical description.

To do this, we have to provide an axiom (or a goal) to the DCG. In practice we have found it convenient to allow the user to explicitly define the axioms he wants to be generated by the interpreter. Following the example developed so far, the user can ask the interpreter to generate the three families described before.

- (15) a. `value IntransitiveFamily`
 b. `value TransitiveFamily`
 c. `value DitransitiveFamily`

The interest of this explicit valuation relies on the possibility to generate either the total grammar being described by valuating every family or subgrammars by valuating only some of the families. These subgrammars can be used either for testing or checking the correctness of the resulting trees on smaller sets, or for generating grammars for applications where a large grammar is not needed.

Constraining the Models Generated. The units manipulated in grammatical description are tree descriptions. Tree descriptions are expressed in a language whose formulae are interpreted as finite trees, the minimal models (See [1] for further details). During grammatical development however we have found convenient to let the user further constrain the class of admissible models. This is achieved by using both properties and principles [1].

User definable properties may be attached to nodes in the tree descriptions. These properties may then be used as parameters of predefined principles constraining the generated models. We illustrate the use of properties and principles in the context of our running example. Until now according to our methodology, the families defined in (12) and (12) allow the generation of trees with multiple extractions. Although some rare cases of multiple extractions are known to

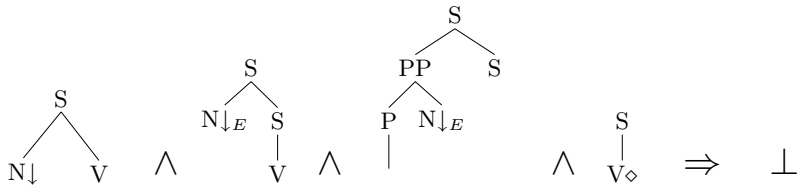


Fig. 5. Unicity of extraction

happen in French[15], it is generally preferable to rule them out of grammatical implementations.

To do this, we introduce a property *extracted* used as a parameter of a uniqueness principle. The uniqueness principle requires that in a valid model there may be at most one node associated to the property specified as parameter. To rule out double extraction we have associated that property to a node in fragments representing extraction and instantiated the principle of unicity with the parameter *extracted*.

Figure 5 illustrates this. The ditransitive family allows us to generate the conjunction of fragments illustrated. From left to right the fragments represent a canonical subject, a questioned object, a questioned indirect object and an active verb form. We have indicated the extracted property with the subscript *E*. No model can be generated given these descriptions since two different nodes cannot be associated with the property.

Beyond our working example, in the development of of a larger French grammar we also have used uniqueness for constraining French clitic rank uniqueness, and for constraining functional uniqueness. Besides uniqueness, we have used a coloration principle described in [1], a principle of ordering between sibling nodes in order to achieve French clitic ordering. And finally we have introduced an island principle that accounts for designing elementary trees that are compatible with the expression of islands constraints in TAG⁷.

Towards a Realistic French Grammar. It should be clear that the example we have given up to now is still quite simplified. The language and the methodology provided here have been successfully reused to extend the grammatical description for handling a significative amount of phenomena related to the syntax of French verbs and valency controlled verbal dependants (Table 1). Some extensions have been added to handle the syntax of French predicative adjectives. The actual grammar we have implemented is close to the one de-

⁷ In TAG, following [15, 4] setting an S node to substitution blocks extraction out of the constituent dominated by that S node, setting an S node to foot allows the extraction out of the dominated constituent. The island principle accounts for wh-islands: if an elementary tree contains both a node marked as extracted and a leaf node *n* being sentential, then the node *n* is marked as substitution, otherwise it is marked as foot.

Table 1. An overview of the coverage of the grammar (Valency controlled dependants)

Constructions	<i>Canonical, Clitic, Interrogative, Relative, Cleft</i>
Syntactic functions	<i>Subject, Object, Indirect object, Genitive, Locative, Obliques, Sentential subject, Sentential objects, Sentential interrogative</i>
Diathesis	<i>Active, Passive, Impersonal, Middle, Reflexive</i>
Subcategorisation	<i>44 subcategorisation frames</i>

scribed by [15, 5]. The actual grammar and some documentation for interfacing it with a lexicon is available in open source by anonymous CVS via the web site <http://sourcesup.cru.fr/XMG>. Future work requires to evaluate the grammar on a reference test suite such as TSNLP.

5 Some Practical Gains

Though similar to that of [5] and [6], the methodology provided here holds for a declarative and monotonic language such as the one provided by XMG. That is not the case in [5, 6]. The reason for this comes from the fact that we do not grant a particular status to a canonical or base tree from which other alternative trees would have been derived⁸. Instead the methodology provided consists of describing a whole family of trees being alternatives of each other, all of them with an equal status. To illustrate this, let us consider the case of the passive given in (11). Both [5, 6] had to introduce in their languages an erasing device, allowing to express that an initial subject argument is erased by a short passive rule. In our context that does not happen since there is no notion of initial nor final realisation of an argument.

Another difference with [5, 6] is related to the language used. Grammatical information is only expressed with conjunction and disjunction of tree fragments. The possibility to use disjunction in the language allows us to conveniently describe alternatives. As a consequence, it has allowed us to remove from the metagrammatical formalism a device central to the proposal of [5] namely “crossing”, whose methodological purpose was left unjustified. By comparison with [6], the use of disjunction allows us to express grammatical knowledge in a single language of grammatical description. We do not need to split the problem of grammatical representation in three different modules that are to be interfaced with each other in a given order⁹.

⁸ This idea, central to express alternatives in lexical rule based systems, remains implicit in both [5, 6].

⁹ In order to account for non-monotonic issues, [6] introduces in her system a central module of Lexical Redistribution Rules (LRR) that allows to express erasing and that justifies the tripartition of her system: the first and third modules handle information in a monotonic context.

In sum, the methodology provided here justifies the use of a monotonic language for the compact representation of TAG grammars. It shows that this language can be kept simple by using two devices for combining tree fragments: conjunction and disjunction. The result of this simplification results in a practical gain that has allowed us to significantly reduce the time needed for grammatical development.¹⁰

6 Conclusion and Perspectives

This paper provides a practical argument showing that a simple language made of abstractions representing descriptions and the ability to reuse them using conjunction and disjunction is enough for describing a large sized computational grammar. Alternatives are handled with disjunction instead of using lexical rules.

Regarding the TAG community, we have shown that the metagrammar [5, 6, 16] may be reduced to a simple grammatical description language by (1) explicitly introducing the possibility to express alternatives and (2) by providing a practical argument: that of developing a non trivial grammar with this language. The use of a different language has nevertheless allowed us to take advantage of the grammatical development methodology introduced by [5]. Furthermore, we have found that the language used here for grammatical development is strikingly similar to the one proposed by [17] except that the structures used in the grammar are different : LFG grammars use feature structures where TAG grammars use trees. This can ease comparisons between development methodologies in the two formalisms.

Finally, it can be shown [14] that the language and the grammatical development methodology is compatible with the effective expression of a syntax-semantics interface for TAGS along the lines of [18]. This can be done by adding parameters to the language classes. The effective implementation of a parser supporting semantics has been achieved by [19].

On formal grounds, further investigations are expected to extend the framework and the methodology to a larger family of syntactic formalisms. Preliminary investigations have been successfully led by G. Perrier who has reused the language and its assorted methodology in the development of a French Interaction Grammar[20] where the grammatical structures are D-Trees instead of trees. We are also interested in investigating how to reuse the language and the methodology to describe the lexicon of an eXtensible Dependency Grammar [21] where the elementary grammatical structures used are directed graphs.

On linguistic grounds, further investigations concern the development of alternative grammatical development strategies. Indeed the methodology presented here reflects a traditional — generative and syntactocentric — approach to syntax. An alternative method for implementing the syntax-semantics in-

¹⁰ By comparison with the metagrammar compiler described by [16] similar to that of [5] and for which we had also developed a grammar, we estimate that the development time is divided by ten.

terface would be to consider driving grammatical development by using lexical classes along the lines of [22]. These are thought to carry out a more semantically fine-grained approach to the syntax-semantics interface problem.

Acknowledgements. We are grateful to Guy Perrier and Jacqueline Lai for their comments and corrections on earlier drafts of this paper. We thank the three anonymous reviewers for their detailed comments who have helped to improve the content of the paper.

References

1. Crabb, B., Duchier, D.: Metagrammar redux. In: *Constraint Solving and Language Processing*, Copenhagen (2004)
2. Duchier, D., Leroux, J., Parmentier, Y.: The metagrammar compiler: An nlp application with a multi-paradigm architecture. In: *Mozart 2004, Charleroi* (2004)
3. Joshi, A.K., Schab, Y.: Tree adjoining grammars. In Rozenberg, G., Salomaa, A., eds.: *Handbook of Formal Languages*. Springer Verlag, Berlin (1997)
4. XTAG Research Group: A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania (2001)
5. Candito, M.H.: Organisation Modulaire et Paramtrable de Grammaires Electroniques Lexicalises. PhD thesis, Universit de Paris 7 (1999)
6. Xia, F.: Automatic Grammar Generation from two Different Perspectives. PhD thesis, University of Pennsylvania (2001)
7. Bresnan, J., Kaplan, R.M.: *The Mental Representation of Grammatical Relations*. The MIT Press, Cambridge MA (1982)
8. Becker, T.: HyTAG: A new Type of Tree Adjoining Grammars for Hybrid Syntactic Representation of Free Word Order Language. PhD thesis, Universitt des Saarlandes (1993)
9. Flickinger, D.: *Lexical Rules in the Hierarchical Lexicon*. PhD thesis, Stanford University (1987)
10. Pereira, F., Warren, D.: Definite clause grammars for language analysis —a survey of the formalism and a comparison to augmented transition networks. *Artificial Intelligence* **13** (1980) 231–278
11. Ait Kaci, H.: Warren’s abstract machine, a tutorial reconstruction. MIT Press (1991)
12. Iordanskaja, L., Mel’Čuk, I.: Towards establishing an inventory of surface-syntactic relations : Valency-controlled surface-syntactic dependents of the french verb. ((to appear))
13. Bresnan, J., Zaenen, A.: Deep unaccusativity in LFG. In Dziwirek, K., Farell, P., Mejias-Bikandi, E., eds.: *Grammatical Relations : A Cross-Theoretical Perspective*. CSLI, Stanford (1990) 45–57
14. Crabb, B.: Projection et monotonie dans un langage de representation lexico-grammatical. In: *Proc. TALN*. (2005) submitted.
15. Abeill, A.: *Une grammaire d’arbres adjoints pour le franais*. Editions du CNRS, Paris (2002)
16. Gaiiffe, B., Crabb, B., Roussanaly, A.: A new metagrammar compiler. In: *Proc. TAG+6, Venise* (2002)
17. Dalrymple, M., Kaplan, R., King, T.H.: Lexical structures as generalizations over descriptions. In: *LFG 04, Christchurch* (2004)

18. Gardent, C., Kallmeyer, L.: Semantic construction in feature-based tree adjoining grammar. In: 10th conference of the European Chapter of the Association for Computational Linguistics. (2003)
19. Gardent, C., Parmentier, Y.: Large scale semantic construction for tree adjoining grammar. In: Proceedings of LACL 2005, Bordeaux (2005)
20. Perrier, G.: Les grammaires d'interaction. Habilitation diriger les recherches en informatique (2003)
21. Debusmann, R., Duchier, D., Koller, A., Kuhlmann, M., Smolka, G., Thater, S.: A relational syntax-semantics interface based on dependency grammar. In: Proceedings of the COLING 2004 Conference, Geneva/SUI (2004)
22. Levin, B.: English Verb Classes and Alternations. The University of Chicago Press (1993)

Lambek-Calculus with General Elimination Rules and Continuation Semantics

Nissim Francez

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

1 Introduction

In modern approaches to proof-theory (e.g., [8]) natural-deduction (ND) proof systems are presented with *general elimination-rules (GE-rules)* [10], derived from a more general (re)formulation of Prawitz's *inversion principle* [13]. The setting of such enterprises are usually intuitionistic (and occasionally classic), but also *linear* [7] propositional calculi.

The aim of this paper is to investigate the effect of passing to GE-rules in the (associative) Lambek-calculus [5] **L**, the heart of Type-Logical Grammar formalisms [6]. The added dimensions here are the following.

- The *non-commutativity* (of both the structural comma, and the product, though we treat here the product-free subcalculus, more relevant to grammar theory).
- The *peripherality* requirement on discharge of assumptions, imposed on the implication-introduction rules in **L**.

These two issues pose certain questions about how should GE-rules for the *directed implications* (the two residuals of the non-commutative product) be formulated.

Since we are dealing with a *linear* subsystem, the Gentzen-style formulation of ND is used, with sequents manipulating assumptions explicitly. Thus, the GE-rule for Intuitionistic implication is formulated as follows.

$$\frac{\Gamma_1 \triangleright (A \rightarrow B) \quad \Gamma_2 \triangleright A \quad \Gamma_3 \triangleright C}{\Gamma_1 \Gamma_2 \Gamma_3 \ominus B \triangleright C} (\rightarrow GE)$$

Here ' $\Gamma \ominus B$ ' denotes removal of any number (including zero) of instances of B from Γ . The premisses of the rule are referred to as *major premiss*, *minor premiss* and *auxiliary premiss*, respectively (viewed from left to right). In presenting derivations, discharged assumption are put in square brackets and indexed, and every instances of $(\rightarrow I)$ is indexed (*uniquely!*) with an index of the assumption-instances it discharges.

However, since we are going to dwell in a linear context, in which neither vacuous nor multiple discharge of assumption is allowed, we may switch to the following notation, closer to the one used in **L**.

$$\frac{\Gamma_1 \triangleright (A \rightarrow B) \quad \Gamma_2 \triangleright A \quad [B]_i \Gamma_3 \triangleright C}{\Gamma_1 \Gamma_2 \Gamma_3 \triangleright C} (\rightarrow GE)_i$$

In an attempt to pass to **L** with GE-rules, the following two questions immediately arise, due to the centrality of *directionality* and *peripherality* (of assumptions):

- (1) What should be the peripherality of the discharged assumption B w.r.t Γ_3 in the auxiliary premiss? Such a question does not rise in regular implication-elimination of **L**, which do not discharge assumptions. Nor does it arise in linear logic (either commutative or non-commutative), which does not impose peripherality on discharged assumptions.
- (2) Where should Γ_3 be positioned in the conclusion, w.r.t Γ_1, Γ_2 (where the relative position of the assumptions of the major and minor premiss are dictated by the direction of the implication)?

The aim, of course, is to remain conservative and not add derivable sequents that are underivable in **L**. We will observe that for the non-commutative **L**, expressing ND-rules using sequents (with explicit manipulation of assumptions), is even more essential than for (merely) linear logic.

Another major aspect of general elimination rules for **L** is the corresponding *Curry-Howard (CH)* terms, that drive the meaning derivations in **L**-based grammars. Such terms, in an extension of the λ -calculus called ΛJ , were proposed in [4]. It uses terms of the form $M(N, z.P)$, referred to there as *generalized application* terms. Below is the $(\rightarrow GE)$ rule *with* the accompanying term assignment.

$$\frac{\Gamma_1 \triangleright (A \rightarrow B) : M \quad \Gamma_2 \triangleright A : N \quad [B]_i : z \quad \Gamma_3 \triangleright C : P}{\Gamma_1 \Gamma_2 \Gamma_3 \triangleright C : M(N, z.P)} (\rightarrow GE)_i$$

While having their own contractions, these terms can be directly embedded back into the λ -calculus by setting $M(N, z.P) =^{df} P[z := M(N)]$, where ‘:=’ denotes substitution (in terms). This embedding is carried here to **L** to express meanings. Both terms are used synonymously.

We use this CH-image of general elimination-rules to obtain an interesting effect within **L**-based grammars’ syntax-semantics interface. We get a *continuation-semantics* [1, 2] similar to the types of meanings in a CPS-style[11]. The general elimination-rules, so to speak, form a “syntactic continuation”, inducing the corresponding semantic continuation without any further stipulation, just by the CH-correspondence. Continuation semantics, originating from semantics of programming languages, was proposed as an independently motivated semantics, having several advantages in its explanatory power of various natural language constructs, such as: quantifier scope alternations, focus, coordination, misplaced modification and crossover. My point here is not to argue in favor of continuation semantics, but to point out that it falls out of the use of GE-rules in **L**, as the CH-counterpart, and need not be “invented”. This is exemplified by deriving quantifier scope alternations, as described in Section 3.

2 General Elimination Rules for **L**

2.1 Inversion Principles

Since we are using here the “Gentzen style” notation for ND, that emphasizes an *explicit* treatment of assumption manipulation, we reformulate Prawitz’s [13] inversion principle to relate explicitly to the assumptions used in the derivations counting as “sufficient ground” for introduction. This will give a handle of the incorporation of *order of assumptions* in the non-commutative setting. The boldface parts are my addition, for explicitness. To keep the notation simple, assume elimination-rules have a major premiss (depending on open assumptions Γ_2), and *one* minor premiss (depending on open assumptions Γ_1).

Let ρ be an application of an elimination-rule with consequence ψ **from open assumptions** $\Gamma = \Gamma_1 \cup \Gamma_2$. Then, the derivation justifying the introduction of the major premiss ϕ of ρ **from open assumptions** Γ_1 , together with the derivations of minor premisses of ρ **from open assumptions** Γ_1 , “contain” already a derivation of ψ **from the same** Γ , without the use of ρ .

Together with some sharpening of the principle in [14], the effect of the principle can be rendered via the following requirements [9] that the proof-system should satisfy (skipping the requirements about the use of open assumptions, for brevity).

Local Soundness: Introducing a connective followed by its immediate elimination should reduce to a derivation without such a “detour”.

Such a reduction is known as *conversion*. Failing local soundness means the elimination-rule is too strong, allowing to conclude more than “known” due to the defining introduction.

Local Completeness: An elimination-rule retains enough information to reconstruct the eliminated connective by an introduction-rule.

The transformation is known as *expansion*. Failing local completeness means the elimination-rule is too weak, not allowing to conclude everything “known” by the defining introduction.

The inversion-principle justifies the “standard” elimination-rules in various propositional calculi. The stronger inversion-principle, that leads to the general elimination-rule, replaces reconstructing a derivation of the ‘*direct grounds*’ justifying an introduction, by a reconstruction of *arbitrary consequences* of those ‘direct grounds’. Its formulation (leaving assumptions implicit) is

Whatever follows from the direct grounds for deriving proposition must follow from that proposition [8] (p.6).

The following lemma will turn to have a useful counterpart in the non-commutative **L** when used for grammars.

Lemma (derived rules): Let, for any $n \geq 1$,

$$\frac{\Gamma_1 \triangleright A_1 : M_1 \quad \dots \quad \Gamma_n \triangleright A_n : M_n}{\Gamma_1 \dots \Gamma_n \triangleright A : M} (R)$$

be a *derived rule* (in a calculus with regular elimination-rules). Then, its *generalization* $(GR)_i$ is a sound derived rule in the corresponding calculus with general elimination rules, discharging the assumption in the additional auxiliary premiss.

$$\frac{\Gamma_1 \triangleright A_1 : M_1 \quad \dots \quad \Gamma_n \triangleright A_n : M_n \quad [A]_i : u \quad \Gamma_{n+1} \triangleright C : P}{\Gamma_1 \dots \Gamma_n \Gamma_{n+1} \triangleright C : P[u := M]} (GR_i)$$

Proof: The derivation is shown below (up to a β -reduction).

$$\frac{\frac{\Gamma_1 \triangleright A_1 : M_1 \quad \dots \quad \Gamma_n \triangleright A_n : M_n}{\Gamma_1 \dots \Gamma_n \triangleright A : M} (R) \quad \frac{[A]_i : u \quad \Gamma_{n+1} \triangleright C : P}{\Gamma_{n+1} \triangleright (A \rightarrow C) : \lambda u. P} (\rightarrow I)_i}{\Gamma_1 \dots \Gamma_n \Gamma_{n+1} \triangleright C : P[u := M]} (\rightarrow E)$$

2.2 The Lambek Calculus and Its Elimination-Rules

Since we are interested primarily in **L**-based *grammars*, we interpret the propositional formulae as *syntactic categories*, formed by the closure under *directed implications*, ‘ \rightarrow ’, ‘ \leftarrow ’, of a finite set of \mathcal{B} of *primitive categories*. Categories are ranged over by \mathbf{c} , \mathbf{c}_i , and primitive categories by \mathbf{b} , \mathbf{b}_i . Sequences of categories are ranged over by Γ , Γ_i . We use ‘ \triangleright ’ for the sequent-separator. Our point of departure is the standard ND presentation of **L** (with “regular” elimination rules), presented in Figure 1. The proof-terms will be discussed later, when semantics is considered. The two notable characteristics of **L** are:

1. In the elimination-rules, the assumptions of the major and minor premisses are *ordered in the conclusion*, where the order is induced by the direction of the implication.
2. In the introduction-rules, the discharged assumption is *peripherally-located* in the open assumptions of the premiss, and the peripherality (leftmost or rightmost) determines the direction of the introduced implication in the conclusion.

$$(ax) \quad \mathbf{c} : x \triangleright \mathbf{c} : x$$

$$\frac{\Gamma_1 \triangleright \mathbf{c}_1 : N \quad \Gamma_2 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) : M}{\Gamma_1 \Gamma_2 \triangleright \mathbf{c}_2 : M(N)} (\rightarrow E) \quad \frac{\Gamma_2 \triangleright (\mathbf{c}_2 \leftarrow \mathbf{c}_1) : M \quad \Gamma_1 \triangleright \mathbf{c}_1 : N}{\Gamma_2 \Gamma_1 \triangleright \mathbf{c}_2 : M(N)} (\leftarrow E)$$

$$\frac{[\mathbf{c}_1]_i : x \quad \Gamma \triangleright \mathbf{c}_2 : M}{\Gamma \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) : \lambda x. M} (\rightarrow I_i) \quad \frac{\Gamma \quad [\mathbf{c}_1]_i : x \triangleright \mathbf{c}_2 : M}{\Gamma \triangleright (\mathbf{c}_2 \leftarrow \mathbf{c}_1) : \lambda x. M} (\leftarrow I_i), \quad \Gamma \text{ not empty}$$

Fig. 1. The **L**-calculus with regular elimination rules

Thus, the order of the open assumptions is part of the “sufficient grounds” for deducing a formula. An adequate inversion principle should relate explicitly to this order (forming the usual difficulties in first-order expression of the requirement). We end up with the following formulation of the inversion-principle:

Let ρ be an application of an elimination-rule with consequence ψ **from ordered open assumptions** Γ_1, Γ_2 . Then, the derivation justifying the introduction of the major premiss ϕ of ρ **from open assumptions** Γ_2 , together with the derivations of minor premisses of ρ **from respective open assumptions** Γ_1 , “contain” already a derivation of ψ **from the same** Γ_i , $i = 1, 2$, **ordered in the same way**, without the use of ρ .

The additional requirement, that of preservation of the order of open assumptions in the “contained proof”, explains the (regular) elimination-rules in Figure 1. For example, when ‘ \rightarrow ’ is introduced in $(\mathbf{c}_1 \rightarrow \mathbf{c}_2)$ by $(\rightarrow I)$, an open assumption \mathbf{c}_1 was placed *left-peripherally* to Γ_2 in deriving \mathbf{c}_2 . Thus, if the introduced formula is immediately eliminated with a minor premiss $\Gamma_1 \triangleright \mathbf{c}_1$, Γ_1 has to be positioned *left to* Γ_2 in the reconstructed derivation (without the “detour”). Had any other order of Γ_1, Γ_2 been used in the introduction-rules, local completeness would not hold. Suppose we consider reversing the order. The expansion (for the right implication) under the correct ordering

$$\Gamma \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) \implies \frac{\frac{\Gamma \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) \quad [\mathbf{c}_1 \triangleright \mathbf{c}_1]_i}{\mathbf{c}_1 \Gamma \triangleright \mathbf{c}_2} (\rightarrow E)}{\Gamma \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2)} (\rightarrow I_i)$$

is blocked under the inverse, wrong, ordering

$$\frac{\Gamma \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) \quad [\mathbf{c}_1 \triangleright \mathbf{c}_1]_i}{\frac{\Gamma \mathbf{c}_1 \triangleright \mathbf{c}_2}{???} (\rightarrow I_i ?)} (\rightarrow E ?)$$

(while the left implication might have been introduced, wrongly).

A conclusion of the final formulation of the inversion-principle is, that when an assumption-discharging rule is involved in a conversion, the open assumptions used to derive a conclusion “replacing” the discharged assumption in the reconstructed derivation are positioned where the discharged assumption was positioned originally.

This conclusion underlies the following *proof-substitution* rule for non-commutative logic, *admissible* for **L**, explaining the order of open-assumptions in the conclusion of implication-elimination rules.

$$\frac{\Gamma_1 \triangleright \mathbf{c}_1 \quad \Gamma_2 \mathbf{c}_1 \Gamma_3 \triangleright \mathbf{c}_2}{\Gamma_2 \Gamma_1 \Gamma_3 \triangleright \mathbf{c}_2} (Sub)$$

While the rule is admissible for \mathbf{L} , in its general form it *is not derivable* in \mathbf{L} . However, its two *peripheral-substitution* special cases, with either Γ_2 or Γ_3 empty, *are* derivable.

$$\frac{\Gamma_1 \triangleright \mathbf{c}_1 \quad \mathbf{c}_1 \Gamma_3 \triangleright \mathbf{c}_2}{\Gamma_1 \Gamma_3 \triangleright \mathbf{c}_2} (Sub^l) \quad \frac{\Gamma_1 \triangleright \mathbf{c}_1 \quad \Gamma_2 \mathbf{c}_1 \triangleright \mathbf{c}_2}{\Gamma_2 \Gamma_1 \triangleright \mathbf{c}_2} (Sub^r)$$

The derivation of (Sub^l) is

$$\frac{\Gamma_1 \triangleright \mathbf{c}_1 \quad \frac{\mathbf{c}_1 \Gamma_3 \triangleright \mathbf{c}_2}{\Gamma_3 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2)} (\rightarrow I)}{\Gamma_1 \Gamma_3 \triangleright \mathbf{c}_2} (\rightarrow E)$$

Interestingly, this explanation underlies also the original *product-elimination* rule $(\bullet E)$ in (the non-associative) \mathbf{L} , which has already the form of a general elimination-rule (similar to disjunction-elimination for Intuitionistic logic):

$$\frac{\Gamma_2 \triangleright \mathbf{c}_1 \bullet \mathbf{c}_2 \quad \Gamma_1[\mathbf{c}_1][\mathbf{c}_2]\Gamma_3 \triangleright \mathbf{c}}{\Gamma_1 \Gamma_2 \Gamma_3 \triangleright \mathbf{c}} (\bullet E)$$

where Γ_2 is placed *between* Γ_1 and Γ_3 , the original placement of the discharged assumptions $\mathbf{c}_1 \mathbf{c}_2$ in the minor premiss. Note that no peripherality is involved here.

While passing, we note that a similar explanation applies in [12] to the ordering of assumptions in the “ordered compartment” of the context; there, a context has three compartments: intuitionistic (unrestricted), linear (admitting only the exchange structural rule), and ordered (like here, admitting no structural rule).

We now turn to general implication-elimination rules. We assume here, in the spirit of \mathbf{L} , that in the auxiliary premiss too, the discharged assumption has to be *peripheral* to the sequence of open assumptions of the premiss. This is in-line with the derivability of the peripheral-substitution rules in \mathbf{L} . We focus on right-implication, as left-implication is symmetric. Apriori, there are four possibilities of a general elimination rule for ‘ \rightarrow ’, by considering both possibilities for the two issues involved: the peripherality of the discharged assumption in the auxiliary premiss, and the relative placement of the assumptions of the auxiliary premiss. Note that here it becomes evident the sequent-presentation of ND-rules is essential in facilitating the explicit expression of peripherality/directionality of

$[c_1]$

\vdots

the discharged assumption in the auxiliary premiss. The notation \vec{c}_3 is just not expressive enough, not showing the peripherality of $[c_1]$ w.r.t the other open assumptions. In all of the potential general-elimination rules, the relative ordering of Γ_1 , Γ_2 , the premises of the major and minor premisses, is assumed to be the same as in the regular elimination-rules, otherwise the latter do not become special cases of the former. The potential rules, referred to as $(\rightarrow GE_i^j)$, $j = 1, \dots, 4$,

$$\begin{array}{c}
\frac{\Gamma_1 \triangleright \mathbf{c}_1 \quad \Gamma_2 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) \quad [\mathbf{c}_2]_i \Gamma_3 \triangleright \mathbf{c}_3}{\Gamma_3 \Gamma_1 \Gamma_2 \triangleright \mathbf{c}_3} (\rightarrow GE)_i^1 \quad \frac{\Gamma_1 \triangleright \mathbf{c}_1 \quad \Gamma_2 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) \quad [\mathbf{c}_2]_i \Gamma_3 \triangleright \mathbf{c}_3}{\Gamma_1 \Gamma_2 \Gamma_3 \triangleright \mathbf{c}_3} (\rightarrow GE)_i^2 \\
\frac{\Gamma_1 \triangleright \mathbf{c}_1 \quad \Gamma_2 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) \quad \Gamma_3 [\mathbf{c}_2]_i \triangleright \mathbf{c}_3}{\Gamma_3 \Gamma_1 \Gamma_2 \triangleright \mathbf{c}_3} (\rightarrow GE)_i^3 \quad \frac{\Gamma_1 \triangleright \mathbf{c}_1 \quad \Gamma_2 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) \quad \Gamma_3 [\mathbf{c}_2]_i \triangleright \mathbf{c}_3}{\Gamma_1 \Gamma_2 \Gamma_3 \triangleright \mathbf{c}_3} (\rightarrow GE)_i^4
\end{array}$$

Fig. 2. Potential general elimination-rules for right implication

are shown in Figure 2 (omitting the CH-terms). Of those four rules, only two, $(\rightarrow GE_i^2)$, $(\rightarrow GE_i^3)$ are sound, and derivable using the regular elimination-rules, as shown in the following derivations. First, assume Γ_3 is not empty.

$$\begin{array}{c}
\frac{\Gamma_1 \triangleright \mathbf{c}_1 : N \quad \Gamma_2 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) : M}{\Gamma_1 \Gamma_2 \triangleright \mathbf{c}_2 : M(N)} (\rightarrow E) \quad \frac{[\mathbf{c}_2]_i : z \quad \Gamma_3 \triangleright \mathbf{c}_3 : P}{\Gamma_3 \triangleright (\mathbf{c}_2 \rightarrow \mathbf{c}_3) : \lambda z.P} (\rightarrow I_i) \\
\frac{\Gamma_1 \Gamma_2 \triangleright \mathbf{c}_2 : M(N) \quad \Gamma_3 \triangleright (\mathbf{c}_2 \rightarrow \mathbf{c}_3) : \lambda z.P}{\Gamma_1 \Gamma_2 \Gamma_3 \triangleright \mathbf{c}_3 : P[z := M(N)]} (\rightarrow E) \\
\\
\frac{\Gamma_1 \triangleright \mathbf{c}_1 : N \quad \Gamma_2 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) : M}{\Gamma_1 \Gamma_2 \triangleright \mathbf{c}_2 : M(N)} (\rightarrow E) \quad \frac{\Gamma_3 [\mathbf{c}_2]_i : z \triangleright \mathbf{c}_3 : P}{\Gamma_3 \triangleright (\mathbf{c}_3 \leftarrow \mathbf{c}_2) : \lambda z.P} (\leftarrow I_i) \\
\frac{\Gamma_1 \Gamma_2 \triangleright \mathbf{c}_2 : M(N) \quad \Gamma_3 \triangleright (\mathbf{c}_3 \leftarrow \mathbf{c}_2) : \lambda z.P}{\Gamma_3 \Gamma_1 \Gamma_2 \triangleright \mathbf{c}_3 : P[z := M(N)]} (\leftarrow E)
\end{array}$$

The derivations of the other two are blocked due to the mismatch of the peripherality of \mathbf{c}_2 in the auxiliary premiss and the relative position of Γ_3 in the conclusion.

For the case where Γ_3 is empty, we must have $\mathbf{c}_2 = \mathbf{c}_3$, the auxiliary premiss is an axiom and can be eliminated from the rule, obtaining the regular¹ elimination-rules as special cases.

Thus, in continuation to the conclusion of the inversion-principle as stated above, the emerging generalization seems to be the following:

- The peripherality of the discharged assumption relative to the open assumptions in the auxiliary premiss is *free*: either left or right.
- The position of the open assumptions of the auxiliary premiss in the conclusion is *determined* by the position of the discharged assumption in the natural way.

The need for *two* general elimination-rules per directed implication is not surprising, as there are two possibilities as to the way an arbitrary consequence can follow from the direct grounds of of introducing a directed implication, not determined by the direction of the introduced implication.

From now on, we refer to the (correct) general elimination-rules as $(\rightarrow GE_i^l)$, $(\rightarrow GE_i^r)$, $(\leftarrow GE_i^l)$, $(\leftarrow GE_i^r)$, with i the index of the assumption discharged

¹ In this case, the directionality/peripherality of \mathbf{c}_1 w.r.t the empty Γ_3 is immaterial.

$$\begin{array}{c}
\frac{\Gamma_1 \triangleright \mathbf{c}_1 : N \quad \Gamma_2 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) : M \quad [\mathbf{c}_2]_i : z \quad \Gamma_3 \triangleright \mathbf{c}_3 : P}{\Gamma_1 \Gamma_2 \Gamma_3 \triangleright \mathbf{c}_3 : M(N, z.P)} (\rightarrow GE_i^l) \\
\\
\frac{\Gamma_1 \triangleright \mathbf{c}_1 : N \quad \Gamma_2 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) : M \quad \Gamma_3 [\mathbf{c}_2]_i : z \triangleright \mathbf{c}_3 : P}{\Gamma_3 \Gamma_1 \Gamma_2 \triangleright \mathbf{c}_3 : M(N, z.P)} (\rightarrow GE_i^r) \\
\\
\frac{\Gamma_1 \triangleright \mathbf{c}_1 : N \quad \Gamma_2 \triangleright (\mathbf{c}_2 \leftarrow \mathbf{c}_1) : M \quad [\mathbf{c}_2]_i : z \quad \Gamma_3 \triangleright \mathbf{c}_3 : P}{\Gamma_2 \Gamma_1 \Gamma_3 \triangleright \mathbf{c}_3 : M(N, z.P)} (\leftarrow GE_i^l) \\
\\
\frac{\Gamma_1 \triangleright \mathbf{c}_1 : N \quad \Gamma_2 \triangleright (\mathbf{c}_2 \leftarrow \mathbf{c}_1) : M \quad \Gamma_3 [\mathbf{c}_2]_i : z \triangleright \mathbf{c}_3 : P}{\Gamma_3 \Gamma_2 \Gamma_1 \triangleright \mathbf{c}_3 : M(N, z.P)} (\leftarrow GE_i^r)
\end{array}$$

Fig. 3. The (revised) associative Lambek-calculus with general elimination-rules

by the rule. For empty Γ , the directional superscript (l or r) is immaterial and omitted in derivations. The revised **L** is presented in Figure 3. The axioms and introduction-rules remain intact and are suppressed in the figure. Next, we present the non-commutative counterpart of the derived-rules lemma.

Lemma (non-commutative derived rules): Let, for any $n \geq 1$ and i_1, \dots, i_n a permutation of $1, \dots, n$,

$$\frac{\Gamma_1 \triangleright \mathbf{c}_1 : M_1 \quad \dots \quad \Gamma_n \triangleright \mathbf{c}_n : M_n}{\Gamma_{i_1} \dots \Gamma_{i_n} \triangleright \mathbf{c} : M} (R)$$

be a *derived rule* in **L**. Then, for Γ_{n+1} not empty, its *generalizations* $(GR^l)_i, (GR^r)_i$ are sound derived rules in **L** with general elimination rules, discharging the assumption in the additional auxiliary premiss.

$$\begin{array}{c}
\frac{\Gamma_1 \triangleright \mathbf{c}_1 : M_1 \quad \dots \quad \Gamma_n \triangleright \mathbf{c}_n : M_n \quad [\mathbf{c}]_i : u \quad \Gamma_{n+1} \triangleright \mathbf{c}^* : P}{\Gamma_{i_1} \dots \Gamma_{i_n} \Gamma_{n+1} \triangleright \mathbf{c}^* : P[u := M]} (GR^l)_i \\
\\
\frac{\Gamma_1 \triangleright \mathbf{c}_1 : M_1 \quad \dots \quad \Gamma_n \triangleright \mathbf{c}_n : M_n \quad \Gamma_{n+1} [\mathbf{c}]_i : u \triangleright \mathbf{c}^* : P}{\Gamma_{n+1} \Gamma_{i_1} \dots \Gamma_{i_n} \triangleright \mathbf{c}^* : P[u := M]} (GR^r)_i
\end{array}$$

The proof is analogous to the commutative case, using the introduction of the corresponding directed implications. Note that the relative order of $\Gamma_{i_1} \dots \Gamma_{i_n}$ remains fixed, and the relative order of Γ_{n+1} w.r.t the whole permutation is determined by the peripherality of the assumption in the auxiliary premiss, just as in the general implication-elimination rules themselves.

As an example, consider the *harmonious composition* rules, which are derived in **L** (and are primitive rules in CCG [15]).

$$\frac{\Gamma_1 \triangleright (\mathbf{c}_3 \leftarrow \mathbf{c}_2) : F_2 \quad \Gamma_2 \triangleright (\mathbf{c}_2 \leftarrow \mathbf{c}_1) : F_1}{\Gamma_1 \Gamma_2 \triangleright (\mathbf{c}_3 \leftarrow \mathbf{c}_1) : \lambda z.F_2(F_1(z))} (FC),$$

$$\frac{\Gamma_1 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_2) : F_2 \quad \Gamma_2 \triangleright (\mathbf{c}_2 \rightarrow \mathbf{c}_3) : F_1}{\Gamma_1 \Gamma_2 \triangleright (\mathbf{c}_1 \rightarrow \mathbf{c}_3) : \lambda z. F_2(F_1(z))} (BC)$$

For (FC) we get the following (harmonious) generalized composition rules (and the symmetric ones for (BC)).

$$\frac{\Gamma_1 \triangleright (\mathbf{c}_3 \leftarrow \mathbf{c}_2) : F_2 \quad \Gamma_2 \triangleright (\mathbf{c}_2 \leftarrow \mathbf{c}_1) : F_1 \quad [(\mathbf{c}_3 \leftarrow \mathbf{c}_1)]_i : F \quad \Gamma_3 \triangleright \mathbf{c} : M}{\Gamma_1 \Gamma_2 \Gamma_3 \triangleright \mathbf{c} : M[F := \lambda z. F_2(F_1(z))]} (GFC^l)_i$$

$$\frac{\Gamma_1 \triangleright (\mathbf{c}_3 \leftarrow \mathbf{c}_2) : F_2 \quad \Gamma_2 \triangleright (\mathbf{c}_2 \leftarrow \mathbf{c}_1) : F_1 \quad \Gamma_3 [(\mathbf{c}_3 \leftarrow \mathbf{c}_1)]_i : F \triangleright \mathbf{c} : M}{\Gamma_3 \Gamma_1 \Gamma_2 \triangleright \mathbf{c} : M[F := \lambda z. F_2(F_1(z))]} (GFC^r)_i$$

3 The Effect of General Elimination-Rules on L-Based Grammars

First, recall that an **L**-based grammar is a tuple $G = \langle \Sigma, \mathcal{B}, \mathbf{c}_0, \alpha \rangle$, where:

- Σ is a (finite) set of *terminals*.
- \mathcal{B} is a (finite) set of *basic categories* (inducing the set \mathcal{C} of *categories*).
- \mathbf{c}_0 is the *initial category*; w.l.o.g, $\mathbf{c}_0 \in \mathcal{B}$.
- $\alpha : \Sigma \longrightarrow 2^{\mathcal{C}}$ is a *lexicon*, assigning each $\sigma \in \Sigma$ a *finite* set of categories $\alpha[\sigma] \subseteq \mathcal{C}$. The mapping α is naturally lifted to $\alpha : \Sigma^+ \longrightarrow (2^{\mathcal{C}})^+$.

The *language defined by* G is given by

$$L[G] = \{w \in \Sigma^+ \mid \exists \Gamma \in \alpha[w] : \vdash_{\mathbf{L}} \Gamma \triangleright \mathbf{c}_0\}.$$

The first observation is that, since introducing general elimination-rules has no effect on the set of provable sequents, $L[G]$ does not change. When considering the derivation-trees as the syntactic descriptions of phrases in the generated language, those do change, affecting the strong generative power of the formalism. However, here we are more interested in the NL-semantics defined by **L**-based grammars. The meanings are determined as follows. First, the lexicon is extended to assign each terminal σ a finite set of *pairs*, each of the form $\mathbf{c} : M_\sigma$, where \mathbf{c} is a category and M_σ is a (simply-typed) λ -term, denoting the meaning of σ (under category \mathbf{c}). Then, if $w = \sigma_1 \dots \sigma_n \in L[G]$, then the CH-image of the derivation, say M , with free variables x_1, \dots, x_n , is a “recipe” for obtaining the meaning of w , by substituting in M M_σ for x_i if $\sigma_i = \sigma$ and β -reducing the outcome.

Here, the revision with general elimination-rules with their CH-image generalized-application, has an interesting effect, related to the continuation-semantics (of natural language) advocated in [1, 2]. To illustrate this effect, consider the following fragment of a lexicon for a subset of English, containing sentences with (transitive and intransitive) verbs, and their *nominal* complements, consisting noun-phrases over *determiners* and *nouns*, using standard *generalized quantifiers theory*. The typing is based on the *primitive types* e (element)

word	category	type	meaning
some	$(np \uparrow \leftarrow n)$	$((e, t), ((e, t), t))$	$\lambda P \lambda Q \exists x. P(x) \wedge Q(x)$
every	$(np \uparrow \leftarrow n)$	$((e, t), ((e, t), t))$	$\lambda P \lambda Q \forall x. P(x) \Rightarrow Q(x)$
girl	n	(e, t)	$\lambda x. \mathbf{g}(x)$
boy	n	(e, t)	$\lambda x. \mathbf{b}(x)$
smiles	$(np \uparrow \rightarrow s)$	$((e, t), t)$	$\lambda x. \mathbf{s}(x)$
loves	$((np \uparrow \rightarrow s) \leftarrow np \uparrow)$	$((e, t), t), ((e, t), t)$	$\lambda y \lambda x. \mathbf{l}(x, y)$

Fig. 4. A fragment of a lexicon

and t (truth-value), and (τ_1, τ_2) is the functional type with domain type τ_1 and range τ_2 . The lexicon is presented in Figure 4. Abbreviate the (raised) category $(s \leftarrow (np \rightarrow s))$ as $np \uparrow$. Let us start with a simple example.

(3) Every girl smiles

Abbreviate $\lambda P \forall x. \mathbf{g}(x) \Rightarrow P(x)$ as GQ_s , and use GQ as a variable of type $((e, t), t)$ (over generalized quantifiers). Below are the two meaning derivations for (3), with regular elimination-rules (first) and general elimination-rules (second). The end result in both cases is, of course, the same:

$$\begin{array}{c}
 GQ_s(\lambda x. \mathbf{g}(x)) \equiv \forall x. \mathbf{g}(x) \Rightarrow \mathbf{s}(x) \\
 \\
 \begin{array}{c}
 \frac{\text{every}}{(np \uparrow \leftarrow n) : \lambda P \lambda Q \forall x. P(x) \Rightarrow Q(x)} \quad \frac{\text{girl}}{n : \lambda x. \mathbf{g}(x)} \quad \frac{\text{smiled}}{(np \uparrow \rightarrow s) : \lambda GQ. GQ(\lambda x. \mathbf{s}(x))} \\
 \hline
 np \uparrow : GQ_s \quad (\leftarrow E) \quad \hline
 s : GQ_s(\lambda x. \mathbf{s}(x)) \quad (\rightarrow E)
 \end{array} \\
 \\
 \begin{array}{c}
 \frac{\text{smiled}}{(np \uparrow \rightarrow s) : \lambda GQ. GQ(\lambda x. \mathbf{s}(x))} \quad [s]_k : u \quad (\rightarrow GE_k) \quad \frac{\text{every}}{(np \uparrow \leftarrow n) : \lambda P \lambda Q \forall x. P(x) \Rightarrow Q(x)} \quad \frac{\text{girl}}{n : \lambda x. \mathbf{g}(x)} \\
 \hline
 [np \uparrow]_i : GQ \quad s : GQ(\lambda x. \mathbf{s}(x)) \quad \hline
 s : GQ_s(\lambda x. \mathbf{s}(x)) \quad (\leftarrow GE_i^l)
 \end{array}
 \end{array}$$

In the “regular” derivation, when the determiner-meaning is applied to the noun-meaning (with a regular elimination rule), it does not “know” how the result is going to be used. On the other hand, in the second derivation, when the determiner-meaning is applied with a general-elimination rule, it “knows” that it is going to be used as a subject in a sentence-meaning determination, so it passes its value instead of an appropriately discharged assumption GQ .

What we see is, that we get *continuation passing* similar to the CPS-style transformation, as proposed by Barker [1, 2].

Fig. 5. Quantifier scope ambiguity derivation with general elimination-rules

Fig. 5. Quantifier scope ambiguity derivation with general elimination-rules

To see this effect better, consider the following sentence, using a transitive verb and two quantified noun-phrases, taken to exhibit the quantifier scope ambiguity. By Barker's continuation passing semantics, one needs to compute *two different* CPSs to account for the two scopes.

(4) Every girl loves some boy

Abbreviate the meaning of the subject generalized quantifier $\lambda P \forall x. \mathbf{g}(x) \Rightarrow P(x)$ as GQ_s , and that of the object $\lambda P \exists y. \mathbf{b}(y) \wedge P(y)$ as GQ_o . The two readings OWS (object wide-scope) and SWS (subject wide-scope) of (4) are given by:

$$GQ_o(\lambda y. GQ_s(\lambda x. \mathbf{l}(x, y)))$$

$$GQ_s(\lambda x. GQ_o(\lambda y. \mathbf{l}(x, y)))$$

Consider the derivations in Figure 5 for the OWS (left) and SWS (right) readings. We make use of the derived rule Ex (exchange), a known derived rule in \mathbf{L} .

Note that the location of the words in the displayed proof is insignificant, as premisses may be ordered arbitrarily. It is only the order of Γ , the set of open assumptions, that determines the actual location of the word as a contributor of an open assignment. Inspecting the two proofs, one can recognize again how each *np*-meaning, i.e., each of GQ_s , GQ_o , “sends itself” as part of the continuation of the other, similarly to the types of meaning-computations via CPSs. The use of the (Ex) derived-rule might be interpreted as the counterpart of [2] changing CPSs for obtaining the same effect of “changing the order of computation”.

4 Conclusion

This paper proposed *general elimination-rules* for the directed implications in the Lambek-calculus \mathbf{L} , studying the effect of *non-commutativity* and *peripherality*, the two characteristics of \mathbf{L} , on the formulation of such rules in this setting. The effect of such rules on \mathbf{L} -based grammars, a central topic in Type-Logical grammar, was shown, albeit by examples, to induce continuation-based semantics for the \mathbf{L} -derived natural-language phrases. This connection is still in need to be formulated as a theorem, characterising it more precisely. It was noticed already in [3] that continuation semantics is related to a proof-theoretic counterpart. There, continuation semantics is related to *type-raising* (a derived rule in \mathbf{L}), viewed as classical negation, while here the connection is with (the GE-extension of) \mathbf{L} itself. While the lexicon in the example grammar in [3] uses simpler categories than the lexicon used here, the meanings are much more complicated, represented by terms in the $\lambda\mu$ -calculus (the CH-counterpart of classical logic). The derivations of the two scopes rely on that calculus not having the Church-Rosser property. Here, we remain at the intuitionistic-linear level.

Of further interest is also augmenting the calculus with *general (directed-implications) introduction-rules*, which also discharge an assumption in an auxiliary premiss, concluding arbitrary consequences of the introduced formula.

This has direct bearing on deriving meaning of, for example, *relative clauses*. This is currently under joint investigation by Chris Barker and myself.

Acknowledgements. I wish to Thank Sara Negri for providing some useful insight on the inversion-principle and on proof-composition. I also Thank Michael Kaminski, who, by “refusing to understand”, forced me to clarify to myself better the issues involved. Thanks to Gila Halperin and Gilad Ben-Avi for various useful comments on early drafts. Thanks to Philippe de Groote for helping to clarify the relationship with his work on continuation semantics and type-raising. The research was partially funded by the VP fund for the promotion of research at the Technion.

References

1. Chris Barker. Continuation and the nature of quantification. *Natural Language Semantics*, 10:211–242, 2002.
2. Chris Barker. Continuation in natural language. In Hayo Thielecke, editor, *Proceedings of the Fourth ACM SIGPLAN Continuations Workshop (CW’04)*, 2004.
3. Philippe de Groote. Type raising, continuations, and classical logic. In Robert van Rooy and Martin Stokhof, editors, *Proceedings of the thirteenth Amsterdam Colloquium*, pages 97–101. ILLC, Amsterdam, 2001.
4. Felix Joachimski and Ralph Matthes. Short proofs of normalization for a lambda calculus, permutative conversions and Gödel’s T. *Archives for Mathematical Logic*, 42(1):59–87, 2003.
5. Joachim Lambek. The mathematics of sentence structure. *Amer. Math. monthly*, 65:154–170, 1958.
6. Michael Moortgat. Categorical type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 93–178. North Holland, 1997.
7. Sara Negri. Varieties of linear calculi. *J. of Phil. Logic*, 32(6):569–590, 2002.
8. Sara Negri and Jav Von Plato. *Structural Proof Theory*. Cambridge University Press, Cambridge, UK. 2001.
9. Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001.
10. Jan Von Plato. Natural deduction with general elimination rules. *Archive Mathematical Logic*, 40:541–567, 2001.
11. Gordon D. Plotkin. Call-by-name, call-by-value and the λ -calculus. *Theoretical Computer Science*, 1(2):125–159, 1975.
12. Jeff Polakow and Frank Pfenning. Natural deduction for intuitionistic non-commutative linear logic. in *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications (TLCA’99)*, LNCS 1581, pages 295–309. Springer-Verlag, April, 1999.
13. Dag Prawitz. *Natural Deduction: Proof-Theoretical Study*. Almqvist and Wicksell, Stockholm, 1965.
14. Dag Prawitz. Ideas and results in proof theory. In J. Fenstad, editor, *Proc. 2nd Scandinavian Symposium*. North-Holland, 1971.
15. Mark Steedman. *The Syntactic Process*. MIT Press, 2000.

A Note on the Complexity of Constraint Interaction: Locality Conditions and Minimalist Grammars^{*}

Hans-Martin Gärtner¹ and Jens Michaelis²

¹ ZAS, Jägerstr. 10–11, 10117 Berlin, Germany
gaertner@zas.gwz-berlin.de

² Universität Potsdam, Institut für Linguistik, PF 601553, 14415 Potsdam, Germany
michael@ling.uni-potsdam.de

Abstract. *Locality Conditions (LCs)* on (unbounded) dependencies have played a major role in the development of generative syntax ever since the seminal work by Ross [22]. Descriptively, they fall into two groups. On the one hand there are *intervention-based LCs (ILCs)* often formulated as “minimality constraints” (“minimal link condition,” “minimize chain links,” “shortest move,” “attract closest,” etc.). On the other hand there are *containment-based LCs (CLCs)* typically defined in terms of (generalized) grammatical functions (“adjunct island,” “subject island,” “specifier island,” etc.). Research on LCs has been dominated by two very general trends. First, attempts have been made at unifying ILCs and CLCs on the basis of notions such as “government” and “barrier” (e.g. [4]). Secondly, research has often been guided by the intuition that, beyond empirical coverage, LCs somehow contribute to restricting the formal capacity of grammars (cf. [3–p. 125], [6–p. 14f]). Both these issues, we are going to argue, can be fruitfully studied within the framework of *minimalist grammars (MGs)* as defined by Stabler [25]. In particular, we are going to demonstrate that there is a specific asymmetry between the influence of ILCs and CLCs on complexity. Thus, MGs, including an ILC, namely, the *shortest move condition (SMC)* have been shown to belong to the *mildly context-sensitive* grammar formalisms by Michaelis [14]. The same has been shown in [16, 18] for a revised version of MGs introduced in [26], which includes the SMC and an additional CLC, namely, the *specifier island condition (SPIC)*. In particular [14] and [16, 18] show that, in terms of derivable string languages, both the original MG-type and the revised MG-type constitute a subclass of the class of *linear context-free rewriting systems (LCFRSs)* in the sense of [28, 29], and thus, a series of other formalism classes all generating the same class of string languages as LCFRSs. Here we will demonstrate that removing the SMC from the revised MG-version increases the generative power in such a way that the resulting formalism is not mildly context-sensitive anymore. This suggests that intuitions to the contrary notwithstanding, imposing an LC as such, here the SPIC, does not necessarily reduce formal complexity.

^{*} This work has been carried out partially funded by DFG-grant. Thanks to two anonymous referees for valuable comments on a previous version of this paper.

1 Introduction

Locality Conditions (LCs) on (unbounded) dependencies have played a major role in the development of generative syntax ever since the seminal work by Ross [22]. Descriptively, they fall into two groups. On the one hand there are *intervention-based LCs (ILCs)*, very sketchily illustrated in (1).

$$[\dots \alpha \dots [\dots \beta \dots \gamma \dots]] \quad (1)$$

An ILC constrains dependencies between α and γ across an intervening β , where intervention is defined in terms of c-command as well as the features and position of β relative to the features and positions of α and γ . ILCs are often formulated as “minimality constraints” (“minimal link condition,” “minimize chain links,” “shortest move,” “attract closest,” etc.) on the assumption that more minimal dependencies could have been formed between α and β , and/or β and γ . On the other hand *containment-based LCs (CLCs)* exist such as given in (2).

$$[\dots \alpha \dots [\beta \dots \gamma \dots]] \quad (2)$$

A CLC constrains dependencies between α and γ across a constituent β containing γ but excluding α . Typically the containers are defined in terms of (generalized) grammatical functions (“adjunct island,” “subject island,” “specifier island,” etc.). Research on LCs has been dominated by two very general trends. First, attempts have been made at unifying ILCs and CLCs on the basis of notions such as “government” and “barrier” (see e.g. [4]). Secondly, research has often been guided by the intuition that, beyond empirical coverage, LCs somehow contribute to restricting the formal capacity of grammars (cf. [3–p. 125], [6–p. 14f]). Both these issues, we are going to argue, can be fruitfully studied within the framework of *minimalist grammars (MGs)* as defined by Stabler [25].¹ In particular, we are going to show that there is a specific asymmetry between the influence of ILCs and CLCs on complexity. Crucial starting point for our demonstration is the fact that MGs, including an ILC, namely, the *shortest move condition (SMC)* have been shown to belong to the *mildly context-sensitive*

¹ Research on LCs in terms of *categorial grammar (CG)* has taken at least two rather divergent directions. Within the *combinatory* brand of CG, *CCG*, there has been a tendency to refrain from syntactically encoding LCs. This is based on the intuition that “the origin of constraints on long-range dependencies ultimately lies in semantic coherence properties of the nonstandard constituents that combination into islands creates [...]” [27–p. 65]. The *type logical* approach to CG has implemented CLCs rather directly in terms of unary operators that “block associativity” [19] or cause “structural inhibition” [20]. For one complexity result on the resulting *multi-modal CGs*, showing that under certain conditions they preserve the weak context-freeness of the Lambek calculus, see [9]. It would be attractive to compare and contrast our results on MGs more closely with CG-approaches. (In particular, it might be worth to explore hybrid versions of both CG-brands, especially given the fact that CCGs undergoing certain restrictions have been shown to constitute a mildly context-sensitive formalism (cf. [28, 29]), and thus, to be rather closely related to MGs.) However, this will have to be left for further research.

grammar formalisms by Michaelis [14]. The same has been shown in [16] for a revised version of MGs defined in [26], which includes the SMC and an additional CLC, namely, the *specifier island condition (SPIC)*.

The type of MG introduced in [25] provides an attempt at a rigorous algebraic formalization of the perspectives currently adopted within the linguistic framework of transformational grammar (see e.g. [5]). An MG, roughly speaking, is a formal device which specifies a countable set of finite, binary (ordered) trees each being equipped with a leaf-labeling function assigning a string of features to each leaf, and with an additional binary relation, the asymmetric relation of (*immediate*) *projection*, defined on the set of pairs of siblings. The base of an MG is formed by a *lexicon* (a finite set of single node trees in the above sense) and two structure building functions: *merge* (combining two trees) and *move* (transforming a given tree). Both functions build structure by canceling particular matching instances of features within the leaf-labels of the trees to which they are applied. The closure of the lexicon under these two functions is the set of trees characterized by the MG. As shown in [14], this MG-type constitutes a mildly context-sensitive formalism in the sense that it provides a weakly equivalent subclass of *linear context-free rewriting systems (LCFRSs)* [28, 29]. Independent work in [8] and [17] has proven the reverse to hold as well. Hence, MGs as defined in [25], beside LCFRSs, join to a series of formalism classes—among which there are e.g. the class of *local unordered scattered context grammars* [21], the class of *multicomponent tree adjoining grammars* in their set-local variant of admitted adjunction (cf. [29]), the class of *multiple context-free grammars* [24], or the class of *simple positive range concatenation grammars* [1]—all generating the same class of string languages. For a list of some further of such classes of generating devices see e.g. [21].

Inspired, i.a., by the linguistic work presented in [12], in [26] a revised type of an MG has been proposed whose departure from the version in [25] can be seen as twofold: the revised type of an MG neither employs any kind of *head movement* nor *covert phrasal movement*, and an additional restriction is imposed on the move-operator regulating which maximal projection may move *overtly* into the highest specifier position. Deviating from the operator *move* as originally defined in [25], a constituent has to belong to the transitive complement closure of a given tree or to be a specifier of such a constituent in order to be movable at all. Employing and extending the methods developed in [14] and [17], it was shown in [16, 18] and [15] that, in terms of derivable string languages, the revised MG-type is not only subsumed by LCFRSs, but is identical to a particular subclass of the latter: the righthand side of each rewriting rule of a corresponding LCFRS involves at most two nonterminals, and if two nonterminals appear on the righthand side then only simple strings of terminals are derivable from the first one. Whether the respective classes of string languages derivable by the corresponding LCFRS-subclass and the class of all LCFRSs—and thus the respective classes of string languages derivable by the class of revised MGs as defined in [26] and the class of MGs as defined in [25]—are identical seems to be an open problem. Here, we will demonstrate that removing the SMC from

the revised MG-version increases the generative power in such a way that the resulting formalism is not mildly context-sensitive. This suggests that intuitions to the contrary notwithstanding, imposing an LC as such, here the SPIC, does not necessarily reduce formal complexity.

The paper is structured as follows: Section 2 introduces (revised) minimalist grammars in two versions, one standard (MGs, Definition 3) and one without the SMC (MG^{SMC} , Definition 4). In Section 3 we show, how to derive the language $\{a^{2^n} \mid n \in \mathbb{N}\}$, i.e. a language without the constant growth property, and thus a non-mildly context-sensitive language. Section 4 provides a short conclusion and an outlook on further research. Throughout the rest of the paper we use the term *minimalist grammar* and its abbreviation *MG* in order to refer to an MG of the revised type as defined in [26], unless explicitly indicated otherwise.

2 (Revised) Minimalist Grammars

Throughout we let $\neg\text{Syn}$ and Syn be a finite set of *non-syntactic features* and a finite set of *syntactic features*, respectively, in accordance with (F1) and (F2) below. We take Feat to be the set $\neg\text{Syn} \cup \text{Syn}$.

- (F1) $\neg\text{Syn}$ is disjoint from Syn and partitioned into a set Phon of *phonetic features* and a set Sem of *semantic features*.
- (F2) Syn is partitioned into a set Base of (*basic*) *categories*, a set Select of *selectors*, a set Licensees of *licensees* and a set Licensors of *licensors*. For each $x \in \text{Base}$, usually typeset as \mathbf{x} , the existence of a matching $x' \in \text{Select}$, denoted by \mathbf{x} , is possible. For each $x \in \text{Licensees}$, usually depicted as $\neg\mathbf{x}$, the existence of a matching $x' \in \text{Licensors}$, denoted by $+\mathbf{x}$, is possible. Base includes at least the category \mathbf{c} .

Definition 1. An *expression (over Feat)* is a five-tuple $\langle N_\tau, \triangleleft_\tau^*, \prec_\tau, <_\tau, \text{label}_\tau \rangle$ obeying (E1)–(E4).

- (E1) $\langle N_\tau, \triangleleft_\tau^*, \prec_\tau \rangle$ is a finite, binary (ordered) tree defined in the usual sense: N_τ is the finite, non-empty set of *nodes*, and \triangleleft_τ^* and \prec_τ are the respective binary relations of *dominance* and *precedence* on N_τ .²
- (E2) $<_\tau \subseteq N_\tau \times N_\tau$ is the asymmetric relation of (*immediate*) *projection* that holds for any two siblings in $\langle N_\tau, \triangleleft_\tau^*, \prec_\tau \rangle$, i.e., for each $x \in N_\tau$ different from the root of $\langle N_\tau, \triangleleft_\tau^*, \prec_\tau \rangle$ either $x <_\tau \text{sibling}_\tau(x)$ or $\text{sibling}_\tau(x) <_\tau x$ holds.³

² Thus, \triangleleft_τ^* is the reflexive-transitive closure of $\triangleleft_\tau \subseteq N_\tau \times N_\tau$, the relation of *immediate dominance* on N_τ .

³ For each $x \in N_\tau$ different from the root of $\langle N_\tau, \triangleleft_\tau^*, \prec_\tau \rangle$, $\text{sibling}_\tau(x)$ is the (unique) sibling of x . If $x <_\tau y$ for any $x, y \in N_\tau$, x is said to (*immediately*) *project over* y .

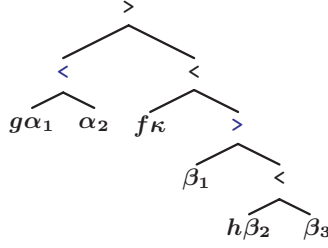


Fig. 1. A typical expression over *Feat*

(E3) $label_\tau$ is the *leaf-labeling function*, i.e., a total function from the set of all leaves of $\langle N_\tau, \triangleleft_\tau^*, \prec_\tau \rangle$ into $Syn^*Phon^*Sem^*$.⁴

(E4) $\langle N_\tau, \triangleleft_\tau^*, \prec_\tau \rangle$ is a subtree of the natural interpretation of a tree domain.⁵

We take $Exp(Feat)$ to denote the set of all expressions over *Feat*.

Let $\tau = \langle N_\tau, \triangleleft_\tau^*, \prec_\tau, <_\tau, label_\tau \rangle \in Exp(Feat)$.⁶

For each $x \in N_\tau$, the *head of x (in τ)*, denoted by $head_\tau(x)$, is the (unique) leaf of τ with $x \triangleleft_\tau^* head_\tau(x)$ such that each $y \in N_\tau$ on the path from x to $head_\tau(x)$ with $y \neq x$ projects over its sibling, i.e. $y <_\tau sibling_\tau(y)$. The *head of τ* is the head of τ 's root. τ is said to be a *head* (or *simple*) if N_τ consists of exactly one node, otherwise τ is said to be a *non-head* (or *complex*).

A five-tuple $v = \langle N_v, \triangleleft_v^*, \prec_v, <_v, label_v \rangle$ is a *subexpression of τ* if $\langle N_v, \triangleleft_v^*, \prec_v \rangle$ is a subtree of $\langle N_\tau, \triangleleft_\tau^*, \prec_\tau \rangle$, and if $<_v = <_\tau \upharpoonright_{N_v \times N_v}$ and $label_v = label_\tau \upharpoonright_{N_v}$ hold.⁷ Thus, $v \in Exp(Feat)$. Such an v is a *maximal projection (in τ)* if v 's root is a node $x \in N_\tau$ such that x is the root of τ , or such that $sibling_\tau(x) <_\tau x$. $MaxProj(\tau)$ is the set of all maximal projections in τ .

⁴ For each set M , M^* is the Kleene closure of M , including ϵ , the empty string. M_ϵ denotes the set $M \cup \{\epsilon\}$. For any $K, L \subseteq M^*$, KL is the product of K and L under concatenation, i.e., the (string) set $\{kl \mid k \in K, l \in L\} \subseteq M^*$.

⁵ We take \mathbb{N} to denote the set of all non-negative integers. A *tree domain* is a non-empty set $N_v \subseteq \mathbb{N}^*$ such that for all $\chi \in \mathbb{N}^*$ and $i \in \mathbb{N}$ it holds that $\chi \in N_v$ if $\chi\chi' \in N_v$ for some $\chi' \in \mathbb{N}^*$, and $\chi i \in N_v$ if $\chi j \in N_v$ for some $j \in \mathbb{N}$ with $i < j$. $\langle N_v, \triangleleft_v^*, \prec_v \rangle$ is the *natural (tree) interpretation* of N_v in the case that for all $\chi, \psi \in N_v$ it holds that $\chi \triangleleft_v \psi$ iff $\psi = \chi i$ for some $i \in \mathbb{N}$, and $\chi \prec_v \psi$ 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 $i < j$.

⁶ Note that the leaf-labeling function $label_\tau$ can easily be extended to a total labeling function ℓ_τ from N_τ into $Feat^* \cup \{<, >\}$, where $<$ and $>$ are two new distinct symbols: to each non-leaf $x \in N_\tau$ we can assign a label from $\{<, >\}$ by ℓ_τ such that $\ell_\tau(x) = <$ iff $y <_\tau z$ for $y, z \in N_\tau$ with $x \triangleleft_\tau y$, $x \triangleleft_\tau z$, and $y \prec_\tau z$. In this sense a concrete $\tau \in Exp(Feat)$ is depictable in the way demonstrated in Fig. 1.

⁷ For a binary relation $r \subseteq A \times B$, A and B being two sets, and for any two sets A' and B' , $r \upharpoonright_{A' \times B'}$ is the restriction of r to $A' \times B'$, i.e., the set $\{(a, b) \in r \mid a \in A', b \in B'\}$. In case r is a function, we also write $r \upharpoonright_{A'}$ instead of $r \upharpoonright_{A' \times B'}$.

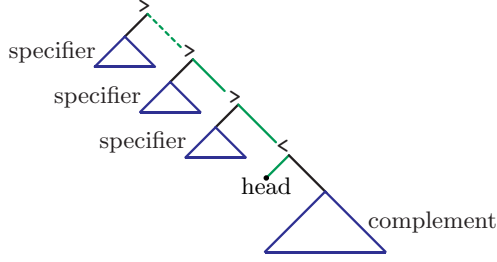


Fig. 2. The typical structure of a (minimalist) expression over *Feat*

$comp_\tau \subseteq MaxProj(\tau) \times MaxProj(\tau)$ is the binary relation defined such that for all $v, \phi \in MaxProj(\tau)$ it holds that $v comp_\tau \phi$ iff $head_\tau(r_v) <_\tau r_\phi$, where r_v and r_ϕ are the roots of v and ϕ , respectively. If $v comp_\tau \phi$ holds for some $v, \phi \in MaxProj(\tau)$ then ϕ is a *complement of v (in τ)*. $comp_\tau^+$ is the transitive closure of $comp_\tau$. $Comp^+(\tau)$ is the set $\{v \mid \tau comp_\tau^+ v\}$.

$spec_\tau \subseteq MaxProj(\tau) \times MaxProj(\tau)$ is the binary relation defined such that for all $v, \phi \in MaxProj(\tau)$ it holds that $v spec_\tau \phi$ iff $r_\phi = sibling_\tau(x)$ for some $x \in N_\tau$ with $r_v \triangleleft_\tau^+ x \triangleleft_\tau^+ head_\tau(r_v)$, where r_v and r_ϕ are the roots of v and ϕ , respectively. If $v spec_\tau \phi$ for some $v, \phi \in MaxProj(\tau)$ then ϕ is a *specifier of v (in τ)*. $Spec(\tau)$ is the set $\{v \mid \tau spec_\tau v\}$.

An $v \in MaxProj(\tau)$ is said to *have*, or likewise, to *display (open) feature f* if the label assigned to v 's head by $label_\tau$ is non-empty and starts with an instance of $f \in Feat$.⁸

τ is *complete* if its head-label is in $\{c\}Phon^*Sem^*$, and if the label of each other leaf is in $Phon^*Sem^*$. Hence, a complete expression over *Feat* is an expression that has category c , and this instance of c is the only instance of a syntactic feature within all leaf-labels.

The *phonetic yield* of τ , denoted by $Y_{Phon}(\tau)$, is the string which results from concatenating in “left-to-right-manner” the labels assigned to the leaves of $\langle N_\tau, \triangleleft_\tau^*, \prec_\tau \rangle$ via $label_\tau$, and replacing all instances of non-phonetic features with the empty string, afterwards.

An $v = \langle N_v, \triangleleft_v^*, \prec_v, <_v, label_v \rangle \in Feat(Exp)$ is (*label preserving*) *isomorphic to τ* if there is a bijective function i from N_τ onto N_v with $x \triangleleft_\tau y$ iff $i(x) \triangleleft_v i(y)$, $x \prec_\tau y$ iff $i(x) \prec_v i(y)$, $x <_\tau y$ iff $i(x) <_v i(y)$ for $x, y \in N_\tau$, and with $label_\tau(x) = label_v(i(x))$ for each $x \in N_\tau$ being a leaf of τ . i is an *isomorphism (from τ to v)*.

Definition 2. For $\tau = \langle N_\tau, \triangleleft_\tau^*, \prec_\tau, <_\tau, label_\tau \rangle \in Exp(Feat)$ with $N_\tau = tN_v$ for some $t \in \mathbb{N}^*$ and some tree domain N_v , and for $r \in \mathbb{N}^*$, $(\tau)_r$ denotes the *expression shifting τ to r* , i.e., the expression $\langle N_{\tau(r)}, \triangleleft_{\tau(r)}^*, \prec_{\tau(r)}, <_{\tau(r)}, label_{\tau(r)} \rangle$ over

⁸ Thus the expression depicted in Fig. 1 has feature f , while its specifier and its complement have feature g and h , respectively.

Feat with $N_{\tau(r)} = rN_v$ such that the function $i_{\tau(r)}$ from N_τ onto $N_{\tau(r)}$ with $i_{\tau(r)}(tx) = rx$ for all $x \in N_v$ is an isomorphism from τ to $(\tau)_r$.⁹

For $v, \phi \in \text{Exp}(\text{Feat})$ let $\chi = \langle N_\chi, \triangleleft_\chi^*, \prec_\chi, <_\chi, \text{label}_\chi \rangle$ be a complex expression over *Feat* with root ϵ such that $(v)_0$ and $(\phi)_1$ are the two subexpressions of χ whose roots are immediately dominated by ϵ . Then χ is of one of two forms: in order to refer to χ we write $[<v, \phi]$ if $0 <_\chi 1$, and $[>v, \phi]$ if $1 <_\chi 0$.

Definition 3 ([26]). A *minimalist grammar (MG)* is a five-tuple of the form $G = \langle \neg\text{Syn}, \text{Syn}, \text{Lex}, \Omega, \mathbf{c} \rangle$ with Ω being the operator set consisting of the structure building functions *merge* and *move* defined w.r.t. *Feat* as in (me) and (mo) below, respectively, and with *Lex* being a *lexicon (over Feat)*, i.e., *Lex* is a finite set of simple expressions over *Feat*, and each lexical item $\tau \in \text{Lex}$ is of the form $\langle N_\tau, \triangleleft_\tau^*, \prec_\tau, <_\tau, \text{label}_\tau \rangle$ such that $N_\tau = \{\epsilon\}$, and such that $\text{label}_\tau(\epsilon)$ is in $(\text{Select} \cup \text{Licensors})^* \text{Base Licensees}^* \text{Phon}^* \text{Sem}^*$.

(me) *merge* is a partial mapping from $\text{Exp}(\text{Feat}) \times \text{Exp}(\text{Feat})$ into $\text{Exp}(\text{Feat})$.

A pair $\langle v, \phi \rangle$ with $v, \phi \in \text{Exp}(\text{Feat})$ belongs to $\text{Dom}(\text{merge})$ if for some $\mathbf{x} \in \text{Base}$ and $\kappa, \lambda \in \text{Feat}^*$, conditions (i) and (ii) are fulfilled:¹⁰

- (i) the head-label of v is $\mathbf{x}\kappa$ (i.e. v has selector \mathbf{x}), and
- (ii) the head-label of ϕ is $\mathbf{x}\lambda$ (i.e. ϕ has category \mathbf{x}).

Then,

(me.1) $\text{merge}(v, \phi) = [<v', \phi']$ if v is simple, and

(me.2) $\text{merge}(v, \phi) = [>\phi', v']$ if v is complex,

where v' and ϕ' result from v and ϕ , respectively, just by deleting the instance of the feature that the respective head-label starts with (cf. Fig. 3).

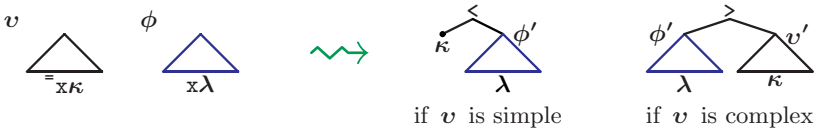


Fig. 3. $\text{merge}(v, \phi)$ according to (me)

(mo) *move* is a partial mapping from $\text{Exp}(\text{Feat})$ to $\text{Exp}(\text{Feat})$. An $v \in \text{Exp}(\text{Feat})$ is in $\text{Dom}(\text{move})$ if for some $\mathbf{x} \in \text{Licensees}$ and $\kappa \in \text{Feat}^*$, (i)–(iii) are true:

⁹ For any $t \in \mathbb{N}^*$ and $N \subseteq \mathbb{N}^*$, tN is just the concatenation product $\{t\}N$, i.e., the set $\{tx \mid x \in N\} \subseteq \mathbb{N}^*$ (cf. fn. 4). Note that for each $\tau = \langle N_\tau, \triangleleft_\tau^*, \prec_\tau, <_\tau, \text{label}_\tau \rangle$ from $\text{Exp}(\text{Feat})$, a $t \in \mathbb{N}^*$ and tree domain N_v with $N_\tau = tN_v$ exist by (E4).

¹⁰ For a partial function f from a set A into a set B , $\text{Dom}(f)$ is the domain of f , i.e., the set of all $x \in A$ for which $f(x)$ is defined.

- (i) the head-label of v is $+X\kappa$ (i.e. v has licenser $+X$),
- (ii) there is exactly one $\phi \in \text{MaxProj}(v)$ with head-label $-x\lambda$ for some $\lambda \in \text{Feat}^*$ (i.e. there is exactly one $\phi \in \text{MaxProj}(v)$ that has feature $-x$), and
- (iii) there exists a $\chi \in \text{Comp}^+(v)$ with $\phi = \chi$ or $\phi \in \text{Spec}(\chi)$.

Then,

$$\text{move}(v) = [_{>\phi', v'}],$$

where $v' \in \text{Exp}(\text{Feat})$ results from v by canceling the instance of $+X$ the head-label of v starts with, while the subtree ϕ is replaced by a single node labeled ϵ . $\phi' \in \text{Exp}(\text{Feat})$ arises from ϕ by deleting the instance of $-x$ the head-label of ϕ starts with (cf. Fig. 4).



Fig. 4. $\text{move}(v)$ according to (mo)

Note that it is condition (ii) of (mo) which can be seen as providing a strict implementation of the *shortest movement condition* (SMC): competing open licensees in one and the same given expression do not allow one to derive a complete expression from the given one. It is condition (iii) of (mo) which provides an implementation of the *specifier island condition* (SPIC): a constituent has to belong to the transitive complement closure of a given tree or to be a specifier of such a constituent in order to be movable at all.

Since we are interested in the question of whether the generative capacity of our formalism is affected by giving up the SMC and sticking to the SPIC, we next present the definition of an MG without SMC.

Definition 4 ([26]). A *minimalist grammar without SMC* (MG^{-SMC}) is a five-tuple of the form $\langle \neg \text{Syn}, \text{Syn}, \text{Lex}, \Omega, c \rangle$ where Ω is the operator set consisting of the structure building functions *merge* and move^{-SMC} defined w.r.t. Feat as in (me) above and (mo^{-SMC}) below, respectively, and where Lex is a lexicon over Feat defined as in Definition 3.

(mo^{-SMC}) *move* is a partial mapping from $\text{Exp}(\text{Feat})$ to $\mathcal{P}_{\text{fin}}(\text{Exp}(\text{Feat}))$.¹¹ An $v \in \text{Exp}(\text{Feat})$ is in $\text{Dom}(\text{move})$ if for some $-x \in \text{Licensees}$ and $\kappa \in \text{Feat}^*$, (i)–(iii) are true:

¹¹ $\mathcal{P}_{\text{fin}}(\text{Exp}(\text{Feat}))$ is the class of all finite subsets of $\text{Exp}(\text{Feat})$.

- (i) the head-label of v is $+X\kappa$ (i.e. v has licenser $+X$),
- (ii) there is some $\phi \in \text{MaxProj}(v)$ with head-label $-x\lambda$ for some $\lambda \in \text{Feat}^*$ (i.e. there is some $\phi \in \text{MaxProj}(v)$ that has feature $-x$), and
- (iii) there exists a $\chi \in \text{Comp}^+(v)$ with $\phi = \chi$ or $\phi \in \text{Spec}(\chi)$.

Then,

$$\text{move}^{\text{-SMC}}(v) = \left\{ \left[> \phi', v' \right] \left| \begin{array}{l} \phi \in \text{MaxProj}(\phi) \text{ with head-label } -x\lambda \\ \text{for some } \lambda \in \text{Feat}^* \text{ such that there is} \\ \text{a } \chi \in \text{Comp}^+(v) \text{ for which } \phi = \chi \text{ or} \\ \phi \in \text{Spec}(\chi) \end{array} \right. \right\},$$

where $v' \in \text{Exp}(\text{Feat})$ results from v by canceling the instance of $+X$ the head-label of v starts with, while the subtree ϕ is replaced by a single node labeled ϵ . $\phi' \in \text{Exp}(\text{Feat})$ arises from ϕ by deleting the instance of $-x$ the head-label of ϕ starts with (cf. Fig. 4).

Let $G = \langle \neg \text{Syn}, \text{Syn}, \text{Lex}, \Omega, \mathbf{c} \rangle$ be an MG, respectively an $\text{MG}^{\text{-SMC}}$. Then the closure of G , $\text{CL}(G)$, is the set $\bigcup_{k \in \mathbb{N}} \text{CL}^k(G)$, where $\text{CL}^0(G) = \text{Lex}$, and for $k \in \mathbb{N}$, $\text{CL}^{k+1}(G) \subseteq \text{Exp}(\text{Feat})$ is recursively defined as the set

$$\begin{aligned} & \text{CL}^k(G) \cup \{ \text{merge}(v, \phi) \mid \langle v, \phi \rangle \in \text{Dom}(\text{merge}) \cap \text{CL}^k(G) \times \text{CL}^k(G) \} \\ & \cup \{ \text{move}(v) \mid v \in \text{Dom}(\text{move}) \cap \text{CL}^k(G) \} \end{aligned}$$

in case G is an MG, respectively as the set

$$\begin{aligned} & \text{CL}^k(G) \cup \{ \text{merge}(v, \phi) \mid \langle v, \phi \rangle \in \text{Dom}(\text{merge}) \cap \text{CL}^k(G) \times \text{CL}^k(G) \} \\ & \cup \bigcup_{v \in \text{Dom}(\text{move}^{\text{-SMC}}) \cap \text{CL}^k(G)} \text{move}^{\text{-SMC}}(v), \end{aligned}$$

in case G is an $\text{MG}^{\text{-SMC}}$. The set $\{Y_{\text{phon}}(\tau) \mid \tau \in \text{CL}(G) \text{ and } \tau \text{ complete}\}$, denoted by $L(G)$, is the (*string*) *language derivable by* G .

Definition 5. A set L is a *minimalist language (ML)* if $L = L(G)$ for some MG G , and it is a *minimalist language without SMC* ($\text{ML}^{\text{-SMC}}$), if $L = L(G)$ for some $\text{MG}^{\text{-SMC}}$ G .

Corollary 1. *Each ML has the constant growth property.*¹² □

This corollary is an immediate consequence of the fact (cf. [16, 18]) that each language derivable by an MG in the sense of Definition 3 is a language derivable by a *linear context-free rewriting system (LCFRS)* in the sense of [28, 29].

¹² For each set M and each $L \subseteq M^*$, L has the constant growth property, if there is an $N \in \mathbb{N}$ such that for all $w_1, w_2 \in L$ with $|w_1| < |w_2|$, and for which there is no $w_3 \in L$ with $|w_1| < |w_3| < |w_2|$, it holds that $|w_2| - |w_1| \leq N$. Here, for $w \in M^*$, $|w|$ denotes the length of w .

2.1 The Notion of a Relevant Expression

The notion of what is a *relevant expression* within the closure of an MG, respectively MG^{SMC} , G , is of some importance. We refer to an expression $\tau \in \text{CL}(G)$ as *relevant* if it serves to derive a complete expression. In particular, we want to emphasize that both in the case of an MG and in the case of an MG^{SMC} , condition (iii) of the definition of the move-operator guarantees the following: whenever an expression $v \in \text{CL}(G)$ can be employed in order to generate a complete expression, there is no maximal projection $\psi \in \text{MaxProj}(v)$ such that ψ displays an unchecked licensee and is properly contained within some $\chi \in \text{Comp}^+(v)$.

To put it differently, if we applied the move-operator to some $v \in \text{CL}(G)$ such that some $\chi \in \text{Comp}^+(v)$ becomes a specifier of the resulting expression, it would be impossible to check off in a later derivation step any licensee feature displayed by some $\psi \in \text{MaxProj}(v)$ properly contained in χ , because applying the move-operator to v , ψ would end up in a position not matching condition (iii) of the definition of the move-operator, and this property is inherited by any expression subsequently derived. In the syntactic literature this is often referred to as a “freezing effect.” Complying with this effect, we can, in the case of an MG^{SMC} , “hide” an unbounded, finite number of different instances of the same licensee “along” the transitive complement closure of a single expression; and we exploit exactly this possibility in the next section, when we define an MG^{SMC} deriving a language which is not mildly context-sensitive.

3 A Non-mildly Context-Sensitive ML^{SMC}

We are now going to present an MG^{SMC} deriving a language which does not fulfil the constant growth property, namely, the language $\{a^{2^n} \mid n \in \mathbb{N}\}$.

Example 1. Assume $G_{\text{ex}} = \langle \neg \text{Syn}, \text{Syn}, \text{Lex}, \Omega, \text{c} \rangle$ to be the MG^{SMC} for which

$$\begin{array}{lll} \text{Sem} = \emptyset & \text{Base} = \{\text{c}, \text{x}, \text{y}, \text{z}\} & \text{Licensees} = \{-1, -\text{m}\} \\ \text{Phon} = \{a\} & \text{Select} = \{^=\text{c}, ^=\text{x}, ^=\text{y}, ^=\text{z}\} & \text{Licensors} = \{+\text{L}, +\text{M}\} \end{array}$$

and for which Lex consists of the following 9 simple expressions:¹³

$$\begin{array}{llll} \beta_1 = \text{w-m} & \beta_2 = ^=\text{wx-1} & & \\ \gamma_1 = ^=\text{x+My-m} & \gamma_2 = ^=\text{y+Lz-1} & \gamma_3 = ^=\text{zy-1} & \gamma_4 = ^=\text{zx-1} \\ \delta_1 = ^=\text{x+Mc} & \delta_2 = ^=\text{c+Lca} & & \end{array}$$

Instead of a strictly formal proof that $L(G_{\text{ex}}) = \{a^{2^n} \mid n \in \mathbb{N}\}$ we will give the crucial details in a descriptive manner.

¹³ Since all lexical entries of G_{ex} are heads, we simply represent each of them by its (unique) label.



Fig. 5. The expression $\text{merge}(\beta_2, \beta_1)$

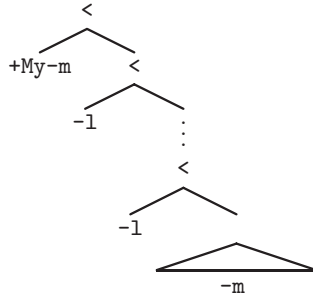


Fig. 6. Starting the derivation cycle by merging with γ_1

Each derivation of an expression belonging to $CL(G_{\text{ex}})$ necessarily starts by merging expressions β_2 and β_1 , yielding an expression which displays category x and contains a maximal projection displaying licensee $-m$ (cf. Fig. 5). Hence, the expression $\text{merge}(\beta_2, \beta_1)$ can be selected by γ_1 as well as δ_1 .

The lexical items γ_1 , γ_2 , γ_3 and γ_4 can be employed to run a derivation cycle in order to double the number of maximal projections appearing in a given expression and displaying licensee -1 . The cycle starts by merging with expression γ_1 (cf. Fig. 6) and next checking an instance of licensee $-m$; and the cycle stops by merging with γ_4 , yielding an expression displaying category x (cf. Fig. 9). In between, repeatedly carrying out sequences of applying merge with γ_2 , applying move^{SMC} , and applying merge with γ_3 lead to the doubling of unchecked instances of licensee -1 (cf. Fig. 7 and 8). The end of the cycle is “indicated” by an appearance of licensee $-m$: after a maximal projection displaying $-m$ has become the lowest embedded constituent displaying any unchecked licensee at all, we necessarily have to merge with γ_4 to prevent the derivation from running into a configuration which makes it impossible to finally generate a complete expression.

Note that, whenever move^{SMC} can be applied to some $v \in CL(G_{\text{ex}})$, a maximal projection $\phi \in \text{MaxProj}(v)$ displaying the corresponding licensee triggering move^{SMC} always belongs to the transitive complement closure of v , i.e. $\text{Comp}^+(v)$. The crucial point now is that, although v may contain even several different maximal projections displaying the same licensee, in any case only the lowest maximal projection from $\text{Comp}^+(v)$ can be moved in order to derive a complete expression.¹⁴

¹⁴ Recall the notion of a *relevant expression*.

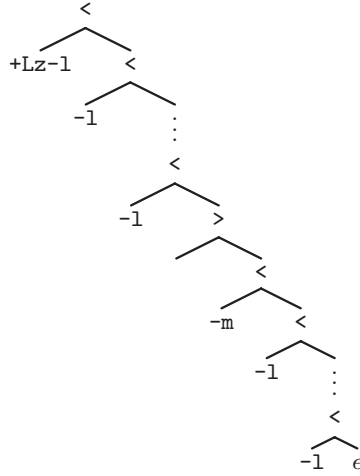


Fig. 7. Within the cycle after merging with γ_2 : “preparing” the doubling of the lowest instance of licensee -1 which now gets checked off by $move^{SMC}$, but, virtually, has already been “reinstantiated” within the label of γ_2

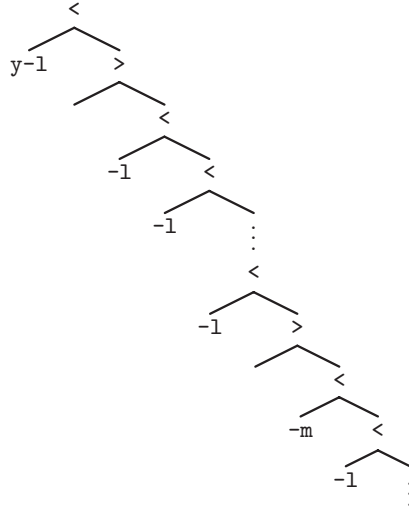


Fig. 8. Within the cycle after merging with γ_3 : the instance of licensee -1 “reinstantiated” before by means of the label of γ_2 (cf. Fig. 7), now has been “doubled” by means of the label of γ_3

After having merged with γ_4 , the repetition of the derivational cycle, just described, is blocked by merging with δ_1 instead of merging with γ_1 (cf. Fig. 10). Then, after the “cycle end-marking” instance of licensee $-m$ has been checked through an application of $move^{SMC}$, all instances of licensee -1 get successively

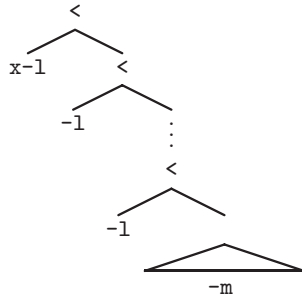


Fig. 9. Stopping the derivation cycle by merging with γ_4

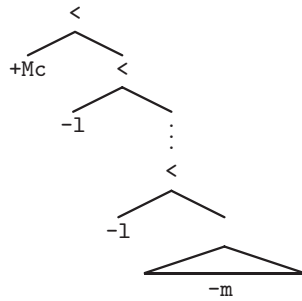


Fig. 10. Leaving the derivation cycle by merging with δ_1

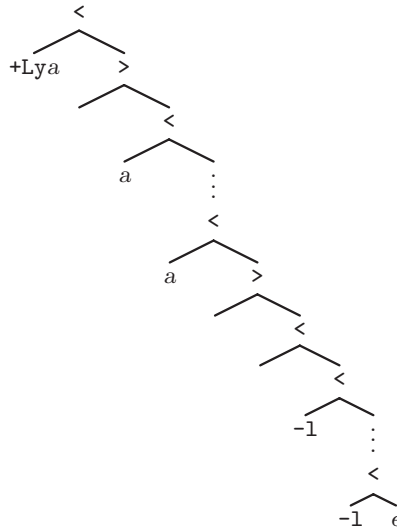


Fig. 11. Finishing the derivation by successively merging with δ_2 and checking off the remaining instances of licensee -1 , thereby introducing an instance of an a for each -1

checked by first merging with δ_2 and applying $move^{SMC}$ afterwards. Hence, in particular, for each instance of licensee -1 exactly one instance of (phonetic) feature a is introduced via merging with δ_2 (cf. Fig. 11). Therefore, $\{a^{2^n} \mid n \in \mathbb{N}\}$ is in fact the language derived by G_{ex} .

4 Conclusion and Outlook

We have shown that removing the SMC from MGs increases their generative capacity beyond mild context-sensitivity. In particular, we have provided an exemplary MG^{SMC} deriving the language $\{a^{2^n} \mid n \in \mathbb{N}\}$, i.e. a language lacking the constant growth property. Importantly, this effect arises in spite of the fact that MG^{SMC} s are constrained by a CLC, namely, the SPIC. This suggests that, intuitions to the contrary notwithstanding, the imposition of LCs on grammars as such, in the case at hand the SPIC, does not automatically reduce their generative capacity.

Note also that closely in keeping with some further suggestions in [12], a certain type of a *strict minimalist grammar (SMG)* has been introduced in [26] as well. This MG-type allows only movement of constituents belonging to the transitive complement closure of a tree. But in contrast to the MG-type matching our Definition 3, the triggering licensee feature may head the head-label of any constituent within the reflexive-transitive specifier closure of a moving constituent. Furthermore, due to the general definition of a lexical item of an SMG, an SMG does not permit the creation of multiple specifiers during the course of a derivation. Beside these differences, SMGs have implemented the SMC within the definition of the move-operator in the same way MGs have. SMGs and MGs have been shown to be weakly equivalent in [15, 18] confirming a conjecture explicitly stated in [26]. Note that, if we defined *strict minimalist grammars without SMC* (SMG^{SMC}) by relaxing the condition (ii) of the move-operator as we did for MGs in Definition 4, our example MG^{SMC} would also match the criteria such an SMG^{SMC} had to fulfil. This, of course, is of interest, since it suggests that providing the implementation of the SPIC with a “final strictness,” does not prevent us from being able to derive non-mildly context-sensitive languages when giving up the SMC.

There has recently been made another interesting attempt, namely, by Kobele [11], to look at the consequences concerning the generative capacity of MGs when changing the original definition from [25] and allowing a certain kind of feature percolation. In fact, Kobele proved that, if the syntactic features of a head are presented as strings which are checked “from left to right,” and if movement to a specifier position generally allows for the possibility that the syntactic features of the specifier’s head are inherited by the attracting head in such a way that they become an integral part of the syntactic features of the attracting head, then MGs modified in this respect allow one to derive any language of

type 0.^{15,16} We conjecture that this is true of MG^{SMC} s as well. The reason for this is that MG^{SMC} s (i.e. MGs for which the SPIC, the specifier island condition, SPIC, but not the SMC, the shortest move condition, hold) on the one hand and MGs of the Kobele-type (i.e. MGs generally allowing for feature percolation from specifiers to heads) on the other appear to constitute complementary pictures of one and the same thing. The sort of feature percolation Kobele considers allows to “collect” instantiations of the same feature type within a single head-label without violating the SMC, since only the first feature instantiation within the head-label is visible to the move-operator. The implementation of the SPIC in MGs discussed in our paper and the simultaneous dropping of the SMC allows to “collect” instantiations of the same feature type within the transitive complement closure, since only the lowest, i.e. most deeply embedded, instantiation can be checked off without leading to a crashing derivation. This connection certainly deserves more attention and has to be elaborated quite carefully.¹⁷

To end on a more linguistic note, we observe that constraint interaction among LCs (ILCs/CLCs) and its impact on the generative capacity of grammars is still deplorably understudied and consequently not very well understood. Our own attempts here have obviously been rather sketchy. Further research will have to provide an exact characterization of the LC/complexity connection. This would involve a clearer picture of the tight relation between the SMC and LCFRSs, which has guided much of the research on the complexity of MGs.¹⁸

¹⁵ This is at least true, when—by means of an “MG-external” encoding—we treat type 0-languages as recursively enumerable subsets of the natural numbers, because what Kobele concretely proves, is that each arbitrary *abacus* in the sense of [13] can be simulated by a corresponding MG. How to define such an MG directly deriving a given type 0-language, seems to be an open problem.

¹⁶ Note also that “permitting percolation of unchecked features of the attracted head into the attracting one,” “representing head-features as strings” and, depending on this representation, “demanding a left-to-right-checking of features” should be seen as properties of a particular instantiation of a slightly more general case still implying the same result on generative capacity (cf. [11]). Here we concentrate on this particular instantiation, just with the intend of keeping our exposition somewhat simpler and more accessible.

¹⁷ To support our conjecture, it should also be mentioned here that Kobele [10] pointed out, how in a different framework, namely, *mirror theoretic grammars* (MTGs) developed in [10] as a formalization of the syntactic theory proposed in [2], it is possible to define an (unrestricted) MTG deriving the language $\{a^{2^n} \mid n \in \mathbb{N}\}$. In fact, MTGs in their unrestricted version can be seen as strongly related to the MG-type Kobele considers in [11] exactly in the way they allow for feature percolation—though, in place of percolation from specifiers to heads, we are concerned with percolation from complements to (selecting) heads in the MTG-case—and the corresponding kind of percolation is employed to derive $\{a^{2^n} \mid n \in \mathbb{N}\}$ by an MTG.

¹⁸ Note that a *prima facie* problematic domain of grammar, namely, multiple-wh-fronting constructions (cf. [23]) can be harmonized with the SMC if, among other things, one assumes wh-cluster formation triggered by special clustering features.

References

1. Pierre Boullier. Proposal for a natural language processing syntactic backbone. Report No. 3342, INRIA research reports, INRIA Rocquencourt, 1998. Available at <http://www.inria.fr/rrrt/rr-3342.html>.
2. Michael Brody. Mirror theory. Syntactic representation in perfect syntax. *Linguistic Inquiry*, 31:29–65, 2000.
3. Noam Chomsky. On wh-movement. In P. Culicover, T. Wasow, and A. Akmajian, editors, *Formal Syntax*, pages 71–132. Academic Press, New York, NY, 1977.
4. Noam Chomsky. *Barriers*. MIT Press, Cambridge, MA, 1986.
5. Noam Chomsky. *The Minimalist Program*. MIT Press, Cambridge, MA, 1995.
6. Noam Chomsky. Beyond explanatory adequacy. MIT Occasional Papers in Linguistics (MITOPL #20), Massachusetts Institute of Technology, Department of Linguistics and Philosophy, Cambridge, MA, 2001.
7. Philippe de Groote, Glyn Morrill, and Christian Retoré, editors. *Logical Aspects of Computational Linguistics (LACL '01)*, LNAI Vol. 2099. Springer, Berlin, Heidelberg, 2001.
8. Henk Harkema. A characterization of minimalist languages. In de Groote et al. [7], pages 193–211.
9. Gerhard Jäger. On the generative capacity of multi-modal categorial grammars. *Research on Language and Computation*, 1:105–125, 2003.
10. Gregory M. Kobele. Formalizing mirror theory. *Grammars*, 5:177–221, 2003.
11. Gregory M. Kobele. Features moving madly. *Research on Language and Computation*, to appear. Draft version available at <http://www.linguistics.ucla.edu/people/grads/kobele/papers.htm>.
12. Hilda Koopman and Anna Szabolcsi. *Verbal Complexes*. MIT Press, Cambridge, MA, 2000.
13. Joachim Lambek. How to program an (infinite) abacus. *Canadian Mathematical Bulletin*, 4:295–302, 1961.
14. Jens Michaelis. Derivational minimalism is mildly context-sensitive. In M. Moortgat, editor, *Logical Aspects of Computational Linguistics (LACL '98)*, LNAI Vol. 2014, pages 179–198. Springer, Berlin, Heidelberg, 2001.
15. Jens Michaelis. Observations on strict derivational minimalism. In *FGMOL '01. Preproceedings*. Joint conference of the 6th conference on Formal Grammar and the 7th meeting of the Association for Mathematics of Language, Helsinki, 2001.
16. Jens Michaelis. *On Formal Properties of Minimalist Grammars*. PhD thesis, Potsdam University, Potsdam, 2001.
17. Jens Michaelis. Transforming linear context-free rewriting systems into minimalist grammars. In de Groote et al. [7], pages 228–244.
18. Jens Michaelis. Implications of a revised perspective on minimalist grammars. Draft, Potsdam University, 2002. Available at <http://www.ling.uni-potsdam.de/~michael/papers.html>.
19. Michael Moortgat. Multimodal linguistic inference. *Journal of Logic, Language and Information*, 5:349–385, 1996.
20. Glyn Morrill. *Type Logical Grammar*. Kluwer, Dordrecht, 1994.
21. Owen Rambow and Giorgio Satta. Independent parallelism in finite copying parallel rewriting systems. *Theoretical Computer Science*, 223:87–120, 1999.
22. John R. Ross. *Constraints on Variables in Syntax*. PhD thesis, MIT, Cambridge, MA, 1967.
23. Catherine Rudin. On multiple questions and multiple wh-fronting. *Natural Language and Linguistic Theory*, 6:445–501, 1988.

24. Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229, 1991.
25. Edward P. Stabler. Derivational minimalism. In C. Retoré, editor, *Logical Aspects of Computational Linguistics (LACL '96)*, LNAI Vol. 1328, pages 68–95. Springer, Berlin, Heidelberg, 1997.
26. Edward P. Stabler. Remnant movement and complexity. In G. Bouma, G.-J. M. Kruijff, E. Hinrichs, and R. T. Oehrle, editors, *Constraints and Resources in Natural Language Syntax and Semantics*, pages 299–326. CSLI Publications, Stanford, CA, 1999.
27. Mark Steedman. *Surface Structure and Interpretation*. MIT Press, Cambridge, MA, 1996.
28. K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the Association for Computational Linguistics (ACL '87)*, Stanford, CA, pages 104–111. ACL, 1987.
29. David J. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1988.

Large Scale Semantic Construction for Tree Adjoining Grammars^{*}

Claire Gardent¹ and Yannick Parmentier²

¹ CNRS, LORIA, Campus Scientifique BP 239, 54 Nancy

`Claire.Gardent@loria.fr`

`http://www.loria.fr/~gardent`

² INRIA, LORIA, Campus Scientifique BP 239, 54 Nancy

`Yannick.Parmentier@loria.fr`

Abstract. Although Tree Adjoining Grammars (TAG) are widely used for syntactic processing, there is to date no large scale TAG available which also supports semantic construction. In this paper, we present a highly factorised way of implementing a syntax/semantic interface in TAG. We then show how the resulting resource can be used to perform semantic construction either during or after derivation.

1 Introduction

Developing a Tree Adjoining Grammar which contains the information necessary for building the basic compositional semantics of sentences is a highly complex engineering task. To ensure consistency, ease of writing, of maintainance and of debugging, it is therefore important that this information be described at the appropriate level of abstraction. In the first part of this paper (sections 2 and 3), we show how to achieve a highly factorised integration of semantic information into a Tree Adjoining Grammar (TAG, [1]) using a particularly expressive grammar formalism recently developed by [2].

The second part of the paper shows how the resulting TAG can be used to support semantic construction that is, to associate the sentences generated by the grammar with a semantic representation. Contrary to other linguistic frameworks such as Lexical Functional Grammar, Head Driven Phrase Structure Grammar or Categorical Grammar, there is no clear consensus in TAG on how to perform semantic construction. This is because Tree Adjoining Grammar associates a derivation not with one, but with two structures namely, a derivation tree and a derived tree; and because it is unclear which of these two structures best supports semantic construction. As TAG elementary trees localise predicate-argument dependencies so that derivation trees resemble semantic dependency trees, the TAG derivation tree has long been taken to provide an appropriate

^{*} We would like to thank Benoit Crabbé, Denys Duchier, Djamé Seddah and Eric Villemonte de la Clergerie for many useful discussions on the themes discussed in this paper.

basis for semantic construction. Nevertheless, it has repeatedly been shown that the derivation tree alone does not provide all the information needed to perform semantic construction in all possible cases [3, 4, 5]; and that information from the derived tree also has to be taken into account.

In the second part of this paper, we show how the semantic TAG described in the first part can be used to support two types of semantic construction processes both of them being based on the information contained in the derived tree. The first method follows traditional unification based grammar practice and performs semantic construction during parsing (section 4) while in the second, semantic construction is done after parsing on the basis of a derivation forest thereby benefiting from the structure sharing supported by such packed representations (section 5). The resulting framework lays the basis for a systematic exploration of the relative efficiency of these two semantic construction methods for TAGs.

2 A TAG with a Unification-Based Syntax/Semantics Interface

In this section, we present the semantic TAG we use for semantic construction. Section 3 shows how to produce such a TAG on a large scale for a core fragment of French. Sections 4 and 5 show how to use it to perform semantic construction in two different ways.

2.1 Feature-Based TAG

In the approach we present here, semantic representations are combined using unification. To this end, we use a unification based version of LTAG namely, Feature-based TAG. A Feature-based TAG (FTAG, [6]) consists of a set of (auxiliary or initial) elementary trees and of two tree composition operations : substitution and adjunction. Substitution inserts a tree onto the leaf node of another tree¹ while adjunction (sketched in Fig. 1) inserts an auxiliary tree into a derived tree (i.e., either an elementary tree or a tree resulting from the combination of a derived tree with an elementary tree by means either of adjunction or of substitution).

In an FTAG, each tree node is associated with two feature structures called **top** and **bottom**. The **top** feature structure encodes information that needs to be percolated up the tree should an adjunction take place whilst the **bottom** one encodes information that remains local to the node at which adjunction takes place. During derivation, the unifications listed in Figure 2 take place.

2.2 Semantic Representation Language and Glue Mechanism

When doing semantic construction, two main questions arise : the choice of the semantic representation language and that of the “glueing” mechanism used for

¹ These leaf nodes must be marked for substitution and are graphically distinguished by a downarrow.

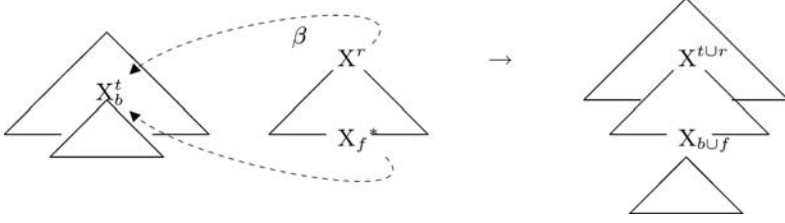


Fig. 1. Adjunction in FTAG

- The adjunction at some node X with **top** features t_X and **bottom** features b_X , of an auxiliary tree with root **top** features r and foot **bottom** features f entails the unification of t_X with r and of b_X with f .
- The substitution at some node X with **top** features t_X and **bottom** features b_X , of a tree with root **top** features t and root **bottom** features b entails the unification of t_X with t and of b_X with b .
- At the end of a derivation, the **top** and **bottom** features of all nodes in the derived tree are unified.

Fig. 2. Unifications in FTAG

putting semantic representations together. Mainly, semantic representations can be feature structures, lambda terms or terms of some underspecified logic whereas the available glueing mechanisms include unification, beta-reduction and linear logic.

The approach described here assumes a unification-based semantic construction process where semantic representations are flat semantic representations allowing for scope underspecification [7]. Importantly, the **semantic parameters** (that is, the semantic indices representing the missing arguments of the semantic functors) are represented by unification variables. As we shall see in the following section, the syntax/semantics interface is specified by the grammar in such a way that, as functors and arguments combine, semantic parameters are unified by the semantic construction process with the appropriate **semantic indices**.

For instance, the semantic representation for the semantic functor *every* and for its potential argument *cat* are as given in Example 1 and Example 2 where atoms starting with a capital letter are unification variables.

Example 1. $l_0 : \forall(X, h_1, h_2), h_1 \geq L_{restr}, h_2 \geq L_{scope}$

Example 2. $l_c : cat(Y)$

Combining these two representations using the grammar described in the sequel will yield the representation for *every cat* given in Example 3 where in particular, the restriction handle L_{restr} in the representation of *every* is unified with the label l_c in the representation for *cat* and the individual variable X in the representation of *every* with the variable Y in that of *cat*.

Example 3. $l_0 : \forall(X, h_1, h_2), h_1 \geq l_c, h_2 \geq L_{scope}, l_c : cat(X)$

For details about the semantic representation language used, we refer the reader to [5]. Note however that the choice of a particular semantic representation language and of a particular glueing mechanism is not particularly important here. Indeed the proposed approach could be applied to other semantic representation languages using some other glueing mechanism.

2.3 Modelling the Relation Between Syntax and Semantics

Syntax specifies which syntactic constituent provides the semantic argument for which semantic functor. To specify this mapping between syntax and semantics, (i) each elementary tree in the grammar is associated with a semantic representation of the type sketched above and (ii) the appropriate nodes of the elementary trees are decorated with semantic indices or parameters.

More precisely, the substitution nodes of the tree associated with a semantic functor will be associated with semantic parameters whilst root nodes and certain adjunction nodes will be labelled with semantic indices. As trees are combined, semantic parameters and semantic indices are unified by the FTAG unification mechanism thus specifying which semantic index provides the value for which semantic parameter. So for instance, the trees for *John*, *loves* and *Mary* will be as given in Figure 3. The tree for *loves* is associated with a semantic representation including the two semantic parameters x and y . These parameters also label the subject and the object substitution nodes of this tree. Conversely, the root node of the tree for *John* is labelled with the semantic index j . If the string parsed is *John loves Mary*, this tree will be substituted at the subject substitution node of the *loves* tree thus instantiating the semantic parameter x to j . And similarly, for the *Mary* tree.

As we shall see in sections 4 and 5, a TAG equipped with the syntax/semantic interface just described can be used to construct semantic representations either during or after derivation. In the first case, the unification variables present both on the tree nodes and in the semantic representations become instantiated as substitution and adjunction take place and the overall semantics of a sentence is the union of the semantic representations of the elementary trees entering in its derivation modulo unification. In the second case, a semantic lexicon is extracted from the grammar and used to do semantic construction on the basis of the derivation forest.

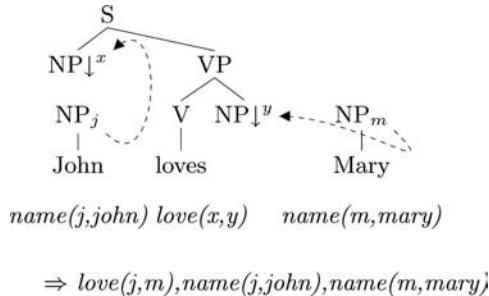


Fig. 3. *John loves Mary*

3 Grammar and Metagrammar: Factorising the Information

We now show how the metagrammar framework presented in [2] allows for a highly factorised specification of the mapping between syntax and semantics described in the preceding section. We start by presenting the grammar formalism used. We then show how it can be exploited to specify the syntax/semantics interface.

3.1 The Metagrammar Formalism

The metagrammar formalism presented in [2] can be seen as a generalisation of Shieber's PATR 2 language [8] which is expressive enough to encode (among others) Tree Adjoining Grammars. The goal of such languages is to provide a formalism that will allow a linguist to express her grammatical knowledge both directly and economically : the language must be expressive ; it must also allow for the factorisation of redundant information.

As space restrictions do not allow for a complete specification of this formalism, we restrict ourselves here to an informal presentation of the concepts needed for the rest of this paper. The interested reader is invited to refer to [2, 9, 10] for more details.

The metagrammar formalism (called XMG for eXtensible MetaGrammar) used supports both syntactic and semantic information.

In the syntactic dimension, tree fragments can be described that will be combined with other fragments to produce complete trees. These tree fragments can be referred to by means of *abstractions* (also called *classes*). Similarly, in the semantic dimension partial flat semantic formulae can be defined and referred to by means of abstraction thus also allowing for the factorisation of semantic information.

Syntactic and semantic abstractions can be combined using one of three operations namely, *conjunction* (to accumulate information), *disjunction* (to introduce non-determinism, used for instance to express diathesis) and *inheritance*. This last operation is used to specialise a class by incrementally adding pieces of information to a parent class. In our concrete syntax, conjunction, disjunction and inheritance are represented by `;`, `|`, and `import` respectively.

Finally, variables can be shared between classes in two main distinct ways. In the first case, the shared variables belong to classes linked by an inheritance relation and the scope of these variables can be explicitly managed using `import` and `export` declarations. In the second case, the shared variables belong to distinct inheritance chunks and sharing is made possible by a naming mechanism called **interfaces** which allow the global naming of a given value. For instance, in the class `Subj` below, the node `X` is named `sujNode` in the interface `*= [sujNode=X]` .

```

class Subj
declare ?X
{ <syn> { node [cat=s]
          node X [cat=n]
        } *= [subjNode=X]
}

```

The scope of an interface feature is global to its parent branch(es) in the hierarchy. As the next section will illustrate, the value of an interface feature can be shared by any other class by means of explicit variable sharing.

3.2 Specifying the Syntax/Semantics Interface

The main issue when developping a large scale semantic TAG is the correct specification of the mapping between syntax and semantics (cf. section 2.3). in [5], we define this mapping for a serie of syntactico-semantic constructions which are known to be problematic for TAG. Here however, we are concerned with the problem of how to design a *large scale* semantic TAG efficiently and economically. In this respect, verbs or more generally, semantic functors are of particular interest as they represent the bulk of the possible variations. We therefore concentrate on verbs and show how to specify the syntax/semantic interface for their various basic subcategorisation frames (transitive, intransitive, etc.), their various possible argument realisations (e.g., cliticisation, extraction, ommission) and their argument redistributions (active, passive, middle voice, impersonnal passive, etc.). Due to space restrictions, other types of syntactico-semantic constructions, although they can be handled by the grammar formalism used, will not be discussed here.

As was illustrated in section 2.3, the specification of the syntax/semantics interface consists in appropriately defining the mapping between grammatical functions (subject, object, etc.) and thematic roles (e.g., agent, patient or more neutrally, arg_1 , arg_2). For instance, in an active mood sentence with two nominal arguments, the subject NP is mapped to the first semantic argument (arg_1) and the object to the second (arg_2) whereas in a passive mood sentence, the inverse occurs so that the subject NP maps to arg_2 and the object to arg_1 .

In a TAG, a word is associated with the set of trees reflecting the range of syntactic configurations this word can occur in. For a verb (and more generally, for any type of syntactic functor), this set can be quite big. For instance, in the grammar for French developed by B. Crabbé [9], a transitive verb with nominal arguments is associated with 153 trees each describing a distinct possible syntactic environment for such a verb. More generally, Crabbé's core grammar for French totals roughly 3 500 trees for the verb fragment thereby covering 35 basic subcategorisation frames.

Clearly, the specification of the syntax/semantics interface needs to be factorised. We do not want to specify and maintain it for each of the 3 500 trees. To extend Crabbé's grammar with the syntax/semantics interface sketched in the preceding section, we proceed as follows :

1. While the semantic indices labelling the tree nodes are all values of an `idx` feature, they are also assigned a global name reflecting the grammatical function fulfilled by the node they label. For instance, the index x on the subject node of the active tree for *loves* in Figure 3 will be assigned the global name `subjectI`.
2. Similarly, the semantic indices occurring in the semantic representations are assigned a global name reflecting their thematic role. For instance, the first semantic argument of a binary relation is named `arg1`.
3. Finally, the mapping between grammatical functions and thematic roles is specified by coindexing the values of grammatical and thematic indices. For instance, in an active mode sentence tree, the value of `subjectI` will be coindexed with that of `arg1`.

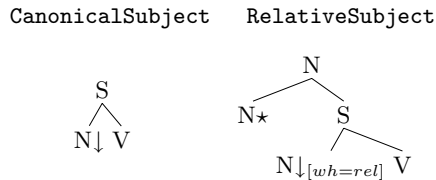
We now show how this works in more detail. We start by showing how the syntactic information is factorised in Crabbé’s grammar. We then go on to show how it can be extended with the syntax/semantics interface described in the previous section.

The Syntax. In Crabbé’s metagrammar [9, 10], the syntactic information associated with a TAG elementary tree is factorised along the following three dimensions.

First, grammatical function classes are defined which describe their structural properties. For instance, the `Subject` class is defined by the disjunction :

```
class Subject{
  CanonicalSubject | RelativeSubject | whSubject | ...
}
```

where each subclass (`CanonicalSubject`, etc.) is associated with the appropriate structural description² e.g.,



Next, alternations are defined in terms of grammatical functions and verbal morphology. For instance, the active and passive alternations for transitive verbs are defined by the following conjunctions of classes :

² To improve readability, we represent tree descriptions using graphics rather than logical formula. The precise tree language supported by XMG is described in [2].

```

class nOVn1Active{
  Subject ; Object ; activeVerbMorphology
}
class nOVn1Passive{
  Subject ; CAgent; passiveVerbMorphology
}

```

where `passiveVerbMorphology`, `activeVerbMorphology`, `Object` and `CAgent` are abstractions over the TAG structural objects associated with these linguistic notions.

Finally, the set of elementary trees associated with a given subcategorisation frame (e.g., `nOVn1`) is defined by the disjunction of its alternations e.g.,

```

class nOVn1{
  nOVn1Active | nOVn1Passive | nOVn1dePassive | nOVn1ShortPassive |
  nOVn1ImpersonalPassive | nOVn1middle
}

```

Augmenting the Metagrammar with Semantic Information. As mentioned above, augmenting the metagrammar with semantic information involves three steps.

First, the semantic indices labelling the tree nodes are named according to their grammatical function. For instance, the index labelling the subject node of the active tree for a transitive verb will be named `subjectI`. As shown below, this is done using an interface constraint : for each possible realisation of a subject, the value of the semantic index labelling the subject node is named `subjectI` by means of the interface constraint. In practice, the naming is done for a total of 12 grammatical functions (Subject, Object, Sentential-Subject, SententialCObject, SententialDeObject, SententialAObject, SententialInterrogative, Iobject, CAgent, Oblique, Locative, Genitive) and 56 realisations.

Class Name	CanonicalSubject	RelativeSubject
Structural description	<pre> S / \ N↓[idx=I] V </pre>	<pre> N / \ N★[idx=I] S / \ N↓[wh=rel] V </pre>
Interface constraint	<code>subjectI = I</code>	<code>subjectI = I</code>

Second, the semantic indices occurring in the semantic representations are named according to their thematic role. For instance, the first semantic argument of a binary relation is named `arg1`. This naming is again enforced by an interface constraint making the value globally accessible under that name.

```

class binaryRel
declare !L0 ?Rel ?E ?I1 !L1 ?I2 !L2
{
  <sem>{
    L0:Rel(E) ; L1:arg1(E,I1) ; L2:arg2(E,I2)
  }
  *=[rel=Rel,evt=E,arg1=I1,arg2=I2]
}

```

Third, the mapping between grammatical functions and thematic roles is specified by coindexing the relevant values.

Consider the class **n0Vn1** for instance, which describes the set of syntactico-semantic configurations associated in a TAG (for French) with verbs taking two nominal arguments. This set covers the configurations possible for the active mode, the long passive mode, the passive in *de*, the short passive, the impersonal passive and the middle form. For each of these modes, there are several possible configurations depending on how the arguments are realised (i.e., whether the subject/object/agent/etc. is canonical, cliticised, extracted, etc.) so that in total the class **n0Vn1** includes 153 trees. The mapping between syntax and semantics for these 153 trees is realised in the metagrammar by the labelling described above and by the following class definition :

```

class n0Vn1{
  binaryRel*=[evt=E,arg1=X,arg2=Y] ;
  { n0Vn1Active*=[subjectI=X,objectI=Y,vbI=E]
    | n0Vn1Passive*=[subjectI=Y,cagentI=X,vbI=E]
    | n0Vn1dePassive*=[subjectI=Y,genitiveI=X,vbI=E]
    | n0Vn1ShortPassive*=[subjectI=Y,vbI=E]
    | n0Vn1ImpersonalPassive*=[objectI=X,vbI=E]
    | n0Vn1middle*=[objectI=Y,vbI=E]
  }
}

```

That is, the set of syntactico-semantic configurations associated with the **n0Vn1** verbs is defined as consisting of (i) a binary semantic relation, (ii) trees realising the different possible verbal modes and grammatical functions realisations and (iii) a mapping between the semantic parameters of the binary relations and the semantic indices labelling the nodes of the trees. Typically, the first semantic parameter is identified with the subject semantic index in the active mode and with the object in the passive mode. If the verb is impersonal passive (*il est arrivé trois femmes*), this first parameter is identified with the semantic index of the object, etc.

In sum, the XMG formalism allows for a direct encoding of the linguistic notions necessary to specify the syntax/semantics interface : naming of the semantic indices labelling the tree nodes realising a given grammatical function, naming of the semantic parameters according to their thematic role and coindexing of the two types of indices. This expressivity in turn permits a clear and

economical encoding : the factorisation is high in that the relevant notions need only be encoded once but are used by many distinct classes. For instance, the labelling of the subject index is done once but is used by all of the 35 verb classes defined in our current TAG for French (since all verb classes make use of the subject class for their definition).

Using this encoding, a core TAG for French can be developed which encodes the semantic information necessary to support semantic construction. We now show how this information can be used in two different ways to compute the compositional semantics of a sentence during (or after) parsing.

4 Derived Tree and Semantic Construction

A first, simple way to construct semantic representations based on the semantic TAG described in the preceding section is to build these during derivation. Such an approach can be integrated in a TAG parser by simply associating the semantic representation of an elementary tree with the anchor node of that tree (cf. Figure 4). Since an anchor node never merges with any other node, there can be no conflict and the semantic representation remains untouched modulo the unification its indices can undergo via the coindexing with the tree nodes indices. The semantics of a derived tree is then the union of the values of the **semf** features present in this tree after unifications have taken place. For instance, if the trees of Figure 4 are combined to parse the string *Jean aime beaucoup Marie*, the resulting derived tree will be as given in Figure 5.

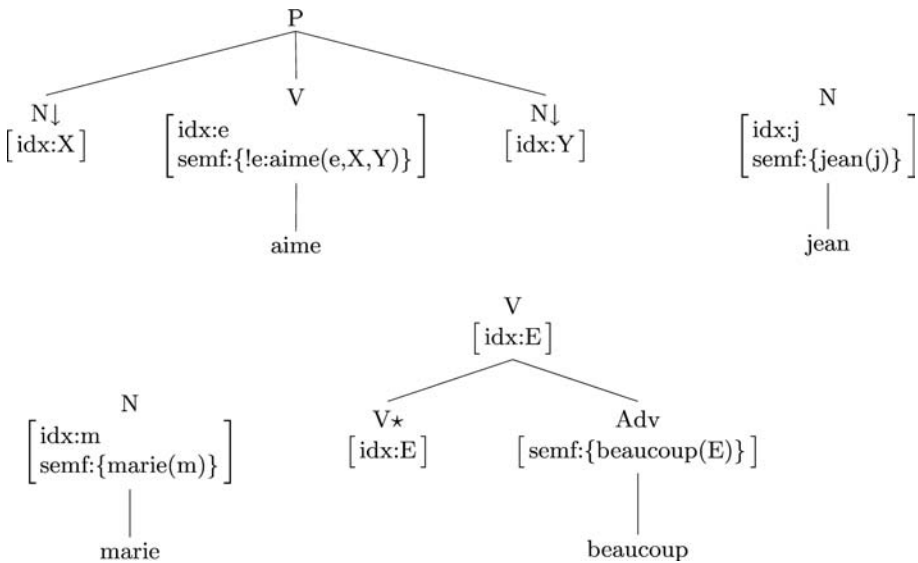


Fig. 4. TAG elementary trees with semantics included

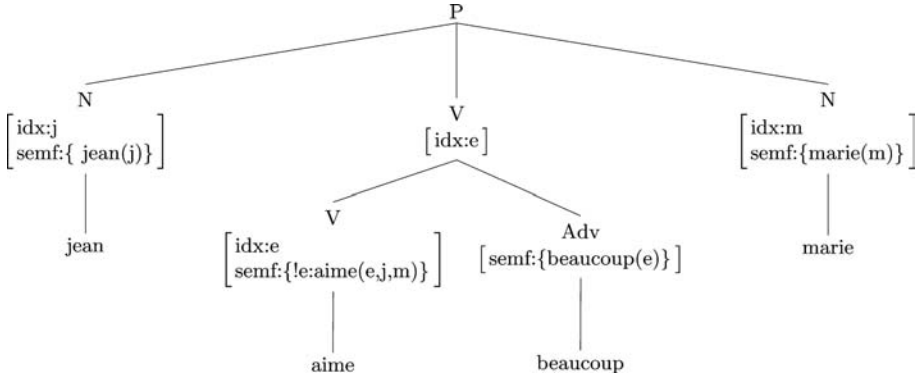


Fig. 5. TAG derived tree with semantics included

As mentioned above, the semantics associated with this tree is the union of the **semf** values after the TAG imposed unifications (cf. section 2) have taken place namely³

$$\{!e : aime(e, j, m), jean(j), marie(m), beaucoup(e)\}$$

5 Derivation Forest, Semantic Lexicon and Semantic Construction

As [5] shows, the semantic construction process described in the previous section accounts for data which an approach based on the derivation tree does not. However, the approach is open to two potential problems. First as mentioned in [11], the semantic representations included in the elementary and derived tree imply an infinite number of labels and individual variables so that in contrast to standard FTAG, the formalism is theoretically no longer equivalent to TAG. In practice of course, real sentences have finite lengths and so an upper bound could be specified which makes the set of feature values finite. Another difficulty however, is that the semantic information labelling the trees might decrease the amount of sharing in a tabular parsing approach and thereby decrease parsing efficiency.

To explore whether the second of these two objections is a real problem, we thus investigate a second way to do semantic construction where in essence, the semantic information is extracted from the TAG and used after parsing to reconstruct on the basis of the derivation tree the semantic representation of the sentence under consideration. This second way of doing semantic construction was first presented in [11]. We show here how it can be implemented on the basis of a standard TAG parser and of the semantic TAG produced by the XMG

³ The ! stands for the existential quantifier.

(cf. section 2). We start by giving a simplified example illustrating the workings of the approach. We then indicate first, how the required semantic lexicon can be automatically extracted from the semantic TAG described in section 2 and second, how semantic construction proceeds.

5.1 A Simple Example

A TAG derivation tree records how the elementary trees used to build a derived tree are put together using the two combining operations permitted by TAG namely, adjunction and substitution. Formally, the nodes of such a tree are labelled with tree names and its edges with a pair $\langle \text{Op}, \text{Id} \rangle$ where Op denotes the combining operation used to combine the trees labelling the vertices of the edge and Id identifies the node at which this operation takes place.

Now suppose that parsing the sentence *Jean court* yields the unique derivation tree pictured in Figure 6.

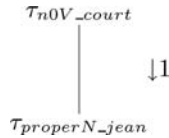


Fig. 6. Derivation tree for *Jean court*

And suppose further that the semantic lexicon extracted from the semantic TAG for *Jean* and *court* is as follows :

TreeName	<i>n0V</i>	TreeName	<i>properN</i>
Lemma	<i>court</i>	Lemma	<i>Jean</i>
SemRepr	!e :court(e,X)	SemRepr	jean(j)
ANodes	2.bot = [idx=e]	ANodes	
SNodes	1.top = [idx=X]	SNodes	
Root	0.bot = [idx=e]	Root	0.bot = [idx=j]

That is, the semantic information extracted from each elementary tree and stored in the semantic lexicon consists of the name of that tree, a record of the lemma anchoring that tree, the semantic representation associated with that tree and a record of the semantic information associated with the nodes (substitution nodes, nodes where adjunction can take place, root and foot nodes) of that tree.

Semantic construction then proceeds by traversing the derivation tree, collecting the lexical semantics associated in the semantic lexicon with each tree present in the derivation tree and performing the unifications imposed by a TAG derivation (cf. Figure 2). In this case, collecting the lexical semantics associated with the two trees occurring in the derivation tree yields :

$$\{ !e : \text{court}(e, X), \text{jean}(j) \}$$

Two unification steps are furthermore involved. The first follows from the substitution of $\tau_{properN_jean}$ at node 1 of τ_{n0V_court} which entails the unification of the feature structures of the root node of $\tau_{properN_jean}$ with those of node 1 in τ_{n0V_court} :

$$\begin{aligned} 1.top &= 0.top = [idx=X] \\ 1.bot &= 0.bot = [idx=j] \end{aligned}$$

The second unification step follows from the requirement that at the end of a TAG derivation the top and bottom feature structures of all nodes in the derived tree be unified. This requirement entails in particular the following unification :

$$\begin{aligned} 1.top &= 1.bot \\ 0.top &= 0.bot \end{aligned}$$

As a result $[idx=X]$ unifies with $[idx=j]$ and the overall semantics of *Jean court* becomes :

$$\{ !e : court(e, j), jean(j) \}$$

5.2 Extracting a Semantic Lexicon from the Semantic TAG

As the above example illustrates, the information required to perform semantic construction on the basis of a derivation forest consists of : a treename, a lemma, a semantic representation and four sets of path equations relating derived tree nodes with the semantic information labelling these nodes. One set of equations pertains to substitution nodes, another to nodes where adjunction can take place, the third to the root node and the fourth to the foot node if any.

That is, for each elementary tree present in the grammar, an entry is added to the semantic lexicon which contains the above information. Note further that the TAG used for parsing does not need to include any semantic information : all the semantic processing is done after parsing has taken place and relies only on the information contained in a (purely syntactic) derivation forest and in the semantic lexicon.

To automatically extract the required semantic lexicon from the semantic TAG G described in section 2, we proceed as follows :

1. For each tree T in G , all the nodes of T are numbered with their gorn addresses so that the nodes of the resulting grammar GG are then labelled both with semantic indices and with a gorn address.
2. For each tree T in GG :
 - (a) create a tree ST by erasing on all nodes of T the semantic information (if any) labelling that node. Call the resulting purely syntactic grammar SG
 - (b) create an entry in the semantic lexicon which contains : the tree name, the semantic representation associated by the metagrammar with this tree, the gorn addresses and the semantic information labelling the tree nodes

5.3 Computing Semantic Representations

As [12] shows, computing semantic representations from a parse forest is a natural way to deal with the combinatorial explosion that can result from enumerating all the readings of a given sentence : by doing semantic construction on the basis of the parse forest rather than the derivation trees, these shared syntactic constituents that have a single reading can also be shared during semantic construction. When combined with the use of an underspecified semantic representation language, such an approach allows for a large amount of structure sharing thereby increasing efficiency.

We now show how the semantic lexicon which, as shown in section 5.2 can be automatically extracted from the semantic TAG described in section 2, can be used in conjunction with a derivation forest to construct semantic representations.

A derivation forest is a compact representation of the derivation trees resulting from a sentence parse. It can be represented either by an and-or graph or by a context free grammar and its precise format may vary depending on the degree of sharing required [13]. Here we assume a CFG format where rules are of the form :

$$\begin{aligned} DTNodeId : : ElTreeId & \leftarrow (DTNode/Op.Node)^+ \\ ElTreeId : : Lemma.TreeName & \end{aligned}$$

with *DTNodeId*, *DTNode* identifying nodes in the derivation tree, *ElTreeId* identifying the elementary tree labelling a derivation tree node, *Op* being either *s* for *substitution* or *a* for *adjunction* and *Node* specifying the node in the elementary tree at which *Op* takes place.

To perform semantic construction, we simply traverse the derivation forest top-down, tabulating the constituents found and checking before constructing an item that it is not already included in the table built so far. For a given derivation tree in the parse forest, semantic construction is performed by a recursive descent through the tree as follows.

To construct the semantics *Sem* of a derivation tree with root *DTNodeId* given the parse forest rule $DTNodeId : : ElTreeId \leftarrow Dtrs$ **do**

$$\begin{aligned} Lemma.TreeName & \leftarrow terminal(DTNodeId) \\ HeadSem & \leftarrow lexSem(Lemma.TreeName) \\ SemDtrs & \leftarrow dtrsSem(HeadSem, Dtrs) \\ Sem & \leftarrow HeadSem + SemDtrs \end{aligned}$$

where *terminal* is a procedure mapping each derivation tree node to the terminal node it directly or indirectly rewrites as within the parse forest ; *lexSem* is a function retrieving from the semantic lexicon described in the preceding section, the lexical semantics associated with a given $\langle Lemma, TreeName \rangle$ pair ; *dtrsSem* is a procedure (described below) constructing the semantic representation of the daughters of a rule given the head semantics of its lhs ; and *+* denotes the operation accumulating the semantic representations being built. The *dtrsSem* procedure is defined as follows.

To construct the semantic representation *Sem* of the daughters *DTNodeId/Op.NodeId* | *ODtrs* of a rule given the head semantics *HeadSem* of its lhs, **do**

<i>Lemma.TreeName</i>	\leftarrow <i>terminal(DTNodeId)</i>
<i>HeadSemD1</i>	\leftarrow <i>lexSem(Lemma.TreeName)</i>
<i>tagUnify(HeadSem,HeadSemD1)</i>	
<i>semODtrs</i>	\leftarrow <i>dtrsSem(HeadSem,ODtrs)</i>
<i>Sem</i>	\leftarrow <i>HeadSemD1 + semODtrs</i>

where *tagUnify* performs the unification operations imposed on TAG derivations (cf. Figure 2) on the node labels provided by the semantic lexicon described in the previous section.

6 Conclusion

The proposal described in this paper is partially implemented. A core TAG for French is available which extends the syntactic TAG described in [9, 10] to include semantic information as described in sections 3 and 2. Semantic construction during derivation is currently being implemented whilst semantic construction after derivation has been implemented using the above grammar, an XSLT style sheet to extract the semantic lexicon and a prolog module to perform semantic construction on the basis of this semantic lexicon and of the derivation forest produced by Eric de la Clergerie's Dyalog TAG parser.

The resulting framework thus supports the comparative evaluation of the two semantic construction procedures for TAG as well as the development and testing of large scale semantic TAGs for French. Future work will focus on comparing the relative efficiency of these two semantic construction procedures; extending the grammar to include further types of alternations and in particular those described in [14] and the LADL tables; and experimenting with different semantic representation languages and glueing mechanisms.

Références

1. Joshi, A.K., Schabes, Y. : Tree-Adjoining Grammars. In Rozenberg, G., Salomaa, A., eds. : Handbook of Formal Languages. Springer (1997) 69–123
2. Duchier, D., Le Roux, J., Parmentier, Y. : The metagrammar compiler : An nlp application with a multi-paradigm architecture. In : Second International Mozart/Oz Conference - MOZ 2004, Charleroi, Belgique. (2004)
3. Frank, A., van Genabith, J. : GlueTag. Linear Logic based Semantics for LTAG. In Butt, M., King, T.H., eds. : Proceedings of the LFG01 Conference, Hong Kong (2001)
4. Kallmeyer, L. : Using an Enriched TAG Derivation Structure as Basis for Semantics. In : Proceedings of TAG+6 Workshop, Venice (2002) 127–136
5. Gardent, C., Kallmeyer, L. : Semantic construction in ftag. In : Proceedings of the 10th meeting of the European Chapter of the Association for Computational Linguistics, Budapest, Hungary (2003)

6. Vijay-Shanker, K., Joshi, A. : Feature structures based tree adjoining grammars. In : Proceedings of COLING, Budapest, Hungary (1988) 714–719
7. Copestake, A., Lascarides, A., Flickinger, D. : An algebra for semantic construction in constraint-based grammars. In : Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, Toulouse, France (2001)
8. Shieber, S. : An Introduction to Unification-based Approaches to Grammar. CSLI Lecture Notes (1986)
9. Crabbé, B., Duchier, D. : Metagrammar redux. In : International Workshop on Constraint Solving and Language Processing - CSLP 2004, Copenhagen. (2004)
10. Crabbé, B. : Grammatical development with XMG. Submitted to LACL05 (2005)
11. Kallmeyer, L., Romero, M. : Ltag semantics with semantic unification. In : Proceedings of the 7th International Workshop on Tree Adjoining Grammar and Related Formalisms, Vancouver, BC, Canada (2004) 155–162
12. Schiehlen, M. : Semantic construction from parse forests. In : Proceedings of the 16th International Conference on Computational Linguistics, Copenhagen (1996)
13. Alonso, M.A., Villemonte de la Clergerie, E., Diaz, V.J., Vilares, M. : 1. In : Relating Tabular Parsing Algorithms for LIG and TAG. Kluwer Academic Publishers (2002) to appear, revised notes of a paper for IWPT2000.
14. Saint-Dizier, P. : Alternation and verb semantic classes for french : Analysis and class formation. In : Predicative forms in natural language and in lexical knowledge bases. Kluwer Academic Publishers (1999)

A Compositional Approach Towards Semantic Representation and Construction of ARABIC

Bassam Haddad and Mustafa Yaseen

Faculty of Information Technology, University of Petra
P.O. BOX 3034, 11181 Amman, Jordan
{haddad, myaseen}@uop.edu.jo

Abstract. In spite of the fact that Arabic offers a well-studied theoretical and historical linguistic knowledge, unfortunately, it has so far received very little computational research and in particular on the level of semantic analysis. The most computational research efforts have been focused on morphological and syntax analysis, whereas research on Arabic computational semantics has been neglected. The main goal of this paper is to characterize the fundamental issues involved in deep logic-based semantic representation of Arabic sentences. The focus of attention of this work is relying on the principle of compositionality for Arabic semantic analysis, utilizing λ -calculus and type theory analysis of some Arabic syntactical constituents for achieving a semantic construction model for a fragment of ARABIC. Since semantic representation has to be compositional in Natural Language Understanding Systems, this approach offers a central concept for developing more intelligent and robust Arabic NLP systems.

1 Introduction and Motivation

The status of research on Computational Arabic is very limited compared with English and other European languages, which have already benefited from the extensive research in this field. For the last two decades, concentration on Arabic Language Processing has been focused on the processing of the *structure* of the language from the *morphological* and *syntactical* points of view.

Despite the essence of these aspects for the NLP, achieving Arabic *understanding* requires actually a differentiated and deep semantic processing.

Our research was initially directed towards building a comprehensive framework for Arabic Language Processing electronically. The first stages of this project addressed the *morphological including spellchecking* and the *syntactic* aspects of the language; a *Term Based Translator for the financial field*, a *Morphological Analyzer* and *Spell Checker* were developed [16], [24].

This paper addresses issues involved in deep semantic analysis of Arabic and tries to put fundamentals for the semantic representation. Based on the compositionality of some Arabic syntactical constituents utilizing λ -calculus and type theoretical analysis of Arabic structure, a conceptual model for constructing meaning representation of Arabic sentences, will be presented.

In the following, we will discuss the Computational Semantics of Arabic and its relationship to predicate logic first order as a meaning representation formalism. Furthermore, we will present our view of *Arabic Generalized Quantifiers* and their semantics as a good possibility to establish compositional rules.

1.1 Computational Semantics of Arabic

Computational Semantic of human languages is a non-trivial problematic issue of natural language processing. *Artificial Intelligence* had a long time ago recognized the *necessity* of performing some *semantic inferences* to achieve *human language understanding*. Unfortunately, as mentioned above, despite the significance of this issue, semantic processing based on logical models in the case of Arabic has so far received very little attention.

Meanwhile, many Arabic morphological analyzers have been successful in solving morphology related issues. Syntax on the other hand has been addressed by many researchers and some success has also been achieved there [1], [2], [7], [10], [21], [22] and many others.

There were few works reported on the *knowledge representation* and on the *computational semantic* of Arabic. Most of the reported works treated this problem superficially [3], [4], [5], [9], [11], [12]. Semantic analysis and in particular the problem of the compositionality of Arabic has so far not been treated deeply, neither linguistically nor logically [14], [15].

One of the main factors for this negligence might reside in the *complexity* of this field and, in the *invisible collaboration between Artificial Intelligence, Arabists, Logisticians and Linguisticians*. Therefore, we believe, that there is a crucial need to *reconsider* an adequate model for *understanding* and particularly for *the semantic processing of Arabic*.

In spite of the fact, that so far no existing formal theory of semantics is able to provide a complete and consistent account of all the phenomena of Arabic and the natural language in general, it remains beneficial to develop a model for semantic processing even if that model is imperfect and incomplete.

2 ARABIC Semantic Processing

Semantic processing has to carry out different necessary semantic tasks in interrelated and sometimes interchangeable semantic levels to achieve the understanding capability: *Semantic Composition, Semantic Resolution and Semantic Evaluation*. *Semantic Composition* can be viewed as the process of construction of meaning representation for capturing the semantic potential of Arabic sentences. *Semantic Resolution and Semantic Evaluation are more concerned with* disambiguation under using context knowledge and scoping rules and extracting of relevant information based on performing some deductions and inferences on the semantic representation of a proposition[14], [23].

In this work, we will focus the attention on the fundamentals involved in the compositionality of Arabic syntactical constituents.

2.1 Compositionality of Arabic Propositions

The key idea underlying compositional approaches is that the meaning of a sentence can be composed from the meaning of its syntactical constituents. As mentioned previously, a semantic formalism has to be *compositional* on the level of semantic representation to assure the modularity, declarativity and its practical employment in robust natural language understanding systems.

Despite the fact that predicate logic corresponds to well-studied and well-understood formal representation formalisms, it does not provide any compositional methods. Based on the *type theory*, λ -calculus offers a standard framework for filling the most important aspects of this gap [20].

In spite of the importance of the Montagovian approach in the computational linguistics, these methods are dealing with the *semantics of sentences*. One of the most important methods for capturing such problems involved in *text anaphoric* represents the *Discourse Representation Theory (DRT)*. Combination of DRT with λ -calculus leads to a compositional framework that is able to capture such problems [8], [18], and [19]. This approach is beyond the scope of this paper. In this paper¹, we are more concerned with Arabic natural language understanding in the context of constructing semantic representation of Arabic sentences by the means of employing of λ -calculus for constructing *logical formulas* acting as meaning representation for Arabic sentences.

2.2 Semantic Representation of Arabic

There are many reasons to choose a logical language as a *target language* for the meaning representation. Logic represents a *well-known meaning representation formalism* that differentiates between *syntax* and *semantics*. In addition, it enables *inferences* over quantified descriptions, which are basic requirements for an adequate meaning representation for any natural language.

Furthermore representing Arabic sentences as *logic programs* has the *facility* of performing some *semantic reasoning* tasks on a code based on Arabic predicates. Therefore, we believe that embedding logical formulas with Arabic predicates is a very *interesting aspect* of logic programming in the context of understanding Arabic. For example, formulas like (2.1) offer more flexibility and are more declarative in performing some semantic tasks on Arabic sentences:

$$[\exists x. \exists y. (\text{طالب}(x) \wedge \text{يدرس}(x, y)) / \exists x. \exists y. (\text{student}(x) \wedge \text{study}(x, y))] \quad (2.1)$$

In general, a semantic formalism designed for practical use, has to satisfy some important methodical principles and constraints. These basics include *Compactness*, *Modularity*, *Generality*, *Expressive Power*, *Efficiency*, *Implementation Independence*, and *Theory Independence* [8], [17]. It should be obvious that such constraints are guidelines rather than absolute criteria for the design of meaning representation formalism and therefore logic is covering these criteria in a large amount in the case of Arabic.

¹ A λ -DRT based Compositional Approach for Arabic is found in [15].

As Arabic is based on *verb-noun in Verbal Sentences*, and on *noun-noun opposition in Nominal Sentences*, we can establish a semantic correspondence between Arabic sentences and the first order predicate logic (PL1) formulas. This concept means, that the selection of logic as a meaning representation formalism for Arabic is capable of covering the most important constraints, which are expected to be satisfied from a meaning representation formalism.

The verb as the head of an Arabic Verbal Sentence (VS) and its complements, or the *الخبر* /; i.e. the nominal predicate as the head of an Arabic Nominal Sentence (NS), can be assigned to a predicate argument-structure of the corresponding PL1 formula. An Arabic Nominal Sentence can be expressed by using constants or by using quantified arguments of some predicates identifying the role of the subject or the object and other semantic roles.

To interpret logical formulas, we need model theoretically an *indirect denotation* function $\llbracket \cdot \rrbracket_g^{\mathcal{M}}$ according a model $\mathcal{M} = (\mathcal{D}, \mathcal{F})$ and a variable denotation \mathcal{G} or simply a denotation $\llbracket \cdot \rrbracket$, where \mathcal{D} represents the Domain and \mathcal{F} represents the involved semantic assignment mapping.

As our approach is proceeding from the perspective, that Arabic syntactical constituents are able to exhibit *relevant compositional rules* to construct a semantic representation for the most important Arabic sentence structures, the denotation $\llbracket \cdot \rrbracket$ also has to be compositional. By indirect denotation, we mean, that our compositional approach might need to transform an Arabic proposition in a meta or intermediate logical form containing higher order logical formulas, which have to be transformed in a PL1 based representation language as a target meaning representation form.

2.1.1 Representation of Lexical Entries

On the lexical level, an interpretation process might although need some *conceptual knowledge* and some *pragmatic contents* in form of lexical semantic knowledge or rules to supplement the meaning and to explain the possible word sense potentials of some Arabic natural propositions in a *specific domain*. For example, interpreting of concepts like *دراسة* /, studying) or some events such as *يَقْل* /, he | she | it transports²) might need some *lexical semantic knowledge* and *pragmatic annotations* about their *mode*, *involved objects and their roles*, *complements*, *compositional structure and time*. This knowledge base can be viewed as kind of a terminology or an ontology describing the involved events and their deep thematic roles including their compositionality encoded in the lexicon [24], [14].

For example, in Arabic verbs are intransitive, transitive or di-transitive and therefore, their current argument structure might depend on their contextual interpretation. The lexical semantic denotation of the transitive verb *يُدْرُس* /, study|he|it studies) using Arabic thematic role notation might be denoted by:

² In this case, the subject pronoun is incorporated in the verb itself.

$$\begin{aligned}
\left[\left[\begin{array}{c} (/يُدْرِس/, study) \\ CAT \quad V_t \end{array} \right] \right] &\equiv (/ \lambda y. \lambda x. \exists e. (يُدْرِس(e, x, y) \wedge حدث(e, دراسة) \wedge فاعل(e, x) \wedge \\
&\quad مفعول(e, y) \wedge [\text{selectional restriction}]) /, \\
&\quad \lambda y. \lambda x. \exists e. (study(e, x, y) \wedge Event(e, studying) \wedge Actor(e, y) \\
&\quad \wedge Object(e, y) \wedge [\text{selectional restriction}])) \quad (2.2)
\end{aligned}$$

The “selectional restriction” is used to enhance the thematic roles by allowing the lexical entries to place certain semantic restrictions on the lexemes and phrases that can accompany them in an Arabic sentence in form of basic common sense and domain specific knowledge. The actor of the meaning of the verb $\left[(/يُدْرِس/, study) \right]$, can be restricted, for instance, to the category student and human and so forth.

Arabic Nouns and Adjectives are principally considered as basic words, which can be interpreted as single argument predicates:

$$\left[(/اسم/, Noun) \right]^3 \equiv \lambda x. \left[\left[(/اسم/, Noun) \right] \right] (x) \quad (2.3)$$

$$\left[(/صفة/, Adj.) \right] \equiv \lambda x. \left[\left[(/صفة/, Adj.) \right] \right] (x) \quad (2.4)$$

A noun in (2.3) means that there is something, which can have the property of being the meaning of $\left[(/اسم/, Noun) \right]$, which represents usually the *canonical form* of the noun itself.

In the following, we will concentrate our presentation on the compositional potential of Arabic syntax.

3 The Analysis of the State of Definite in Arabic and Generalized Arabic Quantifiers

The Arabic article (/ال/, The) can be understood as a determiner. In the standard analysis of determiners in the type theory, an article can be considered as determiner. Determiners are generally of type $\langle \langle e, t \rangle, \langle \langle e, t \rangle, t \rangle \rangle$; i.e. it is a function taking an expression of type $\langle e, t \rangle$ to deliver an expression of type $\langle \langle e, t \rangle, t \rangle$, which has to be applied to an entity of type e in order to deliver the truth value t . This process can be expressed using λ -calculus to produce a compositional framework for Arabic sentences [14].

³ Compare it with (3.3), (3.4) and (3.8).

Considering the Arabic article (/ال/, The) as being a determiner is of type $\langle\langle e, t \rangle, \langle\langle e, t \rangle, t \rangle\rangle$ and regarding that it is expressing a singular entity would result in the following interpretation:

$$\left[\left[\begin{array}{cc} (/ال/, \text{The}) \\ \text{CAT} & \text{DET} \\ \text{ARG} & [\text{NUM} \quad \text{sing}] \end{array} \right] \right] \text{ as } \left[(/ال_I /, \text{The}_1) \right] \text{ where} \quad (3.1)$$

$$\left[(/ال_I /, \text{The}_1) \right] \equiv \lambda P \lambda Q \exists x. (\forall y (P(y) \leftrightarrow x = y) \wedge Q(x)) \quad (3.2)$$

$\left[(/ال_I /, \text{The}_1) \right]$ expresses, that there exists only one thing of being P and Q, which implies that the *cardinality* of P has to be 1; i.e. $|P|=1$ and $P \cap Q \neq \emptyset$.

On the other hand, nominal phrases such as (/الطالب/, The student) can be regarded as quantifier. We proceed from the perspective that quantifiers logically correspond to noun phrases and not to determiners and they denote families of sets. They are used to assert that a set has some property [6]. For example, “ $\exists x \alpha(x)$ ” asserts that the set of things which satisfy $\alpha(x)$; i.e. $\{x/\alpha(x)\}$ is a nonempty set.

The Arabic indications⁵ of *indefinite articles* such as *the tanween* (/ /) in the ending of a nominal in the nominative such as in (/طالبٌ /, (a) student) in (3.3) can be interpreted as \exists -quantifier.

$$\left[\begin{array}{cc} (/طالبٌ /, (a) \text{ student}) \\ \text{CAT} & \text{Noun} \\ \text{ARG} & \left[\begin{array}{cc} \text{NUM} & \text{sing} \\ \text{GEN} & \text{fem} \\ \text{STATE} & \text{indef} \end{array} \right] \end{array} \right] \quad (3.3)$$

Since such *endings in most cases* are omitted in Arabic texts, we can directly consider the nouns, in this context, as noun phrases or logically as quantifiers:

⁴ The function of a definite determiner in the sense of the DRT cannot only be the unique quantification of an object.

⁵ In Arabic there are no indefinite articles such as the English “a”, “an” or the German “ein” or “eine” etc. Furthermore, the definite article (/ال/, The) is usually not separated from the noun.

$$\left[\left[\begin{array}{l} \text{CAT} \quad (/اسم/, Noun) \\ \text{ARG} \quad \left[\begin{array}{ll} \text{NUM} & sing \\ \text{STATE} & indef \end{array} \right] \end{array} \right] \right] \equiv \lambda Q . \exists x_{<1>}. (\left[(/اسم/, Noun) \right] (x) \wedge Q(x)), \quad (3.4)$$

where $\exists x_{<1>}$ quantifies one individual in the set describing the $(/اسم/, Noun)$.

The rule in (3.4) expresses that it exists one individual having the property of being an instance of the meaning of the Noun, i.e. the $\left[(/اسم/, Noun) \right]$

Based on the same principle, the quantifier as a noun phrase $(/الطالب/, The Student)$ can also be interpreted as follows:

$$\left[\left[\begin{array}{l} (/الطالب/ , The - Student) \\ \text{CAT} \quad (/اسم/, Noun) \\ \text{ARG} \quad \left[\begin{array}{ll} \text{NUM} & sing \\ \text{STATE} & def \end{array} \right] \end{array} \right] \right] \equiv \lambda Q. (/ال_1 (x)/, The_1(x)). ((/طالب(x)/ student (x)) \wedge Q(x)) \quad (3.5)$$

As Arabic differentiates between three kinds of numbers: *singular*, *dual*⁶ and *plural*, a noun phrase in dual form can also be considered as an Arabic quantifier satisfying the duality:

$$\left[\left[\begin{array}{l} (/ال/, The) \\ \text{CAT} \quad \text{DET} \\ \text{ARG} \quad \left[\begin{array}{ll} \text{NUM} & dual \end{array} \right] \end{array} \right] \right] \text{ as } \left[(/ال_2 /, The_2) \right] \text{ where} \\ \left[(/ال_2 /, The_2) \right] \equiv \lambda P. \lambda Q . (/ال_2 (x), The_2(x)). (P(x) \wedge Q(x)) \text{ and } |P| = 2, P \cap Q \neq \emptyset \quad (3.6)$$

For example $(/الطالبان/ the two students)$ could be interpreted as follows:

$$\left[\left[\begin{array}{l} (/الطالبان/ , The - two - Students) \\ \text{CAT} \quad \text{Noun} \\ \text{ARG} \quad \left[\begin{array}{ll} \text{NUM} & Dual \\ \text{STATE} & def \\ \text{CASE} & nom \end{array} \right] \end{array} \right] \right] \equiv \lambda Q. (/ال_2 (x)/, The_2(x)). ((/طالب(x)/, student(x)) \wedge Q(x)) \quad (3.7)$$

⁶ In Arabic, dual is not plural and a noun phrase in dual has its special cases, which can be derived from its morphology.

In the case of the *indefinite dual* state, we have also introduced the numeric determiner $\exists x_{<2>}$, expressing the restriction to only two individuals, which could have the property of being the meaning of the Noun, $\llbracket /اسم/, \text{Noun} \rrbracket$:

$$\left\llbracket \begin{array}{cc} \text{CAT} & (/اسم/, \text{Noun}) \\ \text{ARG} & \left[\begin{array}{cc} \text{NUM} & \text{dual} \\ \text{STATE} & \text{indef} \end{array} \right] \end{array} \right\rrbracket \equiv \lambda Q. \exists x_{<2>}. (\llbracket /اسم/, \text{Noun} \rrbracket(x) \wedge Q(x)) \quad (3.8)$$

4 Generalized Arabic Quantifiers

It is clear that the quantifiers of the standard first-order logic are inadequate to represent quantified Arabic sentences, and therefore we need to consider the so-called generalized quantifiers. In our case, we call them *Generalized Arabic Determiners or Quantifiers*.

We have encountered some difficulties in capturing the information expressed by the determiners and numerals in noun phrases. Arabic contains a large number of determiners, which so far have not been logically analyzed in the context of their compositional effect on the sentence structure.

In the analysis of the state of definites in Arabic, we adopt the view that quantifiers correspond to noun phrase and not to determiners. They, therefore, do not necessarily belong to the logical vocabulary. The truth or falsity of many natural language quantifiers does not depend on a priori logic but much more on the underlying model. Quantifiers or noun phrases such as (*كل طالب* /, every student), (*معظم الطلاب* /, most of the students) vary from model to model. On the other hand (*كل* /, every), as determiner is logical, whereas (*معظم ال* /, most of the) is not [6].

In the following, we will present some of these Arabic determiners.

Similar to the analysis of determiners in the case the of $\llbracket (/ال_1 /, \text{The}_1) \rrbracket$, a determiner differentiates between two things: a *restriction* “R” and a *scope* “S.” In (3.1), $P(x)$ represents the restricted set and $Q(x)$ the scope; i.e. the proposition about the restricted set.

Generally, a determiner can be expressed as

$$\llbracket \text{DET}_{\text{Arab}} \rrbracket \equiv \lambda R. \lambda S. \text{DET}_{\text{Arab}}(x). (R, S) \quad (4.1)$$

$\lambda R \lambda S$ expresses in (4.1) that there are formulas expressing the restriction and the scope of the determiner and they can be reduced by applying $\llbracket \text{DET}_{\text{Arab}} \rrbracket$ to R and S.

As mentioned previously, interpreting the truth-values of quantifiers requires some conceptual knowledge about the relationship between a restriction and its scope and their cardinalities in a specific domain.

For example, the determiner (/المعظم-ال/ , most-of-the) or (/معظم/ , most) should be true if the $|R \cap S|$ holds relatively a large portion of $|R|$:

$$\left[\begin{array}{cc} (/المعظم-ال/ , \text{most of the}) \\ \text{CAT} & \text{DET}_{\text{Arab}} \end{array} \right] \equiv \lambda R. \lambda S. (/المعظم-ال(x)/ , \text{most of the}(x)). (R, S) , \text{ where} \\ |R \cap S| > |R - S| \quad (4.2)$$

In the following, we will present some of the common Arabic Generalized Quantifiers:

$$\left[\begin{array}{cc} (/n - على-الأقل- / , \text{at - least - } n) \\ \text{CAT} & \text{DET}_{\text{Arab}} \end{array} \right] \equiv \lambda R. \lambda S. (/n - على-الأقل(x)/ , \text{at - least-}n(x)). (R, S) \\ \text{is true if } |R \cap S| \geq n \quad (4.3)$$

$$\left[\begin{array}{cc} (/أكثر-من-النصف / , \text{more than the half}) \\ \text{CAT} & \text{DET}_{\text{Arab}} \end{array} \right] \equiv \lambda R. \lambda S. (/أكثر-من-النصف(x)/ , \text{more-than-} \\ \text{half}(x)). (R, S) \text{ if } |R \cap S| > 0.5 * |R| \quad (4.4)$$

$$\left[\begin{array}{cc} (/قليل-من / , \text{little - of}) \\ \text{CAT} & \text{DET}_{\text{Arab}} \end{array} \right] \equiv \lambda R. \lambda S. (/قليل(x)/ , \text{little-of}(x)). (R, S) \quad (4.5)$$

and $|R \cap S| < r * |R|$ where r is a relative parameter determining when R should be little.

The determiners like (/جميع/ , total) and (/كل/ , all) can be interpreted as universal-quantifiers as follows:

$$\left[\begin{array}{cc} (/كل/ , \text{all, each}) \\ \text{CAT} & \text{DET}_{\text{Arab}} \end{array} \right] \equiv \lambda P. \lambda Q. \forall (x). (P(x) \rightarrow Q(x)) \quad (4.6)$$

Based on (4.1) (/كل/ , all) can also be expressed as Generalized Arabic Determiner:

$$\left[\begin{array}{cc} (/كل/ , \text{all, each}) \\ \text{CAT} & \text{DET}_{\text{Arab}} \end{array} \right] \equiv \lambda R. \lambda S. (/كل(x)/ , \text{all}(x)). (R, S) \text{ where } R \subseteq S \quad (4.7)$$

5 Rules for Construction Semantic Interpretation

In order to be able to compose logical formulas for Arabic sentences, we need to give a compositional meaning to structured syntactical categories, like VS and NS. We also need to assign meanings to the sub-syntactical categories of these types of sentences in form of semantic compositional rules. It is also to emphasize that at this stage of the semantic analysis, syntactical and semantic information has to be evaluated within such compositional rules.

5.1 Arabic Sentence Structure and Logical Forms

Arabic differentiates between different types of sentences: *Verbal Sentences (VS)*, *Nominal Sentences (NS)* and *Copulative Sentences*.

In contrast to European languages, a Verbal Sentence usually starts with a verb, and in most cases has a V-S-O structure; i.e. Verb-Subject-Object structure. The predicate of a NS usually is a noun, a pronoun, a propositional phrase or an adverb. The predicate of a VS is a verb and its complements. Copulative sentences have a Nominal Sentence or a Verbal Sentence as a predicate that is bound with the subject through a copulative pronoun. Copulative Sentences start with subjects [13].

In the following we will present a fragment of a grammar for construction of common Arabic sentences:

The meaning of a NS can be obtained, by applying the meaning of the opening nominal constituent $\llbracket \langle / \text{مبند} /, \text{Sub.NS} \rangle \rrbracket$; i.e. the subject of Nominal Sentences to the meaning of the predicate of Nominal Sentences $\llbracket \langle / \text{خبر} /, \text{Pr.NS} \rangle \rrbracket$:

$$\llbracket \langle / \text{مبند} /, \text{Sub.NS} \rangle \rrbracket \llbracket \langle / \text{خبر} /, \text{Pr.NS} \rangle \rrbracket \quad (5.1)$$

If the $\langle / \text{مبند} /, \text{Sub.NS} \rangle$ consists of a determiner and a noun; i.e. noun phrase, then $\llbracket \langle / \text{مبند} /, \text{Sub.NS} \rangle \rrbracket$ means the application of the meaning of such a determiner to the noun. The meaning of the whole NS can successively be achieved, by determining the meaning of $\llbracket \langle / \text{خبر} /, \text{Pr.NS} \rangle \rrbracket$ and the application of $\llbracket \langle / \text{مبند} /, \text{Sub.NS} \rangle \rrbracket$ to it:

$$\begin{aligned} R_{i1}: \langle \text{NS} \rangle &\rightarrow \langle \langle / \text{مبند} /, \text{Sub.NS} \rangle \rangle \langle \langle / \text{خبر} /, \text{Pr.NS} \rangle \rangle \\ \llbracket \text{NS} \rrbracket &\equiv \{ \llbracket \langle / \text{مبند} /, \text{Sub.NS} \rangle \rrbracket \llbracket \langle / \text{خبر} /, \text{Pr.NS} \rangle \rrbracket \} \end{aligned} \quad (5.2)$$

$$\begin{aligned} R_{i2}: \langle \langle / \text{مبند} /, \text{Sub.NS} \rangle \rangle &\rightarrow \langle \text{DET}_{\text{Arab}} \rangle \langle / \text{اسم} /, \text{Noun} \rangle | \langle / \text{مضاف} /, \text{Gen. N.P} \rangle | \dots \\ \llbracket \langle \langle / \text{مبند} /, \text{Sub.NS} \rangle \rangle \rrbracket &\equiv \{ \llbracket \text{DET}_{\text{Arab}} \rrbracket \llbracket \langle / \text{اسم} /, \text{Noun} \rangle \rrbracket \} | \dots \end{aligned} \quad (5.3)$$

$$\begin{aligned} R_{j1}: \langle \text{DET}_{\text{Arab}} \rangle &\rightarrow \left\langle \left[\begin{array}{c} \langle / \text{ال} /, \text{The} \rangle \\ \text{ARG} \quad [\text{NUM} \quad \text{sing}] \end{array} \right] \right\rangle | \left\langle \left[\begin{array}{c} \langle / \text{معظم} /, \text{most of the} \rangle \\ \text{ARG} \quad [\text{NUM} \quad \text{sing}] \end{array} \right] \right\rangle | \dots \\ &\quad \left\langle \left[\begin{array}{c} \langle / \text{قليل} /, \text{little-of} \rangle \\ \text{ARG} \quad [\text{NUM} \quad \text{sing}] \end{array} \right] \right\rangle | \dots \\ \left\langle \left[\begin{array}{c} \langle / \text{ال} /, \text{The} \rangle \\ \text{ARG} \quad [\text{NUM} \quad \text{sing}] \end{array} \right] \right\rangle &\equiv \{ \lambda R. \lambda S. \langle / \text{ال} (x) /, \text{The}_1(x) \rangle. (R, S) \} | \end{aligned} \quad (5.4)$$

$$\llbracket \langle \text{معظم-ال} / , \text{most of the} \rangle \rrbracket \equiv \{ \lambda R. \lambda S. (\text{معظم-ال}(x), \text{most-of-the}(x)). (R, S) \} | \dots$$

$$\dots \quad (5.5)$$

$$R_{k1}: \langle \text{مضاف} / , \text{Gen. N.P} \rangle \rightarrow \langle \text{مضاف} / , 1 - \text{G.Noun} \rangle \langle \text{مضاف اليه} / , 2 - \text{G.Noun} \rangle$$

$$\llbracket \text{مضاف} / , \text{Gen. N.P} \rrbracket \equiv \exists x. \exists y. (\llbracket \text{مضاف} / , 1 - \text{G.Noun} \rrbracket (x) \wedge$$

$$\llbracket \text{مضاف اليه} / , 2 - \text{G.Noun} \rrbracket (y) \wedge \llbracket G_1 . \text{Rel.} \rrbracket (x, y)) \quad (5.6)$$

$$R_{k2}: \langle \text{مضاف} / , 1 - \text{G.Noun} \rangle \rightarrow \langle \text{اسم} / , \text{Noun} \rangle \quad (5.7)$$

$$\llbracket \text{مضاف} / , 1 - \text{G.Noun} \rrbracket \equiv \llbracket \text{اسم} / , \text{Noun} \rrbracket$$

$$R_{k3}: \langle \text{مضاف اليه} / , 2 - \text{G.Noun} \rangle \rightarrow \langle \text{DET}_{\text{Arab}} \rangle \left\langle \begin{array}{c} \text{اسم} / , \text{Noun} \\ \text{CASE} \quad \text{Gen} \end{array} \right\rangle | \dots$$

$$\llbracket \text{مضاف اليه} / , 2 - \text{G.Noun} \rrbracket \equiv \llbracket \text{اسم} / , \text{Noun} \rrbracket \quad (5.8)$$

$$R_{k4}: \llbracket G_1 . \text{Rel.} \rrbracket \equiv \langle \text{جزء من} / , \text{Part-of} \rangle | \dots \quad (5.9)$$

$$R_{11}: \langle \text{خبر} / , \text{P.NS} \rangle \rightarrow \langle \text{اسم} / , \text{Noun} \rangle | \langle \text{صفة} / , \text{Adj.} \rangle | \dots$$

$$\llbracket \text{اسم} / , \text{Noun} \rrbracket \equiv \lambda x. \llbracket \text{اسم} / , \text{Noun} \rrbracket (x) \quad (5.10)$$

$$\llbracket \text{صفة} / , \text{Adj.} \rrbracket \equiv \lambda x. \llbracket \text{صفة} / , \text{Adj.} \rrbracket (x) \quad (5.11)$$

Example:

Let us consider the following NS: $\langle \text{الطقس جميل} / , \text{the weather beautiful}^7 \rangle$. The meaning of the $\langle \text{مبتدأ} / , \text{Sub.NS} \rangle$ “ $\langle \text{الطقس} / , \text{the-weather} \rangle$ ” is the application of the meaning of the determiner $\langle \text{ال} / , \text{The} \rangle$ in (5.4) to the meaning of the noun $\langle \text{طقس} / , \text{weather} \rangle$:

$$\llbracket \langle \text{ال طقس} / , \text{The weather} \rangle \rrbracket \Rightarrow:$$

$$\lambda R. \lambda S. (\langle \text{ال} \rangle (x), \text{The}_1(x)). (R, S)) \left(\llbracket \langle \text{طقس} / , \text{weather} \rangle \rrbracket \right)$$

$$\lambda S. (\langle \text{ال} \rangle (x), \text{The}_1(x)). (\lambda x. \langle \text{طقس} / , \text{weather} \rangle (x), S)) \quad (5.12)$$

⁷ Although such nominal phrases do not contain any verbs or substantiated verbs, they represent meaningful and complete Arabic Nominal Sentences.

Applying the meaning of the adjective (/جميل/, beautiful), which takes the role of (/خبر/, Pr.NS) as in (5,10) yields the meaning of the sentence (/الطقس جميل/, the weather beautiful):

$$\begin{aligned} & \llbracket / \text{الطقس جميل} /, \text{ the weather beautiful} \rrbracket \Rightarrow: \\ & \lambda S. ((/ال_1(x)/, \text{The}_1(x)). (\lambda x. (/طقس/, \text{weather})(x), S)) (\llbracket (/جميل/, \text{beautiful}) \rrbracket) \\ & ((/ال_1(x)/, \text{The}_1(x)). (\lambda x. (/طقس/, \text{weather})(x), \lambda x. (/جميل/, \text{beautiful})(x))), \quad (5.13) \end{aligned}$$

which can be interpreted for a specific x_1 as:

$$\begin{aligned} & \Rightarrow: \text{ال}_1(x_1). (\text{جميل}(x_1) \wedge \text{طقس}(x_1)) \text{ or} \\ & \Rightarrow: \text{The}_1(x_1). (\text{Weather}(x_1), \text{beautiful}(x_1)) \end{aligned} \quad (5.14)$$

Considering determiners as a *relation* between the restriction and the scope sets, requires the application of their meanings to the meaning of involved syntactical constituents. Since VS start with verbs, and if the (/فاعل/, Subject of a VS) is in definite state, the meaning of the verbal subject as quantifier can be achieved by applying its meaning to the meaning of the involved noun and the verbal predicate. The verb and its object can take the role of the scope of the determiner of the subject:

$$\begin{aligned} R_{m1}: \langle \text{VS} \rangle & \rightarrow \langle (/فاعل/, \text{Verb}) \rangle \left\langle \left[\begin{array}{cc} (/فاعل/, \text{Sub.VS}) \\ \text{CAT} & \text{Vt} \end{array} \right] \right\rangle \langle (/مفعول/, \text{Object}) \rangle | \dots \\ \llbracket \text{VS} \rrbracket & \equiv \left\{ \llbracket (/فاعل/, \text{Sub.VS}) \rrbracket \left(\llbracket (/مفعول/, \text{Object}) \rrbracket \left(\llbracket (/فاعل/, \text{Verb}) \rrbracket \right) \right) \right\} \end{aligned} \quad (5.15)$$

.....

$$\begin{aligned} R_{n1}: \langle (/فاعل/, \text{Sub.VS}) \rangle & \rightarrow \langle \text{DET}_{\text{Arab}} \rangle \langle (/اسم/, \text{Noun}) \rangle | \dots \\ \llbracket (/فاعل/, \text{Sub.VS}) \rrbracket & \equiv \left\{ \llbracket \text{DET}_{\text{Arab}} \rrbracket \left(\lambda x. \llbracket (/اسم/, \text{Noun}) \rrbracket \right) \right\} | \dots \end{aligned} \quad (5.16)$$

$$\begin{aligned} R_{n2}: \langle (/مفعول/, \text{Object}) \rangle & \rightarrow \langle \text{DET}_{\text{Arab}} \rangle \left\langle \left[\begin{array}{cc} (/اسم/, \text{Noun}) \\ \text{CASE} & \text{Acc} \end{array} \right] \right\rangle | \dots \\ \llbracket (/مفعول/, \text{Object}) \rrbracket & \equiv \left\{ \llbracket \text{DET}_{\text{Arab}} \rrbracket \left(\lambda y. \llbracket (/اسم/, \text{Noun}) \rrbracket \right) \right\} | \dots \end{aligned} \quad (5.17)$$

.....

The meaning of VS of the structure V-S-O is the application of $\llbracket (/فاعل/, Sub.VS) \rrbracket$ to the denotation of $\langle (/مفعول/, Object) \rangle$ and to the $\langle (/فعل/, Verb) \rangle$.

Example:

The meaning of the noun phrase (/ال طالب/, the-student) as $\langle (/فاعل/, Sub.VS) \rangle$ in the Verbal Sentence (/يُدرس الطالب الحاسوب/, the student studies the computer (science)) can be constructed as follows:

$$\llbracket (/ال-طالب/, the student) \rrbracket \Rightarrow \lambda S. ((/ال_1(x)/, The_1(x)). (\lambda x. (/طالب(x)/, student(x)), S))$$

$$(\llbracket (/مفعول/, Object) \rrbracket \quad (\llbracket (/فعل/, Verb) \rrbracket)) \quad (5.4)(5.15)$$

Applying of the denotation of the object $\llbracket (/مفعول/, Object) \rrbracket$ to the meaning of the verb using the lexical information about the verb (/يُدرس/, study) in (3.1) yields the following simplified semantic representation:

$$\llbracket (/يُدرس-ال-حاسوب/, studies the computer(science)) \rrbracket \Rightarrow:$$

$$(/ال_1(y). (computer(y), < \lambda x. \exists e. (يُدرس(e, x, y) \wedge حدث(e, دراسة) \wedge فاعل(e, x) \wedge$$

$$مفعول(e, y) \wedge \llbracket \text{the lexical selectional restriction} \rrbracket >)) /,$$

$$The_1(y). (computer(y), < \lambda x. \exists e. (study(e, x, y) \wedge Event(e, studying) \wedge Actor(e, y) \wedge Object(e, y) \wedge \llbracket \text{the lexical selectional restriction} \rrbracket >)))$$

Applying $\langle (/فاعل/, Sub.VS) \rangle$ to (5.18) yields: (5.18)

$$(/ال_1(x). (طالب(x), /ال_1(y). (حاسوب(y), < \exists e. (يُدرس(e, x, y) \wedge حدث(e, دراسة) \wedge فاعل(e, x) \wedge$$

$$مفعول(e, y) \wedge \llbracket \text{the lexical selectional restriction} \rrbracket >))) /,$$

$$The_1(x). (student(x), The_1(y). (computer(y), < \exists e. (study(e, x, y) \wedge Event(e, studying) \wedge Actor(e, y) \wedge Object(e, y) \wedge \llbracket \text{the lexical selectional restriction} \rrbracket >))))$$

(5.19)

6 Overview and Conclusion

In this paper, we tried to present some of our results towards constructing a compositional semantic model for deep semantic analysis of Arabic sentences. Our approach is based on utilizing λ -calculus and the compositionality to construct

semantic representations in form of logical formulas. In this context, we have introduced the Arabic Generalized Quantifiers concept within different types of Arabic sentences considering the order and meaning of some syntactical constituents of Arabic. As Arabic has received very little computational research and in particular, on the level of deep semantic analysis, we believe, that our contribution might encourage some computational linguisticians and researchers to put more efforts in this complex area of Arabic Natural Language Understanding. Meanwhile we are working on extending and embedding these results in a compositional Arabic Model considering the Discourse Representation Theory as a departure point to capture Arabic discourses and features involved in anaphora representations in form of a λ -DRT within a Unification based Grammar for Arabic.

References

1. Al-Fedaghi, Al-Anzi: A New Algorithm to Generate Arabic Root-Pattern Forms. Proceedings of the 11th National Computer Conference, Saudi Arabia, 1989
2. Ali, N.: Formalization and Computation of Arabic Syntax. Proceedings of the 11th National Computer Conference, Saudi Arabia, 1989
3. Al-Johar, B., McGregor, J.: A Logical Meaning Representation for Arabic (LMRA). Proceedings of the 15th National Computer Conference, Riyadh, Saudi Arabia, 1997
4. Al-Muhtaseb, H., Mellish C.: Towards an Arabic Upper Model: A proposal. Proceedings of the 15th National Conference, Riyadh, Saudi Arabia, 1997.
5. Al-Waer, M. : The Syntactic, Semantic and Phonological Generation of The Passivization in Standard Arabic, A Computational Linguistic Approach. (In Arabic), Proceedings of the Conference on Using Arabic Language in IT, King AbdulAziz Library, Riyadh, Saudi Arabia, 1992
6. Barwise J., Cooper R.: Generalized Quantifiers and Natural Language, Philosophy Language and Artificial Intelligence, Ed. J. Kulas, J. H. Fetzer and T. Ranken, Kluwer Academic Publishers, Dordrecht, Boston, London, 1988
7. Beesley, K. R.: Finite-State Morphological Analysis and Generation of Arabic at Xerox Research: Status and Plans 2001. ACL/EACL01: Conference of the European Chapter, Workshop: Arabic Language Processing: Status and Prospects, 2001
8. Bos, J., Mastenbroek, E., McGlashan, S, Millies, S, Pinkal M. : A Compositional DRS-based Formalism for NLP Applications. Report 59, VerbMobil, Universitaet des Saarlandes, 1994
9. Chalabi, A.: Sakhr Arabic Lexicon. Proceedings of Nemlar International Conference on Arabic Languages Resources and Tools, 2004
10. Ditters. E.: A Formal Grammar for the Description of Sentences Structures in Modern Standard Arabic. ACL/EACL01: Conference of the European Chapter, Workshop: Arabic Language Processing: Status and Prospects, 2001
11. El-Dessouk, A. Nazif, El-Dessouk, O., Ahmad, A.: An Expert System for Understanding Arabic Sentences. Proceedings of the 10th National Computer Conference, Jeddah, Saudi Arabia, 1987
12. Fahmy, A.: Application of Artificial Intelligence and Expert Systems in Arabization. Proceedings of The Meeting on Computer Arabization, King Saud University, 1994
13. Fischer, W. : Grammatik des klassischen Arabisch. Otto Harrassowitz. Wiesbaden 1972

14. Haddad Bassam, Yaseen Mustafa: Towards Understanding Arabic: A Logical Approach for Semantic Representation. *ACL/EACL01: Conference of the European Chapter, Workshop: Arabic Language Processing: Status and Prospect*, 2001
15. Haddad Bassam, Yaseen M.: Towards Semantic Composition of Arabic : a λ -DRT Based Approach. MT Summit IX, Workshop on Machine Translation for Semitic Languages: Issues and Approaches, AMTA, New Orleans, 2003
16. Haddad Bassam, Yaseen Mustafa: Spell Checking & Correcting Arabic Words. Journal Al-Basaer, University of Petra, to appear in 2005
17. Harper, M.: The representation of noun phrases in logical form. PhD thesis, Brown University 1990
18. Kamp, H.: A theory of truth and semantics representation. In: J. Groendijk, T. J. Stokhof, Formal Methods in Study of Languages. Mathematish Centrum, Amsterdam, 1981
19. Kamp, H., Reyle, U.: From Discourse to Logic. Kluwer Academic Publishers, 1993
20. Montague, R.: The Proper Treatment of Quantification in Ordinary English. In: Philosophy, Language and Artificial Intelligence, ed., J. Kulas, J. H. Fetzer and T. Rankin, Kluwer Academic Publishers, Dordrecht, Boston, London 1988
21. Othman E., Shaalan K., Rafea A.: A Chart Parser for Analyzing Modern Standard Arabic Sentences. MT Summit IX, Workshop on Machine Translation for Semitic Languages: Issues and Approaches, AMTA, New Orleans, 2003
22. Ouersighni R.: A major offshoot of the Dinar-MBC project: AraParse, a morphosyntactic analyzer for unvowelled Arabic texts. *ACL/EACL01: Conference of the European Chapter, Workshop: Arabic Language Processing: Status and Prospects*, 2001.
23. Saint-Dizier, P.: Handling Quantifiers Scoping in a Semantic Representation in Natural Language Sentences. In: Natural Language Understanding and Logic Programming. Ed. V. Dahl and P. Saint-Dizier, 1988
24. Yaseen Mustafa, Haddad Bassam, Papegeorgios Harris, Stelios Piperidis, Hattab Mamoun, Theophilopoulos, Krauer Steven: A Term Base Translator Over The Web. Workshop Proceedings, Arabic Language Processing Status and Prospects. ACL, 10th Conference of the European Chapter, 2001

Strict Deterministic Aspects of Minimalist Grammars

John T. Hale¹ and Edward P. Stabler²

¹ Michigan State University,
East Lansing MI 48824-1027, USA

² University of California, Los Angeles,
Los Angeles CA 90095-1543, USA

Abstract. The Minimalist Grammars (MGs) proposed by Stabler(1997) have tree-shaped derivations (Harkema, 2001b; Michaelis, 2001a). As in categorial grammars, each lexical item is an association between a vocabulary element and complex of features, and so the “yields” or “fringes” of the derivation trees are sequences of these lexical items, and the string parts of these lexical items are reordered in the course of the derivation. This paper shows that while the derived string languages can be ambiguous and non-context-free, the set of yields of the derivation trees is always context-free and unambiguous. In fact, the derivation yield languages are strictly deterministic context-free languages, which implies that they are LR(0), and that the generation of derivation trees from a yield language string can be computed in linear time. This result suggests that the work of MG parsing consists essentially of guessing the lexical entries associated with words and empty categories.

1 Introduction

A derivation is a witness to the fact that a string is generated by a grammar. A derivation says *how* a string is generated, or, equally, what that string’s analysis is in terms of the grammar. Derivations are, in this sense, authoritative about a grammar’s view of a well-formed sentence. From the perspective of a grammar, a generated string is just one facet of the full story: its derivation.

So for any purpose where the structure of a string matters, it is desirable to work with the full story, the derivation – or some needed subset of the information it encodes. In computational linguistics, such work might involve drawing pictures of a sentence’s structure, or generating other sentences, for instance, in other languages. These kinds of applications benefit from the efficient coding of derivations.

This paper shows that, for a particular grammar formalism, the Minimalist Grammars (Stabler, 1997; Stabler and Keenan, 2003) there exists an encoding of derivations that is highly restricted: they can be coded by the sequence of lexical entries as it appears along the fringe of any well-formed derivation tree. This is the same unique readability property familiar from

the syntax of logical languages (Enderton, 2001, 40,108), (Shoenfield, 1967, 15), (Ebbinghaus, Flum, and Thomas, 1994, 22). Not only does this lexical sequence determine the derivation, it is also structured in such a way that such determination can be carried out in linear time by a shift-reduce automaton. This kind of operation on lexical sequences is useful for drawing dependency graphs, X-bar trees annotated with traces, and other kinds of diagrams. The compact representation provided by lexical sequences could have other uses as well. If lexical entries contain “semantic” information, such lexical sequences might indeed represent meanings, constituting a “logical form” for sentences in minimalist languages. These logical forms might be combinable with other (perhaps real-world) knowledge in a natural language understanding system. Similarly, such lexical sequences might be useful in a transfer-based machine translation system.

For all these reasons, and also to gain a deeper understanding of the formalism itself, the simplicity of MG yield languages is of interest. The main result in this paper is that for every MG grammar, the language of MG lexical sequences is a strict deterministic language in the sense of Harrison and Havel (1973). This property is defined in section 2. Then in section 3, MGs are presented as an instance of the more general class of “bare grammars” which includes other formalisms. Section 4 presents a context-free grammar (CFG) for the tree-shaped MG derivations, and shows how this CFG can always be extended to be a strict deterministic grammar. Because any strict deterministic grammar is also an LR(0) grammar in the sense of Kunth (1965), this shows that the language of MG lexical sequences is an LR(0) language.

2 Definitions

To indicate the relationship of the central result about MGs to more familiar grammars, it will be helpful to fix some auxiliary definitions. These stipulations “if, and only if, by definition” are abbreviated by \equiv and “equals, by definition” by \doteq .

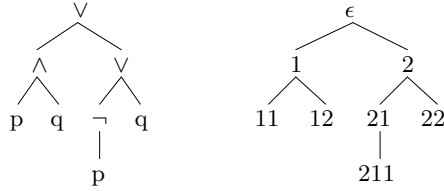
2.1 Trees

At issue most fundamentally are the derivations of linguistic expressions, which are naturally regarded as trees. A **tree** is a function $t : T \rightarrow S$ whose domain is a finite set of sequences T of the positive integers $T \subset (\mathbb{N}_+)^*$ such that, for every $n \in T$:

1. if $n = si$ for some $s \in (\mathbb{N}_+)^*$, $i \in \mathbb{N}$, then $s \in T$,
‘higher nodes are also in the domain’ and
2. if $n = si$ for some $s \in (\mathbb{N}_+)^*$, $i \in \mathbb{N}$, then if $i > 1$ and $j = i - 1$, $tj \in T$
‘lower-numbered sisters are also in the domain’

The elements of a tree domain T are often called **nodes**; ϵ is the **root node**, and the values of t are often called **labels**.

Given any tree $t : T \rightarrow S$ and any $d \in T$, the **subtree** of t at d is the function t/d with domain $\{u \mid du \in \text{dom}(t)\}$, such that $t/d(u) = a$ if and only if $t(du) = a$. In later sections, trees will sometimes be represented with **expression** which are sequences of labels, S and parentheses, as follows. By the previous definitions, for any tree $t : T \rightarrow S$, $t(\epsilon) = a$ is the root label and $k = |\mathbb{N}_+ \cap T|$ is the number of daughters the root has, because intersecting the tree domain T with the positive integers retrieves just the sequences of length 1. When $k = 0$ the expression of t is a , and otherwise it is $a(t_1 \dots t_k)$, where t_1, \dots, t_k are the expressions of the subtrees $t/1, \dots, t/k$. For example, the tree with the conventional depiction here has the tree domain indicated on the right:



The expression of this tree is $\vee(\wedge(pq) \vee (\neg(p)q))$. The **yield** of this tree, on the left, is $pqpq$, where the yield of a tree $t : T \rightarrow S$ is the sequence of elements of S defined as follows,

$$\text{yield}(t) = \begin{cases} a & \text{if } \text{dom}(t) = \{\epsilon\}, t(\epsilon) = a \\ \text{yield}(a/1) \dots \text{yield}(a/k) & \text{otherwise, where } k = |\mathbb{N}_+ \cap T|. \end{cases}$$

2.2 Context-Free Grammars

Following Keenan and Stabler (2003), take a context-free grammar (CFG) to be a triple $G = \langle \Sigma, N, (\rightarrow) \rangle$ where Σ, N are nonempty and disjoint, the start symbol S is an element of the nonterminal set N , and $(\rightarrow) \subseteq N \times (\Sigma \cup N)^*$ is finite. The immediate rewriting relation (\rightarrow) is to be viewed as an infix operator so that $x \rightarrow y$ is a shorthand for $\langle x, y \rangle \in (\rightarrow)$. Similarly, the rewriting relation $\Rightarrow_G \doteq \{ \langle x, y \rangle \in (\Sigma \cup N)^* \times (\Sigma \cup N)^* \mid \exists u, v, w \in (\Sigma \cup N)^*, A \in N, \text{ such that } x = uAv, A \rightarrow w, x = uwv \}$ on sentential forms is abbreviated as $x \Rightarrow y$.

Let \Rightarrow^* be the reflexive transitive closure of \Rightarrow . For any $A \in N$, the sequences of category A , $L_A(G) = \{x \in \Sigma^* \mid A \Rightarrow^* x\}$. For any $A \in N$, $L_A(G)$ is a context-free language.

To set the stage for the following more general framework of subsection 3.1, define the **derivation trees** $\Gamma(G)$ of a CFG G as follows, labeling internal nodes with elements of \rightarrow rather than with just their left sides:

$$\begin{aligned} \Gamma(G)^0 &\doteq \{a \in \Sigma^* \mid A \rightarrow a\}, \\ \Gamma(G)^{k+1} &\doteq \Gamma(G)^k \cup \{A \rightarrow a(a) \mid A \rightarrow a \text{ and } a \in (\Gamma(G)^k)^*\}, \\ \Gamma(G) &\doteq \bigcup_{k \in \mathbb{N}} \Gamma(G)^k. \end{aligned}$$

Furthermore, for any $A \in N$,

$$\Gamma_A(G) \doteq \{t \in \Gamma(G) \mid t(\epsilon) = A \rightarrow a, \text{ for some } a \in (\Sigma \cup N)^*\}.$$

Clearly, $yield(\Gamma_A(G)) = L_A(G)$. G **has ambiguous yields** \doteq there are two distinct trees in $\Gamma(G)$ with the same yield. Typically, one just says that such a grammar is “ambiguous.”

2.3 Strict Determinism

Strict determinism is a property that context-free grammars can have; it is a technical notion that figures prominently in the theory of deterministic context-free grammars, which is reviewed in (Harrison, 1978, §11). Most simply, a CFG is strict deterministic if its symbols can be divided up into blocks in a certain way. The formal definition uses Harrison’s notation $^{(n)}\alpha$ for the prefix of α of length $\min\{n, |\alpha|\}$.

For any CFG G , a partition π of $N \cup \Sigma$ is **strict** if and only if

- i. $\Sigma \in \pi$
- ii. for any $A, A' \in N$, $\alpha, \beta, \beta' \in (N \cup \Sigma)^*$, if $A \rightarrow \alpha\beta$ and $A' \rightarrow \alpha\beta'$ and $A \equiv A' \pmod{\pi}$, then either
 - a. both $\beta, \beta' \neq \epsilon$ and $^{(1)}\beta \equiv^{(1)} \beta' \pmod{\pi}$, or
 - b. $\beta = \beta' = \epsilon$ and $A = A'$.

If the partition π is clear from context, $A \equiv B \pmod{\pi}$ is sometimes simplified to $A \equiv B$, and similarly the block of π where a symbol A resides is written $[A]$ rather than $[A]_\pi$ if no confusion would result.

With this definition, it is possible to say of a CFG G that it is **strict deterministic** if there exists a strict partition π of $N \cup \Sigma$. A language is strict deterministic if some strict deterministic grammar generates it.

A CFG is said to be reduced with respect to a given start symbol S , abbreviated **reduced_S** just in case $\rightarrow = \emptyset$ or for every $A \in N$, $S \Rightarrow^* \alpha A \beta \Rightarrow^* w$ for some $\alpha, \beta \in (N \cup \Sigma)^*$ and $w \in \Sigma^*$.

It is also known that

Theorem (Harrison, 11.4.1). *Any strict deterministic grammar is equivalent to a reduced strict deterministic grammar.*

Theorem (Harrison, 13.2.3). *Any reduced strict deterministic grammar is also an $LR(0)$ grammar.*

These theorems link strict determinism to its consequences for efficient parsing. Harrison (page 347) explains that

The motivation behind this definition is that we wish to make certain restrictions on the simultaneous occurrences of substrings in different productions. Intuitively, if $A \rightarrow \alpha\beta$ is a production in our grammar, then “partial information” about A , together with complete information about a prefix α of $\alpha\beta$, yields similar partial information about the next symbol of β when $\beta \neq \epsilon$, or the complementary information about A when $\beta = \epsilon$. In the formal definition, the intuitive notion of “partial information” is precisely represented by means of the partition π .

Loosely speaking, in an automaton processing a strict deterministic grammar, top-down knowledge of $A \equiv A'$ buys knowledge of $^{(1)}\beta \equiv^{(1)} \beta'$.

The family of all strict deterministic languages is a subfamily of the languages accepted by deterministic pushdown automata that accept by *empty* stack in a final state (Harrison, theorem 11.5.4). In strict deterministic grammars, right-hand sides of rewriting rules cannot be prefixes of one another.

But if this restriction is relaxed (the qualifier “strict” is dropped), by distinguishing a special set of prefix-deriving parents, the defined language family grows to be identical with the one defined by deterministic pushdown automata that accept in a final state with *any* stack configuration (Harrison, problem 4 page 392).

2.4 Examples

This section exercises the definitions just given with some CFGs for a simple language, such as might be used in sentential logic.

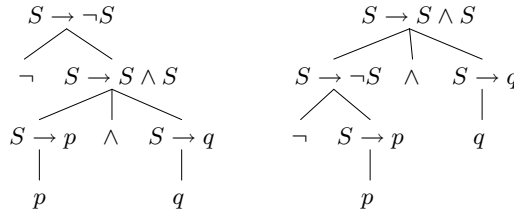
Example 1 (Ambiguity). Consider the context-free grammar $G1 = \langle \Sigma, N, \rightarrow \rangle$, where
 $\Sigma = \{p, q, r, \neg, \vee, \wedge\}$,
 $N = \{S\}$, and
 \rightarrow has the following 6 pairs in it:

$$\begin{array}{lll} S \rightarrow p & S \rightarrow q & S \rightarrow r \\ S \rightarrow \neg S & S \rightarrow S \vee S & S \rightarrow S \wedge S \end{array}$$

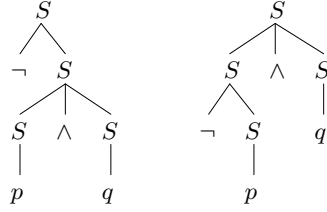
This grammar is (yield-)ambiguous since there are the following derivation trees for $\neg p \wedge q$:

$$\begin{array}{l} S \rightarrow \neg S(\neg, S \rightarrow S \wedge S(S \rightarrow p(p), S \rightarrow q(q))) \\ S \rightarrow S \wedge S(S \rightarrow \neg S(\neg, S \rightarrow p(p)), S \rightarrow q(q)) \end{array}$$

One can draw graphical presentations that are more readable, like this:



These CFG derivation trees are slightly redundant, since the right sides of the rules at each internal node can be read off the daughters. Eliminating the right side of each production gives the standard depictions:



There is only one partition of $\Sigma \cup N$ that contains Σ , namely $\pi = \{\Sigma, \{N\}\}$. Evidently, this partition is not strict, because it fails condition (ii) on page 165. When $\alpha = \epsilon$, then $G1$ has, for example $\beta = p$ and $\beta' = S$. These are both non-empty, and their first symbols are not in the same block of the partition π .

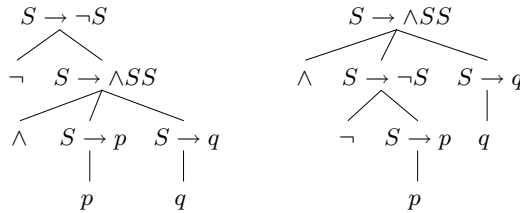
Example 2 (Unambiguous Polish Notation). Consider the context-free grammar $G2 = \langle \Sigma, N, \rightarrow \rangle$, where
 $\Sigma = \{p, q, r, \neg, \vee, \wedge\}$,
 $N = \{S\}$, and
 \rightarrow has the following 6 pairs in it:

$$\begin{array}{lll} S \rightarrow p & S \rightarrow q & S \rightarrow r \\ S \rightarrow \neg S & S \rightarrow \vee S S & S \rightarrow \wedge S S \end{array}$$

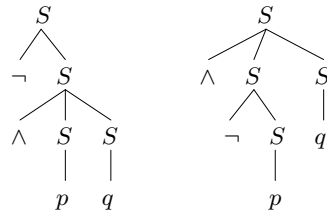
In $G2$, the operators \vee and \wedge have been pushed to the leftmost position in the right-hand side of each rule. On this grammar, there is just one derivation tree for $\wedge \neg pq$, and just one for $\neg \wedge pq$:

$$\begin{aligned} S &\rightarrow \neg S(\neg, S \rightarrow \wedge SS(\wedge, S \rightarrow p(p), S \rightarrow q(q))) \\ S &\rightarrow \wedge SS(\wedge, S \rightarrow \neg S(\neg, S \rightarrow p(p)), S \rightarrow q(q)) \end{aligned}$$

The same, more readable diagrams can be drawn:



The labels on these derivation trees can be shortened to just the left-hand side of the applied rule, in the same way as with $G1$:



There is only one partition of $\Sigma \cup N$ that contains Σ , namely $\pi = \{\Sigma, \{N\}\}$. This partition is strict. When $\alpha = \epsilon$, then no matter which rules we choose, ⁽¹⁾ $\beta \in \Sigma$ and so the conditions are satisfied. And there are no two different β, β' and two A, A' such that for some non-empty α , $A \rightarrow \alpha\beta$ and $A' \rightarrow \alpha\beta'$.

To set the stage for later developments, it is worth briefly considering two further variants of $G2$.

Example 3 (Unambiguous, But Not Strict Deterministic). Consider the context-free grammar $G2a = \langle \Sigma, N, \rightarrow \rangle$, where

$\Sigma = \{p, q, r, \neg, \vee, \wedge\}$,

$N = \{S, B\}$, and

\rightarrow has the following 7 pairs in it:

$$\begin{array}{lll} S \rightarrow p & S \rightarrow q & S \rightarrow r \\ S \rightarrow \neg S & S \rightarrow BSS & \\ B \rightarrow \wedge & B \rightarrow \vee & \end{array}$$

Clearly, $G2a$ generates the same strings of category S as $G2$, but $G2a$ is not strictly deterministic, since the set $\Sigma \cup N$ has no strict partition.

Example 4 (Unambiguous, and Strict Deterministic again). Consider the context-free grammar $G2b = \langle \Sigma, N, \rightarrow \rangle$, where

$\Sigma = \{p, q, r, \neg, \vee, \wedge\}$,

$N = \{S, B, U, A\}$, and

\rightarrow has the following 9 pairs in it:

$$\begin{array}{lll} S \rightarrow A & S \rightarrow US & S \rightarrow BSS \\ A \rightarrow p & A \rightarrow q & A \rightarrow r \\ U \rightarrow \neg & B \rightarrow \wedge & B \rightarrow \vee \end{array}$$

Clearly, $G2b$ generates the same strings of category S as $G2$ and $G2a$, but $G2b$ is strictly deterministic, since the set $\Sigma \cup N$ has the strict partition $\{\Sigma, \{A, U, B\}, \{S\}\}$.

With these examples clarifying what ambiguity and strict determinism amount to in CFGs, section 3 turns to the Minimalist Grammars.

3 Grammars

3.1 Bare Grammars

Minimalist Grammars are one of a variety of formalisms that construes a grammar \mathcal{G} as a set of basic expressions Lex and a set \mathcal{F} of partial functions from tuples of expressions to expressions (Keenan and Stabler, 2003). The language $L(\mathcal{G})$ is then the closure of Lex with respect to the functions in \mathcal{F} . Internal nodes in the **derivations** of \mathcal{G} , $\Gamma(\mathcal{G})$, are labeled with elements of \mathcal{F} just in case $f \in \mathcal{F}$ is applicable to the children:

$$\begin{aligned}
\Gamma(\mathcal{G})^0 &\doteq Lex, \\
\Gamma(\mathcal{G})^{k+1} &\doteq \Gamma(\mathcal{G})^k \cup \{f(a) \mid f \in \mathcal{F} \text{ and } a \in ((\Gamma(\mathcal{G})^k)^* \cap \text{dom}(f))\}, \\
\Gamma(\mathcal{G}) &\doteq \bigcup_{k \in \mathbb{N}} \Gamma(\mathcal{G})^k.
\end{aligned}$$

Clearly, if f labels the root of some tree in $\Gamma(\mathcal{G})$, then f is a function expression whose evaluation returns an element $e \in L(\mathcal{G})$. So in these derivation trees, each node has a **value** which is the denotation of the function expression which is its label, always an element of $L(\mathcal{G})$.

Grammar \mathcal{G} **has ambiguous expressions** \doteq some expression $e \in L(\mathcal{G})$ is the value of the roots of two distinct trees in $\Gamma(\mathcal{G})$. Grammar \mathcal{G} **has ambiguous yields** \doteq there are two distinct trees in $\Gamma(\mathcal{G})$ with the same yield. Notice that the yields of derivations from bare grammars are sequences from Lex^* .

3.2 Minimalist Grammars

Minimalist Grammars instantiate this general picture, with elements of Lex comprising the sequences of a quite limited inventory of “features”, along with two structure-building functions that are constrained in their application. A **Minimalist Grammar** $\mathcal{G} \doteq \langle \Sigma, B, Lex, \mathcal{F} \rangle$, where

1. Σ is a non-empty set (the pronounced elements)
 2. B is a non-empty set of **basic features**, which serve to specify the **features** $F \doteq B \cup S \cup M \cup N$ where $=, -, +$ are 1-1 functions with domain B such that **selectors** $S \doteq \{=f \mid f \in B\}$, **licensees** $M \doteq \{-f \mid f \in B\}$, **licensors** $N \doteq \{+f \mid f \in B\}$, and B, S, M, N are pairwise disjoint.
 3. The **lexicon** $Lex \subset \Sigma^* :: F^*$ is a finite, nonempty set of lexical chains, where the **chains** $C \doteq \Sigma^* T F^*$, and the types $T \doteq \{::, :\}$ distinguish **lexical** chains from **derived chains**, respectively.
 4. The generating functions $\mathcal{F} = \{merge, move\}$ are partial functions from tuples of expressions to expressions, where **expressions** $E \doteq C^+$. It will be convenient to define these functions in a deductive format, with the arguments as premises and the values as the conclusion.
- (a) $merge : (E \times E) \rightarrow E$ is the union of the following 3 functions, for $s, t \in \Sigma^*$, for $\cdot \in \{::, :\}$, for $f \in B$, $\gamma \in F^*$, $\delta \in F^+$, and for chains $\alpha_1, \dots, \alpha_k, \iota_1, \dots, \iota_l$ ($0 \leq k, l$)

$$\frac{s :: =f\gamma \quad t \cdot f, \alpha_1, \dots, \alpha_k}{st : \gamma, \alpha_1, \dots, \alpha_k} \text{r1}$$

r1 applies when s is lexical; it selects its argument on the right

$$\frac{s :: =f\gamma, \alpha_1, \dots, \alpha_k \quad t \cdot f, \iota_1, \dots, \iota_l}{ts : \gamma, \alpha_1, \dots, \alpha_k, \iota_1, \dots, \iota_l} \text{r2}$$

r2 applies when s is phrasal and t has no more features; it selects its argument on the left

$$\frac{s \cdot f\gamma, \alpha_1, \dots, \alpha_k \quad t \cdot f\delta, \iota_1, \dots, \iota_l}{s : \gamma, t : \delta, \alpha_1, \dots, \alpha_k, \iota_1, \dots, \iota_l} \text{r3}$$

r3 applies when s is phrasal and t has more features; it selects its argument on the left

Here st is the concatenation of strings s, t . Note that since the domains of r1, r2, and r3 are disjoint, their union is a function.

- (b) *move* : $E \rightarrow E$ is the union of the following 2 functions, for $s, t \in \Sigma^*$, $f \in B$, $\gamma \in F^*$, $\delta \in F^+$, and for chains $\alpha_1, \dots, \alpha_k, \iota_1, \dots, \iota_l$ ($0 \leq k, l$) satisfying the following condition: (SMC) none of $\alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k$ has $-f$ as its first feature.

$$\frac{s : +f\gamma, \alpha_1, \dots, \alpha_{i-1}, t : -f, \alpha_{i+1}, \dots, \alpha_k}{ts : \gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k} \text{m1} \quad \text{m1 when no features follow } -f$$

$$\frac{s : +f\gamma, \alpha_1, \dots, \alpha_{i-1}, t : -f\delta, \alpha_{i+1}, \dots, \alpha_k}{s : \gamma, \alpha_1, \dots, \alpha_{i-1}, t : \delta, \alpha_{i+1}, \dots, \alpha_k} \text{m2} \quad \text{m2 when some features follow } -f$$

Notice that the domains of m1 and m2 are disjoint, so their union is a function. The (SMC) restriction on the domain of *move* is a simple version of the “shortest move condition” (Chomsky, 1995).

Often, one is interested in a subset of $L(G)$, for instance just the derivations of complementizer phrases. These derivations are all expressions of a particular syntactic category. More generally, for any $f \in B$, the **expressions of category** f , $L_f(\mathcal{G}) \doteq \{s \cdot f \in L(\mathcal{G}) \mid \text{for some } \cdot \in \{:, ::\}\}$; the **strings of category** f , $S_f(\mathcal{G}) \doteq \{s \mid s \cdot f \in L_f(\mathcal{G}) \text{ for some } \cdot \in \{:, ::\}\}$; the **derivations of** f , $\Gamma_f(\mathcal{G}) \doteq \{d \in \Gamma(\mathcal{G}) \mid d(\epsilon) \in L_f(G)\}$. A derivation d is **complete** \doteq it is in some $\Gamma_f(\mathcal{G})$ for some $f \in B$. A set $L \subseteq \Sigma^*$ is a **minimalist language** \doteq for some MG and some $f \in B$, $S_f(\mathcal{G}) = L$.

The MG-definable languages are exactly the same as the languages definable by set-local multicomponent tree adjoining grammars, by multiple context-free grammars, and other well known grammars (Michaelis, 1998; Michaelis, 2001b; Harkema, 2001a).

Example 5 (an MG for an ambiguous language). Consider now an MG similar to the context-free grammar $G1$, $\mathcal{G3} = \langle \Sigma, N, Lex, \mathcal{F} \rangle$, where

$$\Sigma = \{p, q, r, \neg, \vee, \wedge\},$$

$$N = \{S\}, \text{ and}$$

Lex has the following 6 lexical items built from Σ and N

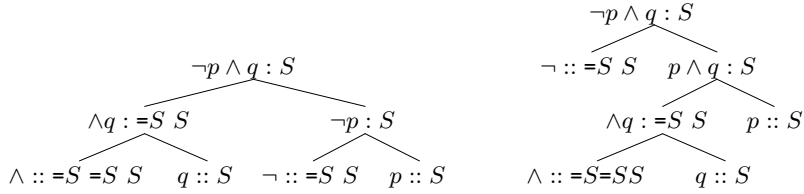
$$\begin{array}{lll} p :: S & q :: S & r :: S \\ \neg :: =S S & \vee :: =S =S S & \wedge :: =S =S S \end{array}$$

Grammar $\mathcal{G3}$ has ambiguous expressions, since we have the following two different derivations of $\neg p \wedge q$:

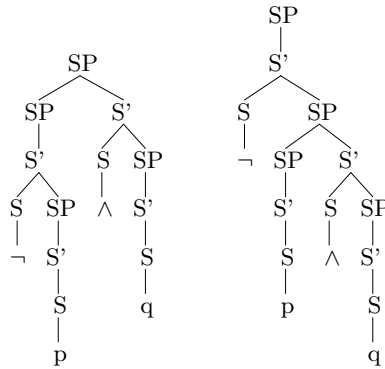
$$\text{merge}(\text{merge}(\wedge :: =S =S S, q :: S), \text{merge}(\neg :: =S S, p :: S))$$

$$\text{merge}(\neg :: =S S, \text{merge}(\text{merge}(\wedge :: =S =S S, q :: S), p :: S))$$

A graphical presentation can be provided which, instead of marking all the internal nodes with the uninformative symbol *merge*, labels them with the values of *merge*:



These derivations correspond to the X-bar structures given below:



While these examples show that $\mathcal{G}3$ has ambiguous expressions, they do not show that $\mathcal{G}3$ has ambiguous yields. Notice that the yields of the two simple derivation trees shown above (not the X-bar structures, but the derivation trees) are not the same. The two yields are, respectively,

$$\begin{array}{l}
 \wedge :: =S=SS \quad q :: S \quad \neg :: =SS \quad p :: S \\
 \neg :: =SS \quad \wedge :: =S=SS \quad q :: S \quad p :: S
 \end{array}$$

In fact, $\mathcal{G}3$ derivations have unambiguous yields. That is, each sequence of lexical items is the yield of at most one derivation. However, while these sequences of lexical items determine their derivations, the corresponding multisets do not, as can be seen in this example from the fact that exchanging the positions of lexical items p and q , gives two new derivations which, respectively have the same multisets of lexical items as the two derivations shown above. These latter two derivations derive the string $\neg q \wedge p$.

The suggestion is that extra information contained in the leaves – thrown away when the structure building function *merge* is evaluated – is enough to fully determine the derivation. This suggestion is amplified into a general procedure in section 4.

4 Strict Deterministic Grammars for MG Derivations

This section first gives the “natural” CFG for MG derivation tree fringes, adapting some basic ideas from Michaelis (1998). The CFGs obtained this way (subsection 4.1) are not, in general, strict deterministic, but subsection 4.2 shows how they can always be extended so as to become so. The argument then is that, because a strict deterministic grammar for the derivation tree fringe language exists, the language is strict deterministic, hence $LR(0)$, hence uniquely readable.

4.1 The Natural Translation

Perhaps the most natural view of MG derivations as generated by CFGs simply ignores the string-manipulation parts of the structure-building functions. Abbreviate by numerical subscripting i the set of i^{th} projections of each element in a set of n -tuples, $i \leq n$. Then for any MG $\mathcal{G} = \langle \Sigma, B, Lex, \{merge, move\} \rangle$, define

$$\begin{aligned} R(Lex) &\doteq \{Fs \rightarrow S :: Fs \mid S :: Fs \in Lex\}, \\ R(\mathcal{G}) &\doteq \text{closure}(R(Lex), \{rmerge, rmove\}) \\ h(\mathcal{G}) &\doteq \langle \Sigma', Cat, \rightarrow \rangle \text{ where } \Sigma' = Lex, Cat = R(\mathcal{G})_1, \text{ and } \rightarrow = R(\mathcal{G}). \end{aligned}$$

The functions $rmerge, rmove$ are the obvious modifications of $merge, move$. To obtain the CFG, simply eliminate the string components everywhere except at the leaves. So instead of chains C with string components, the **cchains** $CC \doteq F^*$ are just feature sequences, and the **possible rules** $R \doteq CC^+ \times (Lex \cup CC^+)$. To generate the needed context-free rules, $rmerge : (R \times R) \rightarrow R$ is the union of the following 3 functions, for $f \in B, \gamma \in F^*, \delta \in F^+, \text{ for cchains } \alpha_1, \dots, \alpha_k, \iota_1, \dots, \iota_l$ ($0 \leq k, l$), for non-lexical right sides $N \in CC^+$, and for arbitrary right sides $M, L \in (Lex \cup CC^+)$

$$\begin{aligned} &\frac{=f\gamma \rightarrow s :: =f\gamma \quad f, \alpha_1, \dots, \alpha_k \rightarrow M}{\gamma, \alpha_1, \dots, \alpha_k \rightarrow =f\gamma \quad f, \alpha_1, \dots, \alpha_k} \text{rr1} \\ &\frac{=f\gamma, \alpha_1, \dots, \alpha_k \rightarrow N \quad f, \iota_1, \dots, \iota_l \rightarrow M}{\gamma, \alpha_1, \dots, \alpha_k, \iota_1, \dots, \iota_l \rightarrow =f\gamma, \alpha_1, \dots, \alpha_k \quad f, \iota_1, \dots, \iota_l} \text{rr2} \\ &\frac{=f\gamma, \alpha_1, \dots, \alpha_k \rightarrow M \quad f\delta, \iota_1, \dots, \iota_l \rightarrow L}{\gamma, t : \delta, \alpha_1, \dots, \alpha_k, \iota_1, \dots, \iota_l \rightarrow =f\gamma, \alpha_1, \dots, \alpha_k \quad f\delta, \iota_1, \dots, \iota_l} \text{rr3} \end{aligned}$$

Note that since the domains of rr1, rr2, and rr3 are disjoint, their union is a function.

Similarly, $remove : R \rightarrow R$ is the union of the following 2 functions:

$$\frac{+f\gamma, \alpha_1, \dots, \alpha_{i-1}, -f, \alpha_{i+1}, \dots, \alpha_k \rightarrow N}{\gamma, \alpha_1, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_k \rightarrow +f\gamma, \alpha_1, \dots, \alpha_{i-1}, -f, \alpha_{i+1}, \dots, \alpha_k} \text{rm1}$$

$$\frac{+f\gamma, \alpha_1, \dots, \alpha_{i-1}, -f\delta, \alpha_{i+1}, \dots, \alpha_k \rightarrow N}{\gamma, \alpha_1, \dots, \alpha_{i-1}, \delta, \alpha_{i+1}, \dots, \alpha_k \rightarrow +f\gamma, \alpha_1, \dots, \alpha_{i-1}, -f\delta, \alpha_{i+1}, \dots, \alpha_k} \text{rm2}$$

Notice several facts about the translation h .

Theorem (Michaelis, Harkema). $R(\mathcal{G})$ is finite.

Because of $R(\mathcal{G})$'s finitude, h is well-defined.

Theorem (h -Correctness). For every MG \mathcal{G} , $s \in \text{yield}(\Gamma_f(\mathcal{G}))$ if and only if $s \in L_f(h(\mathcal{G}))$.

Proof idea: this is established with an easy induction on derivation lengths, since the context-free rules rr and rm correspond to every possible application of merge and move.

Theorem 1 (Non-left-recursive). For every MG \mathcal{G} , $h(\mathcal{G})$ is not left recursive.

Proof idea: This is easy to see, since the label of any left daughter of any node in any derivation is always strictly larger than its parent. In the case of merge, it is one feature larger; in the case of move, it is two features larger.

Examples 6 and 7 show how the range of this translation is not restricted to strict deterministic CFGs. However, subsection 4.2 illustrates another translation g that is restricted in this way.

Example 6 ($h(\mathcal{G}3)$ is not strictly deterministic). Consider grammar $\mathcal{G}3$ from page 170. The CFG $h(\mathcal{G}3) = \langle \Sigma, N, \rightarrow \rangle$ where $\Sigma = \text{Lex}$, $N = \{S, =SS, =S=SS\}$, and \rightarrow is the following 8 pairs:

$$\begin{array}{llll} S \rightarrow =SS & S & S \rightarrow p :: S & S \rightarrow q :: S & S \rightarrow r :: S \\ =SS \rightarrow =S=SS & S & =SS \rightarrow \neg :: =SS & & \\ =S=SS \rightarrow \wedge :: =S=SS & =S=SS \rightarrow \vee :: =S=SS & & & \end{array}$$

Notice that 6 of the 8 pairs are lexical. Also, it is clear that $\Sigma \cup N$ has no strict partition. Notice, in particular, that the category $=SS$ can rewrite as a lexical item or as a pair of nonterminals.

Example 7 ($h(\mathcal{G}3)a \equiv h(\mathcal{G}3)$ and $h(\mathcal{G}3)a$ is strictly deterministic). Let $h(\mathcal{G}3)a \doteq \langle \Sigma, N, \rightarrow \rangle$ where $\Sigma = \text{Lex}$, $N = \{1=SS, 2=SS, 1=S=SS, S=SS, =S=SS\}$, and \rightarrow is the following 11 pairs:

$$\begin{array}{lll}
S \rightarrow =SS & S & S \rightarrow 2=SS \\
2=SS \rightarrow 1=SS & 1=SS \rightarrow p :: S & 1=SS \rightarrow q :: S \quad 1=SS \rightarrow r :: S \\
=SS \rightarrow =S=SS & S & =SS \rightarrow 1=S=SS \\
1=S=SS \rightarrow \neg :: =SS & & \\
=S=SS \rightarrow \wedge :: =S=SS & =S=SS \rightarrow \vee :: =S=SS &
\end{array}$$

Clearly $L_S(h(\mathcal{G}3)) = L_S(h(\mathcal{G}3)a)$, but now $\Sigma \cup N$ has a strict partition:

$$\{\{S\}, \{=SS, 2=SS\}, \{=S=SS, 1=S=SS, 1=SS\}, \Sigma\}.$$

4.2 Extending the Natural Translation

The step from $h(\mathcal{G}3)$ to $h(\mathcal{G}3)a$ can be generalized to show that, for any MG \mathcal{G} the grammar $h(\mathcal{G})$ generates an LR(0) language. This is shown using another language-preserving map g on the grammars $h(\mathcal{G})$ whose range includes only strict deterministic CFGs.

Given $h(\mathcal{G}) = \langle \Sigma, N, \rightarrow \rangle$, for any $A \in N$, sequence $(A_1 A_2 \dots A_n) \in N^+$ is a **left branch** of A if and only if the following three conditions hold:

1. $A = A_1$
2. for all $1 \leq i < n$, either $A_i \rightarrow A_{i+1}$ or $A_i \rightarrow A_{i+1}B$ for some $B \in N$, and
3. $A_n \rightarrow a$ for some $a \in \Sigma$

When $A \rightarrow a$ for $a \in \Sigma$, the empty sequence ϵ is a left branch of A .

As observed earlier (theorem 1), for any MG \mathcal{G} $h(\mathcal{G})$ is never left recursive. Since there are no infinite left-recursive branches, one can define a ranking function $rank : (N \cup Lex) \rightarrow \mathbb{N}$ so that for $a \in \Sigma$, $rank(a) = 0$ and for $A \in N$, $rank(A)$ is the length of the longest left branch of A .

Example 8. In $h(\mathcal{G}3)$, notice that $rank(S) = 3$, $rank(=SS) = 2$, and $rank(=S=SS) = 1$.

Now g can be defined directly on MGs using h . Observe that all rules $p \in h(\mathcal{G})$ are of the form $p = A \rightarrow a$ or $p = A \rightarrow BC$ or $p = A \rightarrow B$. Define a function pad by cases that maps each such rule to a set of pairs as follows:

$p = A \rightarrow a$:

If $rank(A) = 1$, then $pad(p) = \{p\}$. Else we add padded categories from $rank(A)$ down to 1 as follows:

$$pad(p) = \{A \rightarrow (rank(A)-1)_A, (rank(A)-1)_A \rightarrow (rank(A)-2)_A, \dots, 1_A \rightarrow a\}.$$

$p = A \rightarrow BC$:

If $rank(B) = rank(A) - 1$, then $pad(p) = \{p\}$. Else we add padded categories from $rank(A)$ down to $rank(B)$ as follows:

$$pad(p) = \{ A \rightarrow (rank(A) - 1)_B C, (rank(A) - 1)_B \rightarrow (rank(A) - 2)_B, \dots, rank(B)_B \rightarrow B \}.$$

$p = A \rightarrow B$: $pad(p)$ is defined similarly.

The new rule set $R'(\mathcal{G})$ is defined to be $\bigcup_{r \in (\rightarrow)} pad(r)$, and for any MG \mathcal{G} , the context-free grammar $g(\mathcal{G}) \doteq \langle \Sigma, R'(\mathcal{G})_1, R'(\mathcal{G}) \rangle$.

Theorem (*g*-Correctness). *For every MG \mathcal{G} , $h(\mathcal{G}) = \langle \Sigma, N, \rightarrow \rangle$ and $g(\mathcal{G}) = \langle \Sigma, N', \rightarrow' \rangle$, and any $A \in N$, $L_A(h(\mathcal{G})) = L_A(g(\mathcal{G}))$.*

Proof: From the definition of g , we have immediately that $N \subseteq N'$ and a simple induction shows, using the definition of pad , that the same sets of terminal strings are derivable. \square

The ranking of nonterminals by their longest left branch induces a partition. For each $x \in (N \cup \Sigma)$, $[x] \doteq \{y \in (N \cup \Sigma) \mid rank(y) = rank(x)\}$. Since there is a unique maximum left branch length for every nonterminal, the $[x]$ are disjoint and since every nonterminal has a maximum left branch length, the $[x]$ are exhaustive.

Theorem 2 (Strict Determinism of MG derivation languages). *For any MG \mathcal{G} , $g(\mathcal{G}) = \langle \Sigma, N, \rightarrow \rangle$, the set*

$$\pi = \{[x] \mid x \in (N \cup \Sigma)\}$$

is a strict partition of $N \cup \Sigma$.

Proof. Referring again to the definition of strict partitions (2.3) on page 165, each condition is satisfied by construction:

1. $\Sigma \in \pi$ since by the definition of $rank$, $\Sigma = [a]$ for all $a \in \Sigma$.
2. For $\alpha = \epsilon$, it follows from the definition of pad that rules with equivalent left sides have equivalent first symbols on their right sides.
And there are no two different β, β' and two A, A' such that for some non-empty α , $A \rightarrow \alpha\beta$ and $A' \rightarrow \alpha\beta'$. \square

5 Conclusion

In showing that the derivation tree fringes of MGs are strict deterministic, it has been important to keep in mind the difference between having (un)ambiguous *expressions* – i.e. that the string languages are unambiguous, which is certainly false for MGs – and having (un)ambiguous *yields* – i.e. that the language of lexical entry sequences is unambiguous. This latter point is the one demonstrated in this paper. Because strict determinism implies LR(0), shift-reduce automata can quickly assemble a sequence of lexical entries into a tree, dependency graph or other representation, after chart-parsing or some other method has disambiguated a grammatical string into the correct sequence of lexical entries.

References

- Chomsky, Noam. 1995. *The Minimalist Program*. MIT Press, Cambridge, Massachusetts.
- Ebbinghaus, Heinz-Dieter, Jörg Flum, and Wolfgang Thomas. 1994. *Mathematical Logic*. Springer-Verlag.
- Enderton, Herbert B. 2001. *A Mathematical Introduction to Logic*. Harcourt.
- Harkema, Henk. 2001a. A characterization of minimalist languages. In *Proceedings, Logical Aspects of Computational Linguistics, LACL'01*, Port-aux-Rocs, Le Croisic, France.
- Harkema, Henk. 2001b. *Parsing Minimalist Grammars*. Ph.D. thesis, UCLA.
- Harrison, Michael A. 1978. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Massachusetts.
- Harrison, Michael A. and Ivan M. Havel. 1973. Strict deterministic grammars. *Journal of Computer and System Sciences*, 7:237–277.
- Keenan, Edward L. and Edward P. Stabler. 2003. *Bare Grammar: Lectures on Linguistic Invariants*. Stanford Monographs in Linguistics. CSLI Publications.
- Knuth, Donald. 1965. On the translation of languages from left to right. *Information and Control*, 8(6):607–639.
- Michaelis, Jens. 1998. Derivational minimalism is mildly context-sensitive. In *Proceedings, Logical Aspects of Computational Linguistics, LACL'98*, Grenoble.
- Michaelis, Jens. 2001a. *On Formal Properties of Minimalist Grammars*. Ph.D. thesis, Potsdam University.
- Michaelis, Jens. 2001b. Transforming linear context free rewriting systems into minimalist grammars. In *Proceedings, Logical Aspects of Computational Linguistics, LACL'01*, Le Croisic, France.
- Shoenfield, Joseph R. 1967. *Mathematical Logic*. Addison-Wesley.
- Stabler, Edward and Edward Keenan. 2003. Structural similarity. *Theoretical Computer Science*, 293:345–363.
- Stabler, Edward P. 1997. Derivational minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*, pages 68–95. Springer.

A Polynomial Time Extension of Parallel Multiple Context-Free Grammar

Peter Ljunglöf

Department of Computing Science,
Göteborg University and Chalmers University of Technology,
SE-412 96 Göteborg, Sweden
`peb@cs.chalmers.se`

Abstract. It is already known that *parallel multiple context-free grammar* (PMCFG) [1] is an instance of the equivalent formalisms *simple literal movement grammar* (sLMG) [2, 3] and *range concatenation grammar* (RCG) [4, 5]. In this paper we show that by adding the single operation of intersection, borrowed from *conjunctive grammar* [6], PMCFG becomes equivalent to sLMG and RCG. As a corollary we get that PMCFG with intersection describe exactly the class of languages recognizable in polynomial time.

The layout of this paper is as follows. The first section contains definitions of the basic grammar formalisms we are interested in. The second section introduces the intersection operation for PMCFG. The third section contains the main result of the paper – that PMCFG extended with the intersection operation is equivalent to simple LMG and RCG. The fourth and last section is a small discussion of the results.

1 GCFG, PMCFG, sLMG and RCG

1.1 Generalized Context-Free Grammar

Generalized context-free grammar (GCFG) was introduced by Pollard in the 80's as a way of formally describing *head grammar* [7]. There are several definitions of GCFG in the literature; Seki et al [1] use a definition similar to Pollard's original, while others [8, 9, 10] more cleanly separates between abstract and concrete syntax. However, the latter definitions use the term GCFG for only the abstract part of the grammar, and the term *context-free rewriting system* for the abstract grammar together with the concrete interpretation function. While Pollard imposed no restriction on the concrete linearization type, other definitions restrict them to be tuples of strings. Here we use the definition from [11], which is close to the original definition.

Definition 1 (GCFG, abstract part). *The abstract grammar of a GCFG is a tuple $(\mathcal{C}, S, \mathcal{F}, \mathcal{R})$, where \mathcal{C} and \mathcal{F} are finite sets of categories and function symbols respectively, $S \in \mathcal{C}$ is the starting category, and $\mathcal{R} \subseteq \mathcal{C} \times \mathcal{F} \times \mathcal{C}^*$ is a*

finite set of context-free syntax rules. For each function symbol $f \in \mathcal{F}$ there is an associated context-free syntax rule:

$$A \rightarrow f[B_1, \dots, B_\delta]$$

The *arity* of the rule is δ , and in general we write δ_f for the arity of the rule f . The tree rewriting relation $t : A$ is defined as $f(t_1, \dots, t_\delta) : A$ whenever $t_1 : B_1, \dots, t_\delta : B_\delta$. We say that a tree t is *valid* (for a given category A) if $t : A$.

Example 1. The abstract grammar of a simple fragment of English might look like the following,

$$\begin{aligned} S &\rightarrow s_p[\text{NP}, \text{VP}] \\ S &\rightarrow s_t[\text{NP}, \text{VP}] \\ \text{VP} &\rightarrow vp[\text{V}, \text{NP}] \\ \text{NP} &\rightarrow np[\text{D}, \text{N}] \\ \text{D} &\rightarrow \text{some}[] \\ \text{D} &\rightarrow \text{most}[] \\ \text{N} &\rightarrow \text{cat}[] \\ \text{NP} &\rightarrow \text{fish}[] \\ \text{V} &\rightarrow \text{eat}[] \\ \text{V} &\rightarrow \text{catch}[] \end{aligned}$$

The idea is that the grammar should be able to handle both normal word order ('*most cats eat fish*'), and topicalized sentences ('*it is fish that most cats eat*').

Definition 2 (GCFG, concrete part). *To each category A is associated a linearization type A° , which is not further specified. To each function symbol f is associated a partial linearization function f° , taking as many arguments as the abstract syntax rule specifies:*

$$f^\circ \in B_1^\circ \times \dots \times B_\delta^\circ \rightarrow A^\circ$$

The linearization $\llbracket \cdot \rrbracket$ of syntax trees is defined as,

$$\llbracket f(t_1, \dots, t_\delta) \rrbracket = f^\circ(\llbracket t_1 \rrbracket, \dots, \llbracket t_\delta \rrbracket)$$

if the application is defined. Note that the definition imposes no restrictions on the linearization types or the linearization functions; this is left to the actual grammar formalism. For our purposes it is enough to view a linearization type as the set of all its possible linearization values.

To be able to define the *language* of a grammar as a set of strings, we demand that the linearization type of the starting category is $S^\circ = \Sigma^*$. The language of a grammar G then becomes:

$$\mathcal{L}(G) = \{ \llbracket t \rrbracket \mid t : S \}$$

1.2 Parallel Multiple Context-Free Grammar

Parallel multiple context-free grammar (PMCFG) [1, 12] were introduced in the late 80's by Kasami, Seki et al. as a very expressive formalism, incorporating *linear context-free rewriting systems* and other mildly context-sensitive formalisms, but still with a polynomial parsing algorithm.

Definition 3 (PMCFG). *PMCFG is an instance of GCFG, with the following restrictions on linearizations:*

- *Linearization types are restricted to tuples of strings. In other words, each PMCFG grammar defines a linearization arity $d(C)$ for each category C . The linearization types can then be defined as $C^\circ = (\Sigma^*)^{d(C)}$.*
- *The only allowed operations in linearization functions are tuple projections and string concatenations. In other words, each PMCFG linearization function is of the form,*

$$f^\circ(\langle x_{1,1}, \dots, x_{1,d_1} \rangle, \dots, \langle x_{\delta,1}, \dots, x_{\delta,d_\delta} \rangle) = \langle \alpha_1, \dots, \alpha_d \rangle$$

where each α_i is a sequence of variables $x_{j,k}$ and constant strings.

Example 2. The concrete syntax of the example English grammar might look like follows:

$$\begin{aligned} s_p^\circ(x, \langle y_1, y_2 \rangle) &= x \ y_1 \ y_2 \\ s_t^\circ(x, \langle y_1, y_2 \rangle) &= \text{'it is' } y_2 \ \text{'that' } x \ y_1 \\ vp^\circ(x, y) &= \langle x, y \rangle \\ np^\circ(x, y) &= x \ y \\ most^\circ &= \text{'most'} \\ cat^\circ &= \text{'cats'} \\ fish^\circ &= \text{'fish'} \\ eat^\circ &= \text{'eat'} \\ catch^\circ &= \text{'catch'} \end{aligned}$$

Note that verb phrases have to consist of two discontinuous phrases, for the topicalization to function.¹

1.3 Subclasses of PMCFG

A PMCFG where each variable $x_{i,j}$ occurs in its linearization is called *nonerasing*. If no variable $x_{j,k}$ occurs twice in a linearization the grammar is called a *linear* MCFG (LMCFG or just MCFG). A nonerasing and linear grammar (i.e. if each variable occurs exactly once in its linearization), is called a *linear context-free rewriting system* (LCFRS). The following lemma states that LMCFG and LCFRS are equivalent formalisms [1]:

¹ If the example seems strange, there are other languages (such as German or Swedish) where discontinuous verb phrases are more natural.

Lemma 1. *Any PMCFG grammar can be converted into an equivalent nonerasing grammar. Furthermore, linearity is preserved by the conversion.*

1.4 Literal Movement Grammar and Range Concatenation Grammar

Literal movement grammar (LMG; [2, 3]), and its relative *range concatenation grammar* (RCG; [4, 5]), are grammar formalisms based on *predicates* over string tuples. A grammar is a collection of *clauses* for predicates, very similar to Horn clauses and the programming language Prolog. We here define the general formalism of LMG, and then two equivalent subclasses, RCG and *simple* LMG (sLMG). We assume given a finite set Σ of terminal tokens, and an infinite supply of logical variables $x_1, x_2, \dots \in \text{Var}$.

Definition 4 (predicate). *A predicate is a term $A(\alpha_1, \dots, \alpha_n)$, where each $\alpha_i \in (\Sigma \cup \text{Var})^*$ is a concatenative sequence of terminals and logical variables.*

Definition 5 (clause). *A clause is of the form $\phi \vdash \psi_1, \dots, \psi_m$ where each of $\phi, \psi_1, \dots, \psi_m$ are predicates. A clause can be instantiated by substituting a string for each variable in the clause.*

A literal movement grammar consists a finite number of clauses together with a designated start predicate. To define the language of a LMG grammar G , we define a rewriting relation \Rightarrow_G on sequences of instantiated predicates,

$$\Gamma_1, \phi, \Gamma_2 \Rightarrow_G \Gamma_1, \psi_1, \dots, \psi_m, \Gamma_2$$

whenever $\phi \vdash \psi_1, \dots, \psi_m$ is an instantiation of a clause in G . The language of a grammar is then,

$$\mathcal{L}(G) = \{ w \in \Sigma^* \mid S(w) \Rightarrow_G^* \epsilon \}$$

where S is the start predicate in G .

Example 3. The example grammar looks like follows in LMG format:

$$\begin{aligned} S(x \ y_1 \ y_2) &\vdash \text{NP}(x), \text{VP}(y_1, y_2) \\ S(\text{'it is' } y_2 \ \text{'that' } x \ y_1) &\vdash \text{NP}(x), \text{VP}(y_1, y_2) \\ \text{VP}(x, y) &\vdash \text{V}(x), \text{NP}(y) \\ \text{NP}(x \ y) &\vdash \text{D}(x), \text{N}(y) \\ \text{D}(\text{'most'}) &\vdash \epsilon \\ \text{N}(\text{'cats'}) &\vdash \epsilon \\ \text{NP}(\text{'fish'}) &\vdash \epsilon \\ \text{V}(\text{'eat'}) &\vdash \epsilon \\ \text{V}(\text{'catch'}) &\vdash \epsilon \end{aligned}$$

A possible instantiation of the second clause is:

$$S(\text{'it is fish that most cats eat'}) \vdash \text{NP}(\text{'most cats'}), \text{VP}(\text{'eat'}, \text{'fish'})$$

LMG is a very general, Turing-complete, grammar formalism. To get a recognizable subclass of LMG, one can consider two possibilities; to restrict the definition of clause instantiation, or to put syntactic restrictions on the form of the predicates.

Definition 6 (RCG). *A range concatenation grammar (RCG) is an LMG with a restricted form of clause instantiation. A clause can only be instantiated by substrings of the given input string; i.e. if $\phi \vdash \psi_1, \dots, \psi_m$ is an instantiation of a clause, then all arguments to $\phi, \psi_1, \dots, \psi_m$ are substrings of the input.*

By only allowing instantiations by substrings of the input we assure that all strings in a RCG can be replaced by pairs of input positions, called *ranges*.² This has the effect that RCG parsing is polynomial in the length of the input string.

Example 4. If the input string is ‘*b a c h*’, then for the following clauses,

$$\begin{aligned} A(bac) &\vdash B(b), C(c) \\ A(bach) &\vdash B(b), C(ch) \\ A(back) &\vdash B(b), C(ck) \end{aligned}$$

the first two are RCG instantiations of the clause $A(x \ a \ z) \vdash B(x), C(z)$; but not the third.

Definition 7 (sLMG). *A simple LMG (sLMG) is an LMG where each clause obeys the following three syntactic restrictions:*

- **Non-combinatorial (NC):** *The arguments of each ψ_i are variables.*
- **Bottom-up nonerasing (BNE):** *All variables in each ψ_i also occur in ϕ .*
- **Bottom-up linear (BL):** *No variable occurs more than once in ϕ .*

Strictly speaking, bottom-up linearity is not a necessary condition, as the following lemma states:

Lemma 2. *Any LMG clause can be converted to an equivalent bottom-up linear (BL) clause. Furthermore, the conversion preserves NC and BNE.*

Proof (taken from [2, 3]). Assume that the clause in question is $\phi \vdash \psi_1, \dots, \psi_m$, and that there is a variable x occurring twice in ϕ . Replace one occurrence by a new variable x' , and add a call to the bottom-up linear predicate $\text{Eq}(x, x')$, with the following definition:

$$\begin{aligned} \text{Eq}(\epsilon, \epsilon) &\vdash \epsilon \\ \text{Eq}(s \ x, s \ y) &\vdash \text{Eq}(x, y) \quad (\text{for each } s \in \Sigma) \end{aligned}$$

The new clause $\phi \vdash \psi_1, \dots, \psi_m, \text{Eq}(x, x')$ is equivalent to the original, since the predicate call $\text{Eq}(x, x')$ says that x and x' are equal strings.

² Boullier [4, 5] defines RCG directly on ranges, but our definition is equivalent.

The conversion preserves NC, since the predicate $\text{Eq}(x, x')$ is non-combinatorial. Furthermore, it preserves BNE, since the only variable that is introduced on the left-hand side (x') is also introduced on the right-hand side. \square

Both formalisms sLMG and RCG are equivalent, since they describe exactly the class of languages recognizable in polynomial time [2, 3, 4, 5, 13]. Note that sLMG/RCG are closed under intersection; if S_1 and S_2 are the start predicates of G_1 and G_2 , then $S(x) \vdash S_1(x), S_2(x)$ defines the intersection of the languages $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$.

1.5 PMCFG Is an Instance of sLMG/RCG

Assume given the following PMCFG rule:

$$\begin{aligned} A &\rightarrow f[B_1, \dots, B_\delta] \\ f^\circ(x_{1,1}, \dots, x_{1,n_1}; \\ &\quad \dots; \\ x_{\delta,1}, \dots, x_{\delta,n_\delta}) &= \alpha_1, \dots, \alpha_n \end{aligned}$$

By lemma 1, we can assume that the linearization is nonerasing. Furthermore, it is straightforward to convert a nonerasing PMCFG grammar into an equivalent sLMG grammar, as shown in [2, 3]. Each rule above is converted to the clause:

$$\begin{aligned} A(\alpha_1, \dots, \alpha_n) &\vdash B_1(x_{1,1}, \dots, x_{1,n_1}), \\ &\dots, \\ &B_\delta(x_{\delta,1}, \dots, x_{\delta,n_\delta}) \end{aligned}$$

Note that this clause is NC (since each of the $x_{i,j}$ is a variable) and BNE (since f° is nonerasing), and therefore the clause is sLMG.

2 The Intersection Operation

There is an extension of context-free grammar called *conjunctive grammar* [6], where the right-hand sides of rules are extended with a new intersection operator. A conjunctive context-free rule is written:

$$A \rightarrow \alpha_1 \& \dots \& \alpha_n$$

where $\alpha_i \in (N \cup \Sigma)^*$. The informal interpretation is that A can be rewritten to $w \in \Sigma^*$ iff all α_i can be rewritten to w . This operation can be directly transformed to PMCFG linearizations.

Definition 8 (intersection). *The intersection operation is a partial linearization operation with the definition; $\phi_1 \& \phi_2$ is calculated to ϕ_1 iff $\phi_1 = \phi_2$.*

This definition can be made formal by lifting the linearization types to sets of linearization values; where the unit set denotes the existence of a linearization

and the empty set denotes an undefined linearization. String concatenation, tuple forming and tuple projection are straightforwardly lifted to this domain. The definition of the intersection operation then simply becomes set intersection.

We call PMCFG extended with the intersection operation *conjunctive* PMCFG. The following laws for intersections of linearizations are simple consequences of the formal definition:

$$\begin{aligned}\phi \& \phi &= \phi \\ \alpha (\beta_1 \& \beta_2) \gamma &= (\alpha \beta_1 \gamma) \& (\alpha \beta_2 \gamma)\end{aligned}$$

The second law says that we can push out an intersection to a row, which is used in the equivalence proof later.

Example 5. In our running example, we have introduced discontinuous verb phrases to handle topicalization. Groenink [2, 3] suggests to handle verb phrase coordination by using conjunction on the verb component of the verb phrase. In PMCFG format, this looks like follows:

$$\begin{aligned}\text{VP} &\rightarrow \text{coord}[\text{VP}, \text{VP}] \\ \text{coord}^\circ(\langle x_1, x_2 \rangle, \langle y_1, y_2 \rangle) &= \langle x_1 \text{ 'and' } y_1, x_2 \& y_2 \rangle\end{aligned}$$

By combining two verb phrases with the same object, we can form a coordinated verb phrase:

$$\text{coord}^\circ(\langle \text{'catch'}, \text{'fish'} \rangle, \langle \text{'eat'}, \text{'fish'} \rangle) = \langle \text{'catch and eat'}, \text{'fish'} \rangle$$

which in turn can be used to form sentences like ‘many cats catch and eat fish’, or the topicalized version ‘it is fish that many cats catch and eat’.

2.1 A Strict Extension of PMCFG

Theorem 1. *The class of languages recognized by conjunctive PMCFG grammars is closed under intersection.*

Proof. Let G_1 and G_2 be two grammars (with no common categories or function symbols) recognizing the languages $\mathcal{L}(G_1)$ and $\mathcal{L}(G_2)$ respectively. Let G contain all rules from G_1 and G_2 plus the following single rule for the new starting category S :

$$\begin{aligned}S &\rightarrow f[S_1, S_2] \\ f^\circ(x, y) &= x \& y\end{aligned}$$

It is trivial to see that G recognizes all and only those strings that are recognized by both G_1 and G_2 . \square

Corollary 1. *The intersection operation is a strict extension of PMCFG.*

The corollary follows from the fact that PMCFG is not closed under intersection [1], a property it shares with context-free grammars.

2.2 Language-Theoretic Implications

Closedness under intersection has some less desirable properties, which conjunctive PMCFG inherits from *conjunctive grammar* [6]:

- The following decision problems are undecidable: emptiness, finiteness, regularity, context-freeness, inclusion and equivalence. This is because these decision problems are undecidable for finite intersections of context-free grammars, see e.g. [14].
- Conjunctive PMCFG is not closed under homomorphism. This follows from the fact that any recursively enumerable language \mathcal{L} can be described by $h(\mathcal{L}_1 \cap \mathcal{L}_2)$, for some homomorphism h and context-free languages $\mathcal{L}_1, \mathcal{L}_2$, see e.g. [15].

2.3 Usefulness of Intersection

Conjunctive PMCFG is not only closed under intersection, but the closure is also *modular*, i.e. it preserves the structure of the underlying grammar conjuncts. This makes it useful for modular grammar engineering, as already noted in [4, 5]. Intersection might also be useful for modeling secondary/tertiary structures of biological sequences, as has been investigated in [16]. For purely linguistic phenomena, [2, 3] contains a suggestion of how to use intersection to describe verb coordination, as shown in example 5.

3 Conjunctive PMCFG Describes the Polynomial Languages

In this section we show that conjunctive PMCFG, or to be more exact, non-erasing conjunctive PMCFG, is equivalent to sLMG and RCG. The following theorem is a direct consequence of lemmas 3, 4 and 5 below:

Theorem 2. *Nonerasing conjunctive PMCFG is equivalent to sLMG and RCG.*

Since it is already known that sLMG and RCG exactly describe the class of languages recognizable in polynomial time, we get the same result for nonerasing PMCFG extended with intersection.

Corollary 2. *The class of languages recognizable by nonerasing conjunctive PMCFG is exactly the class of languages recognizable in polynomial time.*

3.1 Conjunctive PMCFG Is an Instance of sLMG/RCG

Lemma 3. *Any nonerasing conjunctive PMCFG can be converted to an equivalent sLMG.*

Proof. Since intersections can be pushed out, we can assume that the PMCFG rules are of the form,

$$\begin{aligned}
 A &\rightarrow f[B_1, \dots, B_\delta] \\
 f^\circ(\langle x_{1,1}, \dots, x_{1,n_1} \rangle, & \\
 \dots, & \\
 \langle x_{\delta,1}, \dots, x_{\delta,n_\delta} \rangle) &= \langle \alpha_{1,1} \& \dots \& \alpha_{1,c_1}, \\
 \dots, & \\
 \alpha_{n,1} \& \dots \& \alpha_{n,c_n} \rangle &
 \end{aligned}$$

where each $\alpha_{i,j}$ is a sequence of strings and variables, as above. Translate this to the sLMG clause,

$$\begin{aligned}
 \hat{A}(\alpha_{1,1} \& \dots \& \alpha_{1,c_1}; & \\
 \dots; & \\
 \alpha_{n,1} \& \dots \& \alpha_{n,c_n}) \vdash B_1(x_{1,1}, \dots, x_{1,n_1}), & \\
 \dots, & \\
 B_\delta(x_{\delta,1}, \dots, x_{\delta,n_\delta}) &
 \end{aligned}$$

where the left-hand side is just syntactic sugar for a predicate with arity $c_1 + \dots + c_n$. The clause is NC (since each of the $x_{i,j}$ is a variable) and BNE (since f° is nonerasing), and therefore it is sLMG. Finally, add coercion clauses for $\hat{A}(\dots)$, implementing the intersections:

$$A(x_1, \dots, x_n) \vdash \hat{A}(x_1 \& \dots \& x_1; \dots; x_n \& \dots \& x_n)$$

The resulting sLMG grammar is equivalent to the original PMCFG grammar. \square

Example 6. The following is the result of translating the PMCFG rule for verb coordination in example 5, into sLMG/RCG:

$$\begin{aligned}
 \widehat{VP}(x_1 \text{ 'and' } y_1; x_2 \& y_2) \vdash VP(x_1, x_2), VP(y_1, y_2) \\
 VP(x, y) \vdash \widehat{VP}(x; y \& y)
 \end{aligned}$$

After simplifying away \widehat{VP} , we get the same clause as in Groenink's original example [2, 3]:

$$VP(x \text{ 'and' } y, z) \vdash VP(x, z), VP(y, z)$$

3.2 sLMG/RCG is an Instance of Conjunctive PMCFG

We say that a clause $\phi \vdash \psi_1, \dots, \psi_m$ is *top-down nonerasing* (TNE) if all variables in ϕ also occur in some ψ_i .

Lemma 4. *Any LMG clause can be converted to an equivalent top-down non-erasing (TNE) clause. Furthermore, the conversion preserves NC and BNE.*

Proof. Assume that the clause in question is $\phi \vdash \psi_1, \dots, \psi_m$, and that there is a variable x in ϕ not occurring in any of ψ_1, \dots, ψ_m . Add a call to the top-down nonerasing predicate $\mathbf{Str}(x)$, with the following definition:

$$\begin{aligned} \mathbf{Str}(\epsilon) &\vdash \epsilon \\ \mathbf{Str}(s\ x) &\vdash \mathbf{Str}(x) \quad (\text{for each } s \in \Sigma) \end{aligned}$$

The new clause $\phi \vdash \psi_1, \dots, \psi_m, \mathbf{Str}(x)$ is equivalent to the original, since the predicate $\mathbf{Str}(x)$ only says that x is a string.

The conversion preserves NC, since the predicate $\mathbf{Str}(x)$ is non-combinatorial. Furthermore, it preserves BNE, since no variable is introduced. \square

Lemma 5. *Any top-down nonerasing sLMG can be converted to an equivalent nonerasing conjunctive PMCFG.*

Proof. A sLMG clause is of the following form:

$$\begin{aligned} A(\alpha_1, \dots, \alpha_n) &\vdash B_1(x_{1,1}, \dots, x_{1,n_1}), \\ &\dots, \\ &B_\delta(x_{\delta,1}, \dots, x_{\delta,n_\delta}) \end{aligned}$$

If the variables $x_{i,j}$ all are distinct, it is equivalent to the PMCFG rule:

$$\begin{aligned} A &\rightarrow f[B_1, \dots, B_\delta] \\ f^\circ(\langle x_{1,1}, \dots, x_{1,n_1} \rangle, & \\ \dots, & \\ \langle x_{\delta,1}, \dots, x_{\delta,n_\delta} \rangle) &= \langle \alpha_1, \dots, \alpha_n \rangle \end{aligned}$$

However, in sLMG, the variables in the right-hand side of a clause need not be distinct. Assume therefore that $x_{i',j'} = x_{i,j} = x$. Now, introduce a new variable x' to replace x as $x_{i',j'}$; and replace each occurrence of x in the right-hand side with the conjunction $(x \& x')$.

The resulting rule is a syntactically correct conjunctive PMCFG rule, and equivalent to the given sLMG clause. Furthermore, it is nonerasing since the original clause is TNE. \square

Example 7. The resulting clause from the previous example,

$$\mathbf{VP}(x \text{ 'and' } y, z) \vdash \mathbf{VP}(x, z), \mathbf{VP}(y, z)$$

is converted to the following conjunctive PMCFG rule:

$$\begin{aligned} \mathbf{VP} &\rightarrow f[\mathbf{VP}, \mathbf{VP}] \\ f^\circ(\langle x, z \rangle, \langle y, z' \rangle) &= \langle x \text{ 'and' } y, z \& z' \rangle \end{aligned}$$

4 Discussion

The results in this and earlier papers [1, 2, 3, 4, 5, 11, 17] give some insights into the nature of the class of polynomial time recognizable languages. We can now try to describe what kind of constructions are necessary (and sufficient) to be able to describe any polynomial language, apart from ordinary string concatenation.

Multiple Constituents. Parse time complexity is directly related to the maximal number of discontinuous constituents in a grammar [2, 3, 4, 5, 11, 17]. Therefore there should be some way of coding discontinuous constituents in a grammar, be it with string tuples as in the formalisms discussed in this paper, or with some kind of ingenious coding.

Reduplication. The exponentially growing language a^{2^n} is polynomially recognizable, and we see no other (simple) way of describing that language but to use string duplication – there can only be a finite number of multiple constituents in a grammar, and an intersection cannot be used to duplicate strings.

Intersection. As noted by Boullier [4, 5], the intersection of two polynomially parsable languages is also polynomially parsable – simply recognize for each language in turn. And since formalisms with multiple constituents and reduplication (e.g. PMCFG) are not closed under intersection, we have to introduce intersection as an explicit operation.

Suppose we design a new grammar formalism having some construction which cannot be decomposed into these constructions. Then our conjecture is that the formalism cannot be parsable in polynomial time.

One such construction which we have already come across is *erasing* PMCFG grammars – the possibility to erase linearization information of parts of the syntax tree, as discussed in section 1.3. Without the intersection operation, any erasing grammar can be transformed into an equivalent nonerasing grammar. But for conjunctive PMCFG, it is not clear whether erasing syntax rules can be transformed away. Either it is possible, in which case any conjunctive PMCFG is polynomially parsable; or it is not possible, in which case conjunctive PMCFG is not polynomial in general.

References

1. Seki, H., Matsumara, T., Fujii, M., Kasami, T.: On multiple context-free grammars. *Theoretical Computer Science* **88** (1991) 191–229
2. Groenink, A.: Mild context-sensitivity and tuple-based generalizations of context-free grammar. *Linguistics and Philosophy* **20** (1997) 607–636
3. Groenink, A.: Surface without Structure — Word order and tractability issues in natural language analysis. PhD thesis, Utrecht University (1997)
4. Boullier, P.: A cubic-time extension of context-free grammars. *Grammars* **3** (2000) 111–131

5. Boullier, P.: Range concatenation grammars. In: 6th International Workshop on Parsing Technologies, Trento, Italy (2000) 53–64
6. Okhotin, A.: Conjunctive grammars. *Journal of Automata, Languages and Combinatorics* **6** (2001) 519–535
7. Pollard, C.: Generalised Phrase Structure Grammars, Head Grammars and Natural Language. PhD thesis, Stanford University (1984)
8. Weir, D.: Characterizing Mildly Context-Sensitive Grammar Formalisms. PhD thesis, University of Pennsylvania, Philadelphia, PA (1988)
9. Becker, T.: HyTAG: A New Type of Tree Adjoining Grammars. PhD thesis, Universität des Saarlandes (1994)
10. Chiang, D.: Constraints on strong generative power. In: 39th Meeting of the Association for Computational Linguistics. (2001) 124–131
11. Ljunglöf, P.: Expressivity and Complexity of the Grammatical Framework. PhD thesis, Göteborg University (2004)
12. Kasami, T., Seki, H., Fujii, M.: Generalized context-free grammars and multiple context-free grammars. *IEICE Transactions* **J71-D-I** (1988) 758–765
13. Bertsch, E., Nederhof, M.J.: On the complexity of some extensions of RCG parsing. In: 7th International Workshop on Parsing Technologies. (2001) 66–77
14. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
15. Ginsburg, S.: Algebraic and Automata-Theoretic Properties of Formal Languages. North-Holland/Elsevier (1975)
16. Chiang, D.: Evaluating Grammar Formalisms for Applications to Natural Language Processing and Biological Sequence Analysis. PhD thesis, University of Pennsylvania (2004)
17. Satta, G.: Recognition of linear context-free rewriting systems. In: 30th Meeting of the Association for Computational Linguistics, Newark, Delaware (1992) 89–95

Learnable Classes of General Combinatory Grammars

Erwan Moreau

LINA - FRE CNRS 2729 - Université de Nantes,
2 rue de la Houssinière - BP 92208 - 44322 Nantes cedex 3
`Erwan.Moreau@univ-nantes.fr`

Abstract. Kanazawa has shown that k -valued classical categorial grammars have the property of finite elasticity [1], which is a sufficient condition for learnability. He has also partially extended his result to general combinatory grammars, but left open the question of whether some subsets of general combinatory grammars have finite elasticity. We propose a new sufficient condition which implies learnability of some classes of k -valued general combinatory grammars, focusing on the way languages are expressed through a grammatical formalism rather than the classes of languages themselves.

1 Introduction

The problem of grammatical inference refers to the process of learning grammars and/or languages from data. Applied to natural languages, this problem consists in guessing, from a set of data corresponding to a (natural) language, “something which represents this language”: a grammar. But what kind of grammar ? In this article we propose to study learnability of the formalisms used to represent languages rather than the languages themselves. This question is especially important for natural languages, since a lot of formalisms exist to represent them. As a consequence, the fact that a class of natural languages is learnable is only interesting if there is a way to represent it in a usable and linguistically appropriate way.

Gold’s model of identification in the limit is one of the most important formalizations of the learning process [2]. In this model, the learner must be able to guess the right language after a finite number of examples, from an infinite set of sentences belonging to this language. Several positive results have been obtained in Gold’s model, in particular with categorial grammars: using Buszkowski’s learning algorithm [3] for classical categorial grammars (also called AB grammars), Kanazawa [1] has shown that k -valued AB grammars¹ are learn-

¹ A grammar is *k-valued* if each words is defined by at most k different types in the lexicon. In the special case where $k = 1$, the grammar is said to be *rigid*. This latter case is a strong restriction over the expressive power of the corresponding language: Kanazawa has shown that the class of k -valued AB grammars languages is strictly included in the class of $(k + 1)$ -valued languages. Furthermore, there are words in natural languages that require several definitions: for example, the grammatical word “to” should not have the same type when it is used with an infinitive or used as a preposition.

able from strings. More precisely, Kanazawa has shown that the class of structure languages generated by rigid AB grammars has the property of finite elasticity ([4], [5]), which implies learnability and also that learnability can be extended to k -valued grammars and to string languages. Kanazawa has also generalized his learnability result to *general combinatory grammars*: in this framework it is possible to consider any set of operators and any set of universal rules instead of the usual AB grammars operators and rules. However Kanazawa shows only that rigid structure languages are learnable, because finite elasticity does not hold in this more general case. As a consequence, Kanazawa does not provide any positive learnability result about k -valued general combinatory grammars, which is a strong limitation to the usefulness of this result in the viewpoint of natural languages.

The question of whether some subsets of general combinatory grammars have finite elasticity was left open by Kanazawa. Costa Florêncio has given in [6] a sufficient condition for finite elasticity in the form of restrictions over rules. Here we propose another sufficient condition which does not include Costa Florêncio's one, but which relates more to grammatical formalisms than to technical constraints (a brief comparison between the two results is given in section 4.2).

The linguistic interest for general combinatory grammars lies in the fact that it allows to express various formalisms through the set of universal rules: one can see the set of rules (and operators) as a parameter of the class of languages which is studied. This point is particularly interesting in the framework of grammatical inference, because it permits to test rather easily whether a given grammatical formalism enjoys some learnability properties: if the formalism can be expressed using rules in the form of general combinatory grammars and if these rules fulfill the conditions described in section 3.2, then learnability is proven. Several examples of such formalisms which are (at least partially) learnable are given in section 4.

2 General Combinatory Grammars

The name “general combinatory grammars” is used by Kanazawa to define any class of grammars using a certain set of operators and universal rules (expressed as the rewriting of a sequence of terms containing variables into another term). It refers to *combinatory categorial grammars*, defined by Steedman [7], who proposed to add several rules to AB grammars, in order to give a better syntactic description of natural languages. This means that AB grammars, as well as combinatory categorial grammars, are instances of general combinatory grammars.

Definition 1 (Terms). *Given a set \mathcal{S} of operators and a set \mathcal{V} of variables, the set of \mathcal{S} -terms over \mathcal{V} is the smallest set such that*

- any $v \in \mathcal{V}$ is an \mathcal{S} -term over \mathcal{V} ,
- for any operator $f \in \mathcal{S}$ with $\text{arity}(f) = n$, if t_1, \dots, t_n are \mathcal{S} -terms over \mathcal{V} then $f(t_1, \dots, t_n)$ is an \mathcal{S} -term over \mathcal{V} .²

The size $\|t\|$ and height $h(t)$ of a term t are defined in the usual way: if t is a variable then $\|t\| = 1$ and $h(t) = 0$. Otherwise if $t = f(t_1, \dots, t_n)$ then $\|t\| = \sum_{1 \leq i \leq n} \|t_i\| + 1$ and $h(t) = \max(\{h(t_i) \mid 1 \leq i \leq n\}) + 1$.

$\#_u(t)$ denotes the number of occurrences of a term u in a term t :

- if $u = t$ then $\#_u(t) = 1$
- otherwise $u \neq t$:
 - if t is a variable, then $\#_u(t) = 0$
 - if $t = f(t_1, \dots, t_n)$ then $\#_u(t) = \sum_{1 \leq i \leq n} \#_u(t_i)$

Definition 2 (Universal rule). Let \mathcal{S} be a set of operators. Given a set of variables $\text{Var}(R)$, a universal rule R over \mathcal{S} is any expression of the form $A_1, \dots, A_n \rightarrow A_0$, where each A_i is an \mathcal{S} -type over $\text{Var}(R)$.

Definition 3 (\mathcal{R} -grammar). Let \mathcal{S} be a set of operators and \mathcal{R} a set of universal rules over \mathcal{S} . An \mathcal{R} -grammar is a system $G = \langle \Sigma, Pr, s, \triangleright \rangle$ where

- Σ is the vocabulary,
- Pr is a finite set of variables, called primitive types. The set of types Tp is then defined as the set of \mathcal{S} -terms over Pr .
- s is an \mathcal{S} -term over \emptyset : this is the special type for valid sentences (see definition 8).³
- \triangleright is a binary relation assigning one or several types to each word in the vocabulary: $\triangleright \subseteq \Sigma \times Tp$. Each couple $w \triangleright t$ in this relation is called a lexical rule.

Remark: In the framework of categorial grammars, the special type s is traditionally defined as one of the primitive types. On the contrary, we can benefit here of a more general definition of the set of operators \mathcal{S} , which permits to define s as a type using only these operators. This way it is really considered different from the other types, which is particularly relevant in the framework of grammatical inference. Furthermore, this definition permits to take into account new interesting formalisms (see example in 4.3).

$\text{Lex}(G)$ is defined as the set of types used in the lexicon: $\text{Lex}(G) = \{t \in Tp \mid \text{there exists } w \text{ such that } w \triangleright t\}$.

² The special case where $\text{arity}(f) = 0$ is included in this definition: if f is such an operator, then it is an \mathcal{S} -term over \mathcal{V} .

³ *Remark:* s is an \mathcal{S} -term over the empty set, which implies that there is at least one operator f in \mathcal{S} such that $\text{arity}(f) = 0$.

Definition 4 (One-step derivation). Let \mathcal{S} be a set of operators, \mathcal{R} a set of rules over \mathcal{S} and G an \mathcal{R} -grammar. For every rule $R \in \mathcal{R}$, $R = A_1, \dots, A_n \rightarrow A_0$, the relation $\rightarrow_R \subseteq Tp^+ \times Tp$ is defined as: $t_1, \dots, t_n \rightarrow_R t_0$ if and only if there exists a substitution $\sigma : Var(R) \rightarrow Tp$ such that $\sigma(A_i) = t_i$ for all i .

The relation \rightarrow is defined as $t_1, \dots, t_n \rightarrow t_0$ if there exists a rule $R \in \mathcal{R}$ such that $t_1, \dots, t_n \rightarrow_R t_0$.

From this definition it is possible to define the string derivation relation in the usual following way: let \Rightarrow be the relation defined as $\alpha \beta \gamma \Rightarrow \alpha t_0 \gamma$ if and only if $\beta \rightarrow t_0$, with $\alpha, \gamma \in Tp^*$ and $\beta \in Tp^+$. The relation \Rightarrow^* is defined as the reflexive and transitive closure of \Rightarrow . A string w_1, \dots, w_n is valid for the \mathcal{R} -grammar G if there exists a sequence of types t_1, \dots, t_n such that for all i $w_i \triangleright t_i$ and $t_1, \dots, t_n \Rightarrow^* s$.

Nevertheless, we will rather use the below definition that links derivation to the existence of a “structure”, because we are also interested in structure languages.

Definition 5 (\mathcal{R} -structure). Let \mathcal{S} be a set of operators and \mathcal{R} a set of rules over \mathcal{S} , $\mathcal{R} = \{R_1, \dots, R_m\}$. Given a vocabulary Σ , the set of \mathcal{R} -structures $\mathcal{SL}_{\mathcal{R}}$ is the smallest set such that

- any word $w \in \Sigma$ belongs to $\mathcal{SL}_{\mathcal{R}}$,
- for any rule $R_i = A_1, \dots, A_n \rightarrow A_0$,
if $T_1, \dots, T_n \in \mathcal{SL}_{\mathcal{R}}$ then $[R_i](T_1, \dots, T_n) \in \mathcal{SL}_{\mathcal{R}}$.

Definition 6 (Yield of an \mathcal{R} -structure). Let \mathcal{S} be a set of operators and \mathcal{R} a set of rules over \mathcal{S} . The yield of an \mathcal{R} -structure T , denoted $yield(T)$, is the sequence of words occurring at it leaves. Formally:

- if $T = w$ with $w \in \Sigma$, then $yield(T) = w$
- if $T = [R_i](T_1, \dots, T_n)$, then $yield(T) = yield(T_1), \dots, yield(T_n)$.

Definition 7 (Instance of an \mathcal{R} -structure). Let \mathcal{S} be a set of operators and \mathcal{R} a set of rules over \mathcal{S} and G an \mathcal{R} -grammar. Given an \mathcal{R} -structure T , a couple $\langle \alpha, t_0 \rangle$, where $\alpha \in Tp^+$ and $t_0 \in Tp$, is an instance of T for G if the following condition holds:

- if $T = w$ with $w \in \Sigma$, then $\alpha = t_0$ and $w \triangleright t_0$.
- if $T = [R_i](T_1, \dots, T_n)$, then there exist n couples $\langle \alpha_1, t_1 \rangle, \dots, \langle \alpha_n, t_n \rangle$ such that $\langle \alpha_i, t_i \rangle$ is an instance of T_i for all i , $\alpha = \alpha_1 \bullet \dots \bullet \alpha_n$ and $t_1, \dots, t_n \rightarrow_{R_i} t_0$.

Definition 8 (Languages of a grammar). Let \mathcal{S} be a set of operators, \mathcal{R} a set of rules over \mathcal{S} and $G = \langle \Sigma, Pr, s, \triangleright \rangle$ an \mathcal{R} -grammar.

- An \mathcal{R} -structure T belongs to the structure language defined by G , denoted $T \in SL(G)$, if there exists an instance $\langle \alpha, s \rangle$ of T for G .

- A string (sequence of words) $w_1 \dots w_n$ belongs to the string language defined by G , denoted $w_1 \dots w_n \in L(G)$ if there exists an \mathcal{R} -structure $T \in SL(G)$ such that $w_1 \dots w_n \in \text{yield}(T)$.

Example 1 (AB grammars). Let define the set of operators \mathcal{S}_{AB} as $\mathcal{S}_{AB} = \{/(2), \backslash(2), s(0)\}$ and the set of rules \mathcal{R}_{AB} as the set containing only the two following rules:

$$\begin{array}{lll} FA : & A/B \ B \rightarrow A & \text{Var}(FA) = \{A, B\} \\ BA : & B \ B \backslash A \rightarrow A & \text{Var}(BA) = \{A, B\} \end{array}$$

The class of \mathcal{R}_{AB} -grammars corresponds exactly to the class of AB grammars.

Let G be the \mathcal{R}_{AB} -grammar defined with the lexicon $\{ \text{Peter} \triangleright n ; \text{Mary} \triangleright n ; \text{loves} \triangleright (n \backslash s)/n \}$. the \mathcal{R}_{AB} -structure $T = BA(\text{Peter}, FA(\text{loves}, \text{Mary}))$ belongs to $SL(G)$, because $\langle (n, (n \backslash s)/n, n), s \rangle$ is an instance of T for G . Since $\text{yield}(T) = \{ (\text{Peter loves Mary}) \}$, the sentence “Peter loves Mary” belongs to $L(G)$.

Definition 9 ($\#G$). We denote the number of elements in a set E by $\#E$. Applied to a grammar $G = \langle \Sigma, Pr, s, \triangleright \rangle$, the set considered is the set of the lexical rules in G :

$$\#G = \# \{ (w, t) \text{ with } w \in \Sigma \text{ and } t \in Tp \mid w \triangleright t \}$$

As a consequence of this definition, a grammar G is included in a grammar G' , denoted $G \subseteq G'$, if all lexical rules in G are also defined in G' .

Proposition 1. Let G and G' be two \mathcal{R} -grammars. If $G \subseteq G'$ then $SL(G) \subseteq SL(G')$.

Proof. Let $T \in SL(G)$: there exists an instance $\langle t_1 \dots t_n, s \rangle$ of T for G . $G \subseteq G'$ implies that if $w_i \triangleright_G t_i$ then $w_i \triangleright_{G'} t_i$, so $\langle t_1 \dots t_n, s \rangle$ is also an instance of T for G' , thus $T \in SL(G')$. \square

Definition 10 (k -valued grammar). $G = \langle \Sigma, Pr, s, \triangleright \rangle$ is k -valued if for any word $w \in \Sigma$: $\# \{ t \in Tp \mid w \triangleright t \} \leq k$.

Proposition 2. For any $k \geq 0$ there exists $k' \geq 0$ such that

$$\{ G \mid G \text{ is } k\text{-valued} \} \subseteq \{ G \mid \#G \leq k' \}$$

Proof. Clearly any k -valued grammar can not have more than $k' = k \times \#\Sigma$ rules. \square

Definition 11 (Equivalent grammars). Two \mathcal{R} -grammars G and G' are said to be equivalent if they differ only by a renaming of their primitive types.

Proposition 3. If G and G' are equivalent, then $SL(G) = SL(G')$ (and as a consequence $L(G) = L(G')$).

3 k -Valued Flat Grammars Are Learnable from Structures

The distinction between *functors* and *arguments* is important in the first learning algorithm for AB grammars defined by Buszkowski [3], and also in the extensions given by Kanazawa [1]. However it is not the main point used by Kanazawa to show finite elasticity of k -valued AB grammars. Here we propose to focus on this distinction and its consequences in the learnability viewpoint, and see how this particular property of AB grammars can be generalized. Thus we obtain a sufficient condition for learnability of grammars through the criterion of “*flatness*”.

To show that such languages are learnable we will use Shinohara’s criterion of *bounded finite thickness*. The general framework proposed by Shinohara has the advantage to take into account the way languages are represented in the formalization of learning. Actually this is not the case in the basic definition of Gold’s identification in the limit, nor in Wright’s criterion of finite elasticity, where the fact that languages must have a grammatical representation is only implicit. But in the natural languages viewpoint this point is essential. In particular, it is probably more interesting to be able to test whether a given grammatical formalism has some learnability properties (due to the formalism itself) than to know if it contains a learnable subclass of languages.

3.1 Gold’s Model of Identification in the Limit

In the following we use the abstract word *objects* to refer to the elements of a language. These objects may be strings (that is simple sequences of words), but also structures which may be more or less complex.

Gold’s model of identification in the limit is a formal model of learning [2]. In this model, the learner has to guess the right language from an infinite sequence of objects belonging to this language (positive examples). Formally, let ϕ be a learning function, and L a language. Let $\langle a_i \rangle_{i \in \mathbb{N}}$ be any infinite sequence of objects such that $a \in \langle a_i \rangle_{i \in \mathbb{N}}$ if and only if $a \in L$. ϕ *converges* to L if there exists $n \in \mathbb{N}$ such that $\phi(\langle a_1, a_2, \dots, a_n \rangle) = L$, and for all $i > n$ $\phi(\langle a_1, a_2, \dots, a_i \rangle) = L$. A class of languages \mathcal{L} is learnable if there exists a learning function ϕ such that for all $L \in \mathcal{L}$, ϕ converges to L for any enumeration of L .

Wright has proposed in [4], [5] a sufficient condition for learnability, called *finite elasticity*. A class \mathcal{L} has infinite elasticity if there exist two infinite sequences a_0, a_1, \dots of objects and L_1, L_2, \dots of languages such that for any $k \geq 1$ $\{a_0, a_1, \dots, a_{k-1}\} \subseteq L_k$ but $a_k \notin L_k$. A class \mathcal{L} has finite elasticity if \mathcal{L} does not have infinite elasticity. Kanazawa has shown an important theorem about finite elasticity in [1]: if a class \mathcal{L}_1 has finite elasticity and there exists a *finite-valued*⁴ relation between \mathcal{L}_1 and \mathcal{L}_2 , then \mathcal{L}_2 has also finite elasticity.

Shinohara proposed in [8] a framework in which languages are defined through a set of expressions, called a formal system. His definition of formal systems

⁴ A relation $R \subseteq \mathcal{U}_1 \times \mathcal{U}_2$ is *finite-valued* iff for every $a \in \mathcal{U}_1$ there are at most finitely many $b \in \mathcal{U}_2$ such that Rab .

corresponds to a general definition of grammars, except that these grammars must consist of a set of elements (called expressions), that correspond usually to rules: a *concept defining framework* is a triple $\langle \mathcal{U}, \mathcal{E}, \mathcal{M} \rangle$ of a universe \mathcal{U} of objects, a set \mathcal{E} of expressions and a semantic mapping \mathcal{M} that maps finite subsets of \mathcal{E} (grammars) to subsets of \mathcal{U} (languages). A semantic mapping \mathcal{M} is monotonic if $G \subseteq G'$ implies $\mathcal{M}(G) \subseteq \mathcal{M}(G')$.

Example 2. Let \mathcal{S}_{AB} and \mathcal{R}_{AB} be the sets of AB grammars operators and rules, as defined in example 1. Given a vocabulary Σ , let $\mathcal{E}_{\mathcal{R}}$ be the set of all possible lexical rules $w \triangleright t$, with $w \in \Sigma$ and t an \mathcal{S}_{AB} -term. Then the concept defining framework $\langle \mathcal{S}_{\mathcal{R}}, \mathcal{E}_{\mathcal{R}}, \mathcal{S}L \rangle$ describes the grammatical system of the structure language of AB grammars, and the concept defining framework $\langle \Sigma, \mathcal{E}_{\mathcal{R}}, L \rangle$ describes the grammatical system of the string language of AB grammars.

Given a concept defining framework $\langle \mathcal{U}, \mathcal{E}, \mathcal{M} \rangle$, a grammar $G \subseteq \mathcal{E}$ is *reduced* with respect to a finite set of objects $D \subseteq \mathcal{U}$ if $D \subseteq \mathcal{M}(G)$ but $D \not\subseteq \mathcal{M}(G')$ for any grammar $G' \subset G$.

Definition 12 (Bounded finite thickness). *A concept defining framework $\langle \mathcal{U}, \mathcal{E}, \mathcal{M} \rangle$ has bounded finite thickness if \mathcal{M} is monotonic and the set*

$$\{ \mathcal{M}(G) \mid G \text{ is reduced with respect to } D \text{ and } \#G \leq n \}$$

is finite for any $D \subseteq \mathcal{U}$ and any $n \geq 0$.

Shinohara has shown that if a concept defining framework $\langle \mathcal{U}, \mathcal{E}, \mathcal{M} \rangle$ has bounded finite thickness, then for any $n \geq 0$ the class $\mathcal{L}_n = \{ \mathcal{M}(G) \mid G \subseteq \mathcal{E} \text{ and } \#G \leq n \}$ has finite elasticity (and then is learnable).

General combinatory grammars do not enjoy (in the general case) the same learnability properties as AB grammars. This point is shown for example in [9], where Costa-Florêncio gives an example of a class of rigid general combinatory grammars which is not learnable.

3.2 Flat Grammars

The sufficient condition for learnability of general combinatory grammars that we propose below lies in the restriction to *flat* grammars. Informally, this restriction is based on the distinction between *principal* and *argument* types of a complex type: each position in an operator is defined as principal or argument position, and in the latter case any type built with this operator must verify that only an atomic (or primitive) type is allowed in this position. This way the height of any type in argument position is bounded, without bounding the height (nor the size) of a type in general. We will show that this condition together with suitable restrictions on the form of the rules (see definition 14) allow that the class of languages has finite elasticity.

Definition 13 (Flat types). *Let \mathcal{S} be a set of operators, and for each operator f in \mathcal{S} let arg_f be a function from $\{1, \dots, \text{arity}(f)\}$ to $\{0, 1\}$. Given a set of variables \mathcal{V} , the set of flat \mathcal{S} -types over \mathcal{V} , denoted $FT(\mathcal{V})$, is defined as the smallest set such that*

- $\mathcal{V} \subseteq FT(\mathcal{V})$,
- for any $f \in \mathcal{S}$ with $\text{arity}(f) = n$, if t_1, \dots, t_n are flat \mathcal{S} -types over \mathcal{V} then $f(t_1, \dots, t_n) \in FT(\mathcal{V})$ if $t_i \in \mathcal{V}$ for any i such that $\text{arg}_f(i) = 1$.

A subtype u in a type t is said to be in *argument position* in t if there is a subtype w in t such that $w = f(t_1, \dots, t_i, \dots, t_n)$, $t_i = u$ and $\text{arg}_f(i) = 1$.⁵ Clearly, if t is a flat \mathcal{S} -type over \mathcal{V} then any subtype u which is in argument position in t must be in \mathcal{V} .

Definition 14 (Flat universal rule). Let $R = A_1, \dots, A_n \rightarrow A_0$ be a universal rule over a set \mathcal{S} of operators. R is a flat rule if the following conditions hold:

- for any $v \in \text{Var}(R)$, if v is a subtype of A_0 then there exists A_i with $i \geq 1$ such that v is also a subtype of A_i .
- For each i , $A_i \in FT(\text{Var}(R))$.
- For each variable $v \in \text{Var}(R)$ such that v is not in argument position in the left hand side of R (that is there is no A_i with $i \geq 1$ in which v is in argument position):
 - v is not in argument position in A_0 ,
 - and for each $i \geq 1$: $\#_v(A_i) \leq \#_v(A_0)$.

Given a set \mathcal{R} of flat universal rules, $G = \langle \Sigma, Pr, s, \triangleright \rangle$ is a flat \mathcal{R} -grammar if every type $t \in \text{Lex}(G)$ is flat.

Example 3. The argument positions of AB grammars operators are defined in the following way:

- $\text{arg}_/(1) = \text{arg}_\backslash(2) = 0$
- $\text{arg}_/(2) = \text{arg}_\backslash(1) = 1$

This means that in A/B (as well as in $B \backslash A$) B is the only argument type. One can see that AB grammars rules (defined in example 1) verify the conditions of flat universal rules: In particular there is only one variable in each rule (namely A) which does not occur in argument position in the left hand side, and the condition that A must not occur in argument position in the right hand side is fulfilled.

It is important to notice that all AB grammars types are not flat: for example, $(a/b)/c$ is flat whereas $a/(b/c)$ is not. The relationship between flat AB grammars and unrestricted AB grammars is discussed in section 4.1.

The following proposition shows that “flatness” of types is closed under derivation with flat rules. This means that it is not necessary to add a restriction to each derivation step so that it outputs only flat types, which would be counter-intuitive: if all types are flat at the beginning and only flat rules are used, then only flat types can appear.

⁵ *Remark:* in the case where there are several occurrences of u in t , it is sufficient that one occurrence verifies the condition.

Proposition 4. *Let R be a flat rule, Pr a finite set of variables and t_1, \dots, t_n flat \mathcal{S} -types over Pr . If $t_1, \dots, t_n \rightarrow_R t_0$, then t_0 is also a flat \mathcal{S} -type.*

Proof. Let $R = A_1, \dots, A_n \rightarrow A_0$. $t_1, \dots, t_n \rightarrow_R t_0$ implies that there is a substitution σ such that $\sigma(A_i) = t_i$ for all i .

Suppose $t_0 = \sigma(A_0)$ is not a flat type. Then there must be a subtype u in $\sigma(A_0)$ such that $u = f(u_1, \dots, u_m)$, and there is a k , $1 \leq k \leq m$, such that $\arg_f(k) = 1$ and $u_k \notin Pr$. For any variable $v_j \in Var(R)$ (used in any t_i) we have $\sigma(v_j) \in FT(Pr)$, because $\sigma(v_j)$ is a subtype of at least one t_i , $i \geq 1$, and $t_i \in FT(Pr)$. Therefore there is no v_j such that u is a subtype of $\sigma(v_j)$. As a consequence, A_0 contains $f(a_1, \dots, a_m)$ as a subtype, with $\sigma(a_k) \notin Pr$. But $A_0 \in FT(Var(R))$ and $\arg_f(k) = 1$, so $a_k \in Var(R)$. Since R is a flat rule and a_k is in argument position in A_0 , a_k must appear in argument position in some A_i , $i \geq 1$. Thus $\sigma(A_i)$ contains $\sigma(a_k)$ in argument position whereas $\sigma(a_k) \notin Pr$: this contradicts the hypothesis that $\sigma(A_i) = t_i$ is a flat type. \square

The following propositions are used to show that flat general combinatory grammars have bounded finite thickness. The proof, which is similar to Shinozaki's one in [8], consists in bounding the size of the possible reduced grammars.

Proposition 5. *Let R be a flat rule, Pr a finite set of variables and t_1, \dots, t_n flat \mathcal{S} -types over Pr . If $t_1, \dots, t_n \rightarrow_R t_0$, then for all $i \geq 1$*

$$\|t_i\| \leq \|t_0\| + M_R, \text{ where } M_R = \max(\{\|A_i\| \mid 1 \leq i \leq n\}) - \|A_0\|$$

Proof. Let $R = A_1, \dots, A_n \rightarrow A_0$ and $Var(R) = \{v_1, \dots, v_m\}$. $t_1, \dots, t_n \rightarrow_R t_0$ implies that there is a substitution σ such that $\sigma(A_i) = t_i$ for all i .

For any $0 \leq i \leq n$ we have

$$\|t_i\| = \|\sigma(A_i)\| = \|A_i\| + \sum_{j=1}^m (\#_{v_j}(A_i) \times (\|\sigma(v_j)\| - 1)),$$

$Var(R)$ is partitioned into two subsets \mathcal{V}_{arg} and \mathcal{V}_{pr} :

$$\mathcal{V}_{arg} = \{v \in Var(R) \mid v \text{ is in argument position in the left hand side of } R\},$$

and $\mathcal{V}_{pr} = Var(R) - \mathcal{V}_{arg}$. Let $\mathcal{V}_{arg} = \{v'_1, \dots, v'_a\}$ and $\mathcal{V}_{pr} = \{v''_1, \dots, v''_p\}$. For any $0 \leq i \leq n$

$$\|t_i\| = \|A_i\| + \sum_{j=1}^a (\#_{v'_j}(A_i) \times (\|\sigma(v'_j)\| - 1)) + \sum_{j=1}^p (\#_{v''_j}(A_i) \times (\|\sigma(v''_j)\| - 1))$$

Since $t_i = \sigma(A_i)$ is a flat type, for all j , $1 \leq j \leq a$, $\sigma(v'_j) \in Pr$. Therefore $\|\sigma(v'_j)\| = 1$, which gives

$$\sum_{j=1}^a (\#_{v'_j}(A_i) \times (\|\sigma(v'_j)\| - 1)) = 0,$$

then $\|t_i\| = \|A_i\| + \sum_{j=1}^p (\#_{v''_j}(A_i) \times (\|\sigma(v''_j)\| - 1))$

Since any $v_j'' \in \mathcal{V}_{pr}$ is not in argument position in the left hand side of R and R is a flat rule, v_j'' verifies the condition $\#_{v_j''}(A_i) \leq \#_{v_j''}(A_0)$ for all $1 \leq i \leq n$. Thus for all $i \geq 1$:

$$\begin{aligned}
 \|t_i\| &= \|A_i\| + \sum_{j=1}^p (\#_{v_j''}(A_i) \times (\|\sigma(v_j'')\| - 1)) \\
 &\leq \|A_i\| + \sum_{j=1}^p (\#_{v_j''}(A_0) \times (\|\sigma(v_j'')\| - 1)) \\
 &\leq M_R + \|A_0\| + \sum_{j=1}^p (\#_{v_j''}(A_0) \times (\|\sigma(v_j'')\| - 1)) \\
 &\leq M_R + \|t_0\|
 \end{aligned}$$

□

Proposition 6. *Let G be a flat \mathcal{R} -grammar and T an \mathcal{R} -structure. If $\langle t_1 \dots t_n, t_0 \rangle$ is an instance of T for G , then for all i , $1 \leq i \leq n$:*

$$\|t_i\| \leq h(T) \times M_{\mathcal{R}} + \|t_0\|, \text{ where } M_{\mathcal{R}} = \max(\{ M_R \mid R \in \mathcal{R} \}).$$

Proof. We show by induction on $h = h(T)$ that $t_0 \in FT(Pr)$ and $\|t_i\| \leq h \times M_{\mathcal{R}} + \|t_0\|$:

- $h = 0$. $T = w \in \Sigma$, therefore $n = 1$ and $t_0 = t_1 \in FT(Pr)$ because $w \triangleright t_1$ and G is a flat grammar.
- $h > 0$. Suppose the property holds for any $h' < h$. Let $T = [R](T_1, \dots, T_m)$, and let $\langle \alpha_1, u_1 \rangle, \dots, \langle \alpha_m, u_m \rangle$ be instances of T_1, \dots, T_m such that $t_1 \dots t_n = \alpha_1 \bullet \dots \bullet \alpha_m$ and $u_1, \dots, u_m \rightarrow_R t_0$. By induction hypothesis, the property holds for any T_i : $u_i \in FT(Pr)$ and for all $t_i \in \alpha_j$ we have $\|t_i\| \leq h_j \times M_{\mathcal{R}} + \|u_j\|$, with $h_j \leq h'$. From proposition 4 $t_0 \in FT(Pr)$, and from proposition 5 $\|u_j\| \leq \|t_0\| + M_R$, then $\|t_i\| \leq h_j \times M_{\mathcal{R}} + \|t_0\| + M_R$. Since $h_j \leq h - 1$ and $M_R \leq M_{\mathcal{R}}$, we obtain $\|t_i\| \leq (h - 1) \times M_{\mathcal{R}} + \|t_0\| + M_{\mathcal{R}}$, that is $\|t_i\| \leq h \times M_{\mathcal{R}} + \|t_0\|$. □

Corollary 1. *Let D be a finite set of \mathcal{R} -structures and G a flat \mathcal{R} -grammar. If G is reduced with respect to D then every type $t \in Lex(G)$ verifies $\|t\| \leq H_D \times M_{\mathcal{R}} + \|s\|$, where $H_D = \max(\{ h(T) \mid T \in D \})$.*

Proof. $w \triangleright_G t$ and G is reduced with respect to D , so there exists an \mathcal{R} -structure $T \in D$ and an instance $\langle \alpha, s \rangle$ of T for G such that $t \in \alpha$ (otherwise it would be possible to remove the rule $w \triangleright_G t$ from G , and G would not be reduced). By proposition 6, $\|t\| \leq h(T) \times M_{\mathcal{R}} + \|s\| \leq H_D \times M_{\mathcal{R}} + \|s\|$. □

Proposition 7. *Let \mathcal{R} be a finite set of flat rules, and $\mathcal{E}_{\mathcal{R}}$ the set of all flat \mathcal{R} -grammars lexical rules. The concept defining framework $\langle \mathcal{SL}_{\mathcal{R}}, \mathcal{E}_{\mathcal{R}}, SL \rangle$ has bounded finite thickness.*

Proof. $G \subseteq G'$ implies that $SL(G) \subseteq SL(G')$ from proposition 1. Let $G \subseteq \mathcal{E}_{\mathcal{R}}$ be a grammar reduced with respect to a finite set $D \subseteq \mathcal{SL}_{\mathcal{R}}$, with $\#G \leq n$. Corollary 1 shows that the size of each type in $Lex(G)$ is bounded by a constant

$H_D \times M_{\mathcal{R}} + \|s\|$. There must be only finitely many pairwise inequivalent such grammars, because the number of rules in G is also bounded. Since two equivalent grammars have the same language (from proposition 3), we obtain that the set $\{ SL(G) \mid G \subseteq \mathcal{E}_{\mathcal{R}} \text{ is reduced with respect to } D \text{ and } \#G \leq n \}$ is finite for any set D and any $n \geq 0$. \square

Remark: contrary to the case of simple AB grammars, where any functor-argument structure is compatible with at least one grammar, there may be no \mathcal{R} -grammar corresponding to a given \mathcal{R} -structure in $\mathcal{SL}_{\mathcal{R}}$. Thus given a set D of \mathcal{R} -structures it is possible that there is no grammar reduced with respect to D . Of course this does not contradict the previous result, since an empty set is clearly finite.

Since Shinohara has shown that bounded finite thickness implies that the class of languages definable by grammars with at most k rules has finite elasticity, the following corollaries are obtained easily with proposition 2:

Corollary 2. *Given any set \mathcal{R} of flat rules and any $k \geq 0$, the class of \mathcal{R} -structure languages definable by k -valued flat \mathcal{R} -grammars is learnable.*

Corollary 3. *Let \mathcal{R} be a set of flat rules such that any rule $R = A_1, \dots, A_n \rightarrow A_0$ in \mathcal{R} verifies $n \geq 2$. For any $k \geq 0$, the class of string languages definable by k -valued flat \mathcal{R} -grammars is learnable.*

Proof. Suppose every rule $A_1, \dots, A_n \rightarrow A_0$ in \mathcal{R} verify $n \geq 2$. We show that there is a finite-valued relation between the classes of \mathcal{R} -structure languages and of string languages definable by k -valued flat \mathcal{R} -grammars. Let G be a k -valued flat grammar. For every $w \in L(G)$ the height h of the corresponding \mathcal{R} -structure must verify $h < |w|$, because $n \geq 2$ for every rule in \mathcal{R} . So there is only a finite number of \mathcal{R} -structures corresponding to a string w . Using the property (shown by Kanazawa in [1]) that finite elasticity can be extended through a finite-valued relation, the class of string languages definable by k -valued flat grammars has also finite elasticity. \square

4 Application to Some Classes of \mathcal{R} -Grammars

In this section we propose to apply these learnability results to several formalisms, which are more or less close to the general framework of categorial grammars. Some of these positive results have already been proved separately. Even so, the fact that they can be deduced directly in our framework is interesting because it emphasizes the fact that their learnability is due to some common properties. Thus learnability of flat general combinatory grammars is also to some extent a generalization of different ways to prove learnability.

4.1 Flat Classical Categorical Grammars and Extensions

Kanazawa has explored learnability of classical categorical grammars in [1]. In particular, he has shown that k -valued AB grammars have finite elasticity, thus are learnable from strings. We have seen in example 3 that AB grammars rules verify the conditions of flat universal rules, therefore it is possible to show learnability of k -valued flat AB grammars using corollaries 2 and 3 (from structures as well as for strings).

Since all AB grammars are not flat, this result is included in the learnability result obtained by Kanazawa (in which the definition of types is the usual one, which does not require that types be flat). Nevertheless, any AB grammar can be transformed into a flat AB grammar as it is shown in [10]. In this article, the authors obtained a slightly different learnability result concerning also AB grammars, by using also flat types (called *compact types* in their article). They show that there exists a flat AB grammar for each AB grammar, thus proving that the classes of languages are equivalent.⁶

Extensions. The interest in the framework of general combinatory grammars is that it becomes possible to add specific rules to the usual AB grammars system. For example, it is interesting in the dependency grammars viewpoint (see for example [12]) to add the new operators $/^*$ (iterative types), $/^+$ (repetitive types), $/^?$ (optional types). The argument positions are defined in the same way as for AB grammars rules, that is $arg_{/*}(1) = arg_{/+}(1) = arg_{/?}(1) = 0$ and $arg_{/*}(2) = arg_{/+}(2) = arg_{/?}(2) = 1$. These operators are used with the following rules:

$$\begin{array}{lll}
 R_1^* : & A/*B \ B \rightarrow A/*B & Var(R_1^*) = \{A, B\} \\
 R_2^* : & A/*B \rightarrow A & Var(R_2^*) = \{A, B\} \\
 R_1^+ : & A/^+B \ B \rightarrow A/^+B & Var(R_1^+) = \{A, B\} \\
 R_2^+ : & A/^+B \ B \rightarrow A & Var(R_2^+) = \{A, B\} \\
 R_1^? : & A/^?B \ B \rightarrow A & Var(R_1^?) = \{A, B\} \\
 R_2^? : & A/^?B \rightarrow A & Var(R_2^?) = \{A, B\}
 \end{array}$$

Remark: Symmetrical rules have to be defined for symmetrical operators on the left: \backslash^* , \backslash^+ , $\backslash^?$.

Like AB grammars rules these rules are flat (the conditions of definition 14 are verified), therefore k -valued flat AB grammars with all these rules are also learnable from structures. However this class is not necessarily learnable from strings, because the rules R_2^* and $R_2^?$ do not fulfill the condition defined in corollary 3: there must be at least two types in the left hand side. Actually, among these operators the only one that can be added to AB grammars languages without losing learnability from strings is the repetitive operator $/^+$ (see [12] for details about this point).

⁶ Such a proof can also be achieved using the classical transformation between AB grammars and context-free grammars given in [11]: if the AB grammar is converted into a CFG grammar and re-converted into an AB grammar, then the latter is flat.

4.2 Steedman's Combinatory Categorical Grammars

Costa Florêncio has shown in [6] a sufficient condition for finite elasticity of any class of k -valued \mathcal{R} -grammars. The condition that he provides is based on the reduction of the class of languages to the class of k -valued AB grammars languages (which has finite elasticity: shown in [1]), using the fact that a finite-valued relation exists between the two classes. His criterion differs from ours: the method used implies that it must be possible to transform the set of universal rules into a set of rules which are very similar to AB grammars rules. But his criterion also has the advantage that it does not put any restriction on the form of the types, contrary to our condition (types must be flat). Actually the two methods lead to two different learnability results, each result allowing to learn classes that the other does not allow.

Costa Florêncio illustrates his learnability result with Steedman's Combinatory Categorical Grammars rules (see for example [7] about CCG): he shows that that the language generated by a subset of CCG rules is learnable. However one of the usual CCG rules, namely the composition rule, can not be included in this subset, whereas it can be used in our framework:⁷

$$\begin{array}{lll}
 >B \text{ (Forward Composition)} & A/B \ B/C \rightarrow A/C & \text{Var}(>B) = \{A, B, C\} \\
 <B \text{ (Backward Composition)} & C \setminus B \ B \setminus A \rightarrow C \setminus A & \text{Var}(<B) = \{A, B, C\} \\
 >S \text{ (Forward Substitution)} & (A/B)/C \ B/C \rightarrow A/C & \text{Var}(>S) = \{A, B, C\} \\
 <S \text{ (Backward Substitution)} & C \setminus B \ C \setminus (B \setminus A) \rightarrow C \setminus A & \text{Var}(<S) = \{A, B, C\}
 \end{array}$$

One can see that these rules verify the conditions of flat universal rules. In particular, it is worth noting that type B in the composition rule $>B$ is in argument position in the left hand side (in A/B), even if there is also an occurrence of B in B/C . Nevertheless, the fact that this rule can be used in a learnable class of grammars within our framework should not hide the fact that it is usable only with flat types.

To provide a complete view of learnability of CCG rules, a word must be said about the type raising rules:

$$\begin{array}{lll}
 >T : & A \rightarrow B/(A \setminus B) & \text{Var}(>T) = \{A, B\} \\
 <T : & A \rightarrow (B/A) \setminus B & \text{Var}(<T) = \{A, B\}
 \end{array}$$

These rules do not fulfill the conditions of “learnable rules” neither in Costa Florêncio's framework nor in ours (these rules are not flat because B does not appear in the left hand side). However types that can be used in these rules are restricted to a finite set of categories, so it is possible to “simulate” these rules directly in the lexicon [7]. This solution permits that such class of languages also have finite elasticity.

In a totally different approach, Hockenmaier has explored the acquisition of a combinatory categorical grammars lexicon in a practical viewpoint, in order to build a wide-coverage parser for English [13].

⁷ *Remark:* traditionally, types using the \setminus operator in CCG are written *Functor* \setminus *Argument*, whereas the notation *Argument* \setminus *Functor* is used in classical AB grammars. We keep this latter notation here for consistency.

4.3 Categorical Link Grammars

Link grammars are defined by Sleator and Temperley in [14]. This is a rather simple formalism which is able to represent in a reliable way natural languages. This can be seen in the modelization that the authors provided for English in this system: their grammar deals with most of the linguistic phenomena in English, as it can be verified using their link grammar parser [15].

Béchet has shown in [16] that k -valued link grammars are learnable from strings, using also Shinohara's property of bounded finite thickness. It is shown in [17] that basic link grammars rules are equivalent to the following set of \mathcal{R} -grammars rules:

$$\begin{aligned} R_l : \quad & d(L, \text{cons}(c, R)) \quad d(\text{cons}(c, \text{nil}), \text{nil}) \rightarrow d(L, R) \quad \text{Var}(R_l) = \{c, L, R\} \\ R_r : \quad & d(\text{nil}, \text{cons}(c, L)) \quad d(\text{cons}(c, L), R) \rightarrow d(L, R) \quad \text{Var}(R_r) = \{c, L, R\} \end{aligned}$$

With these rules, a sequence of words is a correct sentence for the grammar if there is a type for each word such that the sequence of types can be reduced into the special type $d(\text{nil}, \text{nil})$.

Example 4. Let define a categorical link grammar G with the following lexicon:⁸

$$\begin{aligned} \text{a, the} & \triangleright d([], [D]) \\ \text{cat, snake} & \triangleright d([D], [S]), d([0, D], []) \\ \text{chased} & \triangleright d([S], [0]) \end{aligned}$$

The following derivation shows that the sentence “the cat chased a snake” is correct for G :

$$\begin{array}{ccccccccc} & \text{the} & & \text{cat} & & \text{chased} & & \text{a} & & \text{snake} \\ & d([], [D]), d([D], [S]), d([S], [0]), d([], [D]), d([D, 0], []) & & & & & & & & \\ \Rightarrow & d([], [S]), & & d([S], [0]), d([], [D]), d([D, 0], []) & & & & & & \\ \Rightarrow & d([], [S]), & & d([S], [0]), & & d([0], []) & & & & \\ \Rightarrow & d([], [S]), & & & & d([S], []) & & & & \\ \Rightarrow & & & & & d([], []) & & & & \end{array}$$

This system is called *Categorical Link Grammars (CLG)* (see [17] for more details). Since the formalism of link grammars is (at first sight) very different from categorical grammars, this equivalence permits to include the learnability result for link grammars obtained by Béchet in the more general framework of general combinatory grammars.

Let consider that the set of operators is $\{d(2), \text{cons}(2), \text{nil}(0)\}$, with $\text{arg}_d(1) = \text{arg}_d(2) = 0$, $\text{arg}_{\text{cons}}(1) = 1$ and $\text{arg}_{\text{cons}}(2) = 0$. Clearly rules R_l and R_r are flat, because L and R , which are the only variables that are not in argument position in the left hand side, do not appear in argument position in the right hand side. Thus it is possible to apply corollaries 2 and 3 to conclude that k -valued flat categorical link grammars are learnable from structures and from strings. Since the original definition of link grammars includes only flat types, we obtain here the same result as Béchet in [16].

⁸ For a better readability, the notation $[c_1, c_2, \dots, c_n]$ for connectors lists is used here instead of $\text{cons}(c_1, \text{cons}(c_2, \dots, \text{cons}(c_n, \text{nil}) \dots))$.

5 Conclusion

In this study, we did not show that a new class of languages is learnable. But we have shown that our result includes (partially or totally) several previous learnability results. Actually, the main interest in this result is that it is focused on the way languages are expressed with a grammatical formalism: the examples show that the framework of general combinatory grammars permits to express very different formalisms through the set of universal rules, and that the condition of flat grammars is not so restrictive. In particular the example of link grammars shows that using other operators than the standard binary AB grammars operators is possible and useful.

It should also be emphasized that the criterion of flat grammars, which is a sufficient condition for learnability of k -valued grammars, is not an ad hoc “technical” condition deduced from the constraints of the learning framework: this criterion is suitable for learning from structures, and it can be easily tested with any class of \mathcal{R} -grammars. As a future work, it remains to see if this criterion can be extended to more complex types. In particular, flat types means that all types in argument positions must be atoms, and the consequence (which is the main point of the proof) is that is possible to bound the size of all types. Therefore an interesting question would be to know if it is possible to relax this constraint (for example by bounding the order of the types) without losing this consequence.

References

1. Kanazawa, M.: Learnable classes of categorial grammars. Cambridge University Press (1998)
2. Gold, E.: Language identification in the limit. *Information and control* 10 (1967) 447–474
3. Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Technical Report TR-89-05, Department of Computer Science, University of Chicago (1989)
4. Wright, K.: Identification of unions of languages drawn from an identifiable class. In: *Proceedings of the Second Annual Workshop on Computational Learning Theory*, Morgan Kaufmann (1989) 328–333
5. Motoki, T., Shinohara, T., Wright, K.: The correct definition of finite elasticity: corrigendum to Identification of unions. In: *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, San Mateo, CA, Morgan Kaufmann (1991) 375
6. Costa Florêncio, C.: Combinatory categorial grammars and finite elasticity. In: Hoste, V., Pauw, G.D., eds.: *Proceedings of the Eleventh Belgian-Dutch Conference on Machine Learning*, University of Antwerp (2001) 13–18
7. Steedman, M.: *The Syntactic Process*. The MIT Press, Cambridge, Massachusetts (2000)
8. Shinohara, T.: Inductive inference of monotonic formal systems from positive data. *New Generation Computing* 8 (1991) 371–384

9. Costa Florêncio, C.: Learning categorial grammars. PhD thesis, Utrecht University (2003)
10. Besombes, J., Marion, J.Y.: Learning reversible categorial grammars from structures. In: Proceedings of Categorial Grammars 2004, Montpellier, France. (2004) 148–163
11. Bar-Hillel, Y., Gaifman, C., Shamir, E.: On categorial and phrase structure grammars (1960)
12. Béchet, D., Dikovsky, A., Foret, A., Moreau, E.: On learning discontinuous dependencies from positive data. In: Proceedings of the 9th conference on Formal Grammar. (2004)
13. Hockenmaier, J.: Data and models for statistical parsing with Combinatory Categorial Grammar. PhD thesis, School of Informatics, The University of Edinburgh (2003)
14. Sleator, D.D.K., Temperley, D.: Parsing english with a link grammar. Technical Report CMU-CS-TR-91-126, Carnegie Mellon University, Pittsburgh, PA (1991)
15. Temperley, D., Sleator, D., Lafferty, J.: Link grammar. <http://hyper.link.cs.cmu.edu/link/> (1991)
16. Béchet, D.: k-valued link grammars are learnable from strings. In: Proceedings Formal Grammars 2003. (2003) 9–18
17. Moreau, E.: From link grammars to categorial grammars. In: Proceedings of Categorial Grammars 2004, Montpellier, France. (2004) 31–45

On Expressing Vague Quantification and Scalar Implicatures in the Logic of Partial Information

Areski Nait Abdallah¹ and Alain Lecomte^{2,3}

¹ INRIA-Rocquencourt, France

² Université de Grenoble, CLIPS-IMAG

³ Université de Bordeaux, équipe SIGNES, LaBRI-INRIA, France

Abstract. In this paper, we use the logic of partial information to re-examine some early analyses of vague quantifiers in French such as *quelques*, *peu*, *beaucoup* that are found in particular in the work of O. Ducrot [2]. Our approach is based on the paradigm offered by the logical formalization of the sorites paradox. We claim that this paradox offers a general scheme along which the argumentation structure of all vague quantifiers in French may be expressed. We offer a variational principle approximating Grice's maxims in the case of vague quantification.

Keywords: vague quantification, implicatures, argumentation scales, partial information.

1 Introduction

Most natural language quantifiers are vague. In this paper we are interested in vague quantification in the French language, although our methods may be generalized to other languages. In French, some examples of vague quantifiers are given by *beaucoup*, *peu*, *la plupart*, *quelques*, *pas tous*, etc. They all express partial information, and as such, their adequate treatment falls under the scope of a logic of partial information [4]. Such a logic should allow the correct inferences expected from linguistic usage to be drawn formally. These inferences are of more than one kind. For example, from

il a lu quelques romans de Balzac (1)

(he has read some novels of Balzac) one should be able to trivially infer:

il a lu au moins un roman de Balzac (2)

(he has read at least one novel of Balzac.) Such a consequence *literally* follows from the sentence uttered: we shall say that it is “hard” consequence of the sentence in the sense that it is impossible to question such a consequence, once the statement of the sentence from which it follows has been accepted. But one should also be able to deduce (first pointed out by [3], see also [1]):

il n'a pas lu quelques romans de Balzac (3)

(he has not read some novels of Balzac). Such a step is usually justified by calling up Grice's maxims : if it were not the case, namely if the person spoken about had read all of Balzac novels, and if the speaker had been aware of that fact, then he would not have uttered (1), but rather

il a lu tous les romans de Balzac (4)

The latter inference, however, does not have the same "firmness" as (2), as later in the discourse one may find a retraction such as

il les a même tous lus (5)

(he has even read all of them.) According to [2], and in a manner that we feel is similar, one should be able to infer:

il connaît donc un peu Balzac (6)

(he knows a bit about Balzac.) This situation differs from sentence

il n'a pas lu tous les romans de Balzac (7)

(he has not read all of Balzac novels) which, although apparently inducing that the person under consideration has read more novels than the one spoken about in (1), is preferentially oriented towards the tentative conclusion:

il ne connaît donc pas parfaitement Balzac (8)

(he does not have a perfect knowledge of Balzac.) It should be noted however, that these two conclusions are defeasible, in the sense that we very well may have fragments of discourse analogous to the following:

il a lu quelques romans de Balzac, pourtant il ne connaît pas parfaitement Balzac (9)

(he has read some of Balzac novel, still he does not have a perfect knowledge of Balzac.) or

il a lu quelques romans de Balzac, pourtant il connaît un peu Balzac (10)

(he has read some of Balzac novels, still he does have some knowledge of Balzac.) In contrast, it must be noted that it is impossible to write

**il a lu quelques romans de Balzac, pourtant (ou même, etc.) il n'en a lu aucun.* (11)

(he has read some of Balzac novels, still (or even, etc.) he has read none.) This means that the conclusions drawn in (3) and (6) do not have the same status as those drawn in (2). Such defeasible conclusions will be called *weak*. The aim of the logic of Partial Information is to provide an account of both *hard* and *weak* inference types. In that sense we think its application to the issue of scalar implicatures is legitimate.

2 Logic of Partial Information (LPI)

We briefly sketch the major ideas of the logic of partial information [4]. The logic may be seen as an extension of classical partial logic, with two truth values 0, 1 and an absence of truth value denoted by \perp . Kleene's strong tables for propositional logic connectives will be preferably used.

$\phi \wedge \psi$	1	0	\perp
1	1	0	\perp
0	0	0	0
\perp	\perp	0	\perp

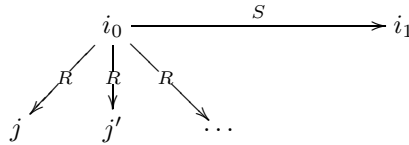
$\phi \vee \psi$	1	0	\perp
1	1	1	1
0	1	0	\perp
\perp	1	\perp	\perp

$\phi \neg \phi$	1	0	\perp
1	0	1	1
0	1	0	1
\perp	\perp	\perp	\perp

$\phi \rightarrow \psi$	1	0	\perp
1	1	0	\perp
0	1	1	1
\perp	1	\perp	\perp

This yields a fragmented notion of *truth* in a *partial model* and of *semantic scope*. If P is the set of propositional variables, f a partial truth value assignment (also called valuation) $f : P \rightarrow \{0, 1\}$, φ a formula, one defines *satisfaction* in the classical sense as $f \models \varphi$ iff $f(\varphi) = 1$, and *potential satisfaction* as $f \Vdash \varphi$ iff $f(\varphi) \neq 0$.

In order to fill the gaps of our partial knowledge by means of *tentative knowledge*, valuations are generalized to triples (i_0, J, i_1) where i_0 and i_1 are valuations, and J is a non-empty set of valuations, such that i_1 extends i_0 , every $j \in J$ extends i_0 . The basic schematic structure of ionic interpretations (i_0, J, i_1) may be summarized as follows:



Valuation i_0 corresponds to hard knowledge, i_1 corresponds to soft, tentative knowledge, and set J represents the justifications.

- i_0 is the kernel valuation (*hard knowledge*)
- $J = \{j, j', \dots\} \neq \emptyset$ are potential extensions of i_0 (*justification knowledge*)
- i_1 is the belt valuation (*soft knowledge*)

where

1. soft knowledge extends hard knowledge,
2. potential extensions extend hard knowledge: $i_0 \sqsubseteq j$ for each $j \in J$

At the syntactic level, one introduces a new partial implication $\star(n, p)$, called a *partial information ion*. In essence $\star(n, p)$ is true iff justification n is acceptable, and conclusion p is true in a “soft” sense, or justification n is not acceptable.

The reader is referred to [4] for further details. The central points of relevance to this paper have to do with the following comparison of LPI versus non-monotonic reasoning:

1. Non-monotonic reasoning theories offer a *global approach* to ordinary commonsense reasoning; they specify coherent sets of beliefs.
2. In the standard approaches to non-monotonic reasoning, justifications are swept under the rug.
3. In a theory of implicitness, justifications cannot be swept under the rug.
4. Whence the need of a *local approach* managing stepwise all the elements of the reasoning process at each point.
5. It turns out that the non-monotonic character of commonsense reasoning is a secondary surface effect following from the fact that we always reason with partial information.

3 The Logic of Partial Information and the Sand Heap Paradox

The logic of partial information offers a paradigmatic way of treating imprecision, which illustrated by the famous *sand heap paradox*, due to Eubulides of Miletus. I start with a heap of sand containing N grains of sand. I remove those grains one by one. I know that when there will be only *a few* grains left, there will be no heap left to be spoken about. What happens in-between these two extreme cases? Above all, when am I in a position to decide that there is no sand heap left? [4] p. 463 proposes to summarize the data as follows:

$$\neg p(1) \tag{12}$$

$$p(N) \tag{13}$$

$$\forall n. p(n+1) \rightarrow \star(p(n), p(n)) \tag{14}$$

where $p(n)$ stands for property : *n grains of sand constitute a heap of sand*. In this description, formula $\neg p(1)$ corresponds to situations where we are in a position to answer a definite hard *no* to the question *Do we have a sand heap?* It corresponds to the *negative boundary* of predicate p . Similarly, formula $p(N)$ corresponds to the case where we are in a position to answer a definite hard *yes* to the question *Do we have a sand heap?* It corresponds to the *positive boundary* of predicate p .

The third formula (14) expresses a *sorites axiom*, and should be understood as follows: for any n , if $n+1$ grains of sand constitute a heap of sand, then on the condition that we admit as a *justification* that n grains of sand still constitute a heap, we will be able to deduce, albeit in a *weak* manner, that n grains of sand constitute a heap. If we start with N grains, which is the value for which we acknowledge that there is indeed a “heap of sand” and if we decrease n , then we must admit more and more justifications in order to infer the presence of a heap of sand. Furthermore, such a presence will only be inferred in a weaker and

weaker form. One may assume that there is in this descent some value n_{max} , such that $p(n_{max})$ is still an admissible justification, but not $p(n_{max} - 1)$. When n ranges between n_{max} and 1, the truth value of $p(n)$ is simply *undetermined*: this is in good correspondence with our intuition.

In this case we have only one sorites axiom, with a downward induction.

4 An Application to the Case of *Quelques*

By using LPI, we get defeasability and the availability of justifications as computation objects for free. Remain to be solved the following two questions: *What justifications should one accept in order to admit the truth of a sentence such as (1)? What is the computational meaning of Ducrot's orientation of the argumentation scale?*

4.1 Generalizing the Sorites to Vague Quantifier *Quelques*

Let us start with the following formalization for a sentence containing *quelques*. Let

- $lu(k, n)$ stand for “ k a lu exactement n romans de Balzac.”
- $heap(k)$ for “ k a lu quelques romans de Balzac.”
- $luq(k, n) := (lu(k, n) \rightarrow heap(k))$ for “ k a lu quelques romans de Balzac s’il en a lu n ”

We assume that the semantic content of a sentence such as (3) is given by the following axioms:

- (Q1) $\forall k \, lu(k, 0) \rightarrow \neg heap(k)$
- (Q2) $\exists N \exists N' \, N \leq N'. \forall k \forall n. N \leq n \leq N' \rightarrow luq(k, n)$
- (q2b) $\forall k \forall n < N. luq(k, n + 1) \rightarrow \star(luq(k, n), luq(k, n))$
- (q2c) $\forall k \forall n \geq N'. luq(k, n) \rightarrow \star(luq(k, n + 1), luq(k, n + 1))$
- (Q3) The above reasons are the sole basis upon which one may claim predicate $heap(k)$ to hold.

The scope of the existential quantification $\exists N \exists N'$ is the three universal formulae in (Q2) through (q2c). These axioms will be part of the definition of *quelques*, in other words, we have a (hard) *boundary* part and a (weak) *sorites* part:

- *negative boundary*: it is not possible to claim *on a lu quelques romans* (one has read some novels) when one has not read any,
- *positive boundary*: there exists two bounds, a left bound N and a right bound N' , such that it is difficult to claim that *on a lu quelques romans* (one has read some novels), in the case one has read less than N novels. It is also difficult to claim that *on n’a lu **que** quelques romans* (one has **only** read some novels) in the case one has read more than N' novels
- *sorites*: There exist two induction “principles” allowing one to extend hard information:

- *downward induction*: it becomes increasingly difficult to admit that one has read some novels once one goes below left bound N of the positive boundary. This is expressed by downward sorites axiom ($q2b$).
- *upward induction*: it becomes increasingly difficult to admit that one has read some novels once one goes beyond right bound N' of the positive boundary. This is expressed by upward sorites axiom ($q2c$).

Clause (Q1) is not expressible in terms of predicate $lu(k, n)$, it is part of the *definition* of this predicate. This incites us to think that there is a dual definition associated with the definition of *quelques* namely the definition of *pas . . . quelques* (corresponding in the current context to the construction *il n'a pas lu quelques romans de Balzac* (he has not read some novels of Balzac). Using abbreviations,

- $nheap(k)$ for “ k n’a pas lu quelques romans de Balzac.”
- $negluq(k, n) := (\neg lu(k, n) \rightarrow nheap(k))$ for “ k n’a pas lu quelques romans de Balzac s’il n’en a pas lu n ”

We assume that the semantic content of a sentence such as (1) is given by the following axioms:

- ($\overline{Q}1$) $\forall k \, lu(k, T) \rightarrow \neg nheap(k)$
- ($\overline{Q}2$) $\exists M \exists M' M < M'$.
- ($\overline{Q}2a$) $\forall k \forall n. M \leq n \leq M' \rightarrow negluq(k, n)$
- ($\overline{Q}2b$) $\forall k \forall n < M. negluq(k, n+1) \rightarrow \star(negluq(k, n), negluq(k, n))$
- ($\overline{Q}2c$) $\forall k \forall n \geq M'. negluq(k, n) \rightarrow \star(negluq(k, n+1), negluq(k, n+1))$
- ($\overline{Q}3$) The above reasons are the sole basis upon which one may claim predicate $nheap(k)$ to hold.

where T is the total number of Balzac novels. Observe that the pair (M, M') is not necessarily identical to the pair (N, N') occurring in the first set of clauses, even though it might be.

Also observe that clauses (Q3) and ($\overline{Q}3$) express closure properties. They permit a Prolog-like negation as finite failure to be defined: it is impossible to claim *Pierre a lu quelques romans de Balzac* (resp. *Pierre n’a pas lu quelques romans de Balzac*) unless such a claim follows from clauses (Q1) through (Q3) (resp. from clauses ($\overline{Q}1$) through ($\overline{Q}3$)). In other words, “*on n’a pas lu quelques romans de Balzac*” (one has not read some of Balzac novels) may be claimed only if:

- $N \leq n \leq N'$ and one has read n such novels
- or $n < N$, $n \neq 0$ and one has some justifications for the claim
- or $n > N$, and one has some justifications for the claim.

These clauses are thus to be read as a *logic program*, and this point will be elaborated upon later. So, let us reconsider sentence (1) above *il a lu quelques romans de Balzac*. Such a sentence is valid (accepted) only if one falls into one of the three previous cases, namely:

- if $N \leq n \leq N'$, then $lu(k, n) \rightarrow heap(k)$, but $lu(k, n) \rightarrow \neg(lu(k, T - n))$. This yields three subcases:
 - $M \leq T - n \leq M'$, in which case one has $nheap(k)$
 - $T - n < M$, in which case some justifications will be needed to establish conclusion $nheap(k)$, which will be available as a *weak* conclusion.
 - $T - n > M'$, in which case some justifications will also be needed to establish weak conclusion $nheap(k)$. Since necessarily $n \neq 0$, one has $T - n \neq T$, which prevents from reaching $nheap(k)$ in the case k has read all novels.
- if $n < N$, then some justifications are needed in order to prove $heap(k)$; for $nheap(k)$, however, such a proof will depend on the position of $T - n$ with respect to M' , and so on.
- the case $n > N'$ is similar.

Therefore, in all cases, from (1) one may deduce (3) (whether in a strong sense or in a weak sense.)

It thus appears that $heap(k)$ and $nheap(k)$ are not identical, if only because the former is irreducibly false whenever $lu(k, 0)$, whereas the same holds for the latter when $lu(k, T)$. We shall assume another difference that will best show their duality : the likelihood of $heap(k)$ decreases more drastically when the number n of novels decreases on the short side of N than when n grows well beyond N' , while the opposite holds for $nheap(k)$ (this will be made more precise later on). In such a case, we shall say that $heap(k)$ is positively oriented (i.e. in the direction of ascending induction), while $nheap(k)$ is negatively oriented (i.e. in the sense of descending induction). If we now admit that “the words of the discourse” (“les mots du discours” in Ducrot’s expression) are sensitive with respect to this orientation, then we shall be able to explain why

$$il \text{ a lu quelques romans de Balzac, il les a } \mathbf{même} \text{ tous lus} \quad (15)$$

(he has read some novels of Balzac, he has **even** read all of them) is permitted, whereas

$$*il \text{ a lu quelques romans de Balzac, il en a } \mathbf{même} \text{ lu un} \quad (15)$$

(he has read some novels of Balzac, he has **even** read one of them) is not.

We suggest positing the following hypothesis:

Même fits in the preferred orientation of the predicate associated with the scalar word

and even more precisely:

Même triggers the maximal inference possible in the preferred orientation of the predicate associated with the scalar word

4.2 Evaluating Softness Degrees: Natural Deduction and Logic Programming

We are now in a position to start addressing the question asked at the beginning of this Section.

Let us assume that as a general rule, *k a lu quelques romans de Balzac s'il en lu N_1* (*k* has read some novels of Balzac if he has read N_1 such novels), expressed as $\forall k.lug(k, N_1)$. Such a rule may be given by the context of the discourse under consideration. Let us also assume that *la personne dont on parle en a effectivement lu N_2* (the person being spoken about has in fact read N_2 novels), expressed by $lu(c, N_2)$.

Let $N_1 = 5$, $N_2 = 3$. The goal is to establish the degree of softness of assertion *il a lu quelques romans*, knowing that the general rule is that *X a lu quelques romans de Balzac s'il en a lu cinq*, expressed by $\forall k. luq(k, 5)$ —and that *“la personne dont on parle en a effectivement lu trois.”* expressed by $lu(c, 3)$. The corresponding natural deduction uses these two premisses as well as the first (downward) sorites axiom. It is as follows.

$$\frac{\frac{\frac{\forall k.lug(k, 5)}{lug(c, 5)} \forall \mathcal{E} \quad \forall k \forall n.lug(k, n) \rightarrow \star(lug(k, n-1), lug(k, n-1))}{\star(lug(c, 4), lug(c, 4))}}{\star(lug(c, 4), \star(lug(c, 3), lug(c, 3)))}}{\frac{lu(c, 3) \quad \star(lug(c, 4), \star(lug(c, 3), lu(c, 3) \rightarrow heap(c)))}{\star(lug(c, 4), \star(lug(c, 3), heap(c)))} \Rightarrow C\mathcal{E}}} \exists \mathcal{I}$$

Conclusion $\exists j_1, \dots, j_s. \star(j_1, \dots, \star(j_s, \text{heap}(c)) \dots)$ corresponds to defeasible statement “Il a lu quelques romans de Balzac, sous réserve de l’acceptation des justifications j ” (he has read some novels of Balzac, subject to the acceptance of justifications j). This natural deduction calls upon hard fact $lu(c, 3)$, rule $\forall k.lug(k, 5)$ and the downward sorites axiom. The models scheme having soft conclusion $\text{heap}(c)$ includes pattern $+\star lug(c, 4), +\star lug(c, 3) \models_{soft} \text{heap}(c)$

This conclusion is reminiscent of a logic programming query. As a matter of fact, if one starts with infinitary query (see [4] p. 507) , $heap(c)^\infty$, the following logic programming derivation in LPI obtains:

$$heap(c)^\infty \rightarrow \quad (16)$$

$$(lu(c, 3) \rightarrow heap(c))^{\infty} \rightarrow \quad \text{use fact } lu(3, c) \quad (17)$$

$$(luq(c, 3))^\infty \rightarrow \text{use def. } luq(3, c) \quad (18)$$

$$(luc(c, 4))^{\infty} \rightarrow \text{use justif. } \xi_1 = luc(3, c) \quad (19)$$

$$(luq(c, 5))^\infty \rightarrow \text{use justif. } \xi_2 = luq(4, c) \quad (20)$$

$$(\quad)^\infty \rightarrow \text{use axiom } \forall k. lug(k, 5), k = c \quad (21)$$

This derivation ends with an empty query, and thus succeeds with justification answer substitution given by $\xi_1 = \text{lug}(3, c)$, $\xi_2 = \text{lug}(4, c)$. These correspond to

the justifications that must be accepted in order to derive initial goal *Il a lu quelques romans de Balzac*.

4.3 Topology of the Argumentation Scale

It seems that an important property of the framework outlined here is the subdivision of the argumentation scale into three parts: the positive and negative boundaries, corresponding to the hard information part, and the sorites induction axioms part, corresponding to the cases where only soft conclusions may be drawn.

4.3.1 Orientation of the Argumentation Scale and Approximation of Grice's Maxims

The orientation of the argumentation scale in the sense of Ducrot may be expressed in our framework by saying that those justifications that aim at obtaining weak conclusions that are closer to the negative boundary have a cost, and that they *cost more* than justifications supporting a weak conclusion closer to the positive boundary. Hard conclusions have a zero cost.

More precisely, rule stated above:

Même triggers the maximal inference possible in the preferred orientation of the predicate associated with the scalar word

may be expressed by saying the following:

1. "maximal possible inference" may be understood as "draw a maximal set of conclusions"
2. "preferred orientation" : all predicate modified by a vague quantifier are oriented towards their positive boundary.
3. in other words, one will draw, as a set of possible conclusions, all those that follow from the positive boundary, plus what follows from the application of the sorites induction axioms.
4. in **il a lu quelques romans de Balzac, il en a même lu un*, claim *il a lu un* has a maximal cost, according to the definition above.
5. therefore it must be discarded.

Technical details are as follows.

4.3.2 Cost Ordering

One may posit as a general principle that

- Every piece of knowledge produced in the reasoning process has a cost.
- The aim of the reasoning process is to minimize the cost of the conclusions obtained.

Hard conclusions have cost zero. Soft conclusions that tend to the negative boundary have a higher cost than those tending to the positive boundary i.e.

the likelihood of the corresponding conclusion decreases more drastically in the former case than in the latter.

The cost of a conclusion is (the minimum possible value of) the total cost of all justifications supporting such a conclusion. More precisely, if Γ is the current context, and φ a claim:

$$\text{cost}(\varphi) = \text{cost of the shortest justification sequence } \psi_1, \dots, \psi_n \text{ such that} \\ \Gamma \vdash \star(\psi_1, \star(\psi_2, \dots \star(\psi_n, \varphi) \dots))$$

Minimizing costs of conclusions amounts to

- spontaneously preferring hard conclusions over soft ones, and
- spontaneously preferring soft conclusions tending towards the positive boundary over soft conclusions tending towards the negative boundary.

Partial order on sets of beliefs may be only partially known.

We may even go further in formalisation, as seen in the next subsection.

4.3.3 Paths, Phase Space and Variational Principle

The notion of *path* (syntactic and semantic) is introduced in [4] chap. 19. Syntactically, a *path* is an increasing sequence of sets of partial information ionic formulae (Φ_0, Φ_1, \dots) , semantically, it corresponds to a sequence of models $(m_0, m_1, \dots, m_n, \dots)$ such that for each $i = 0, \dots, m_i \sqsubseteq m_{i+1}$ for the information ordering, $m_i \models \Phi_i$ and the least upper bound m of (m_n) is a model of the least upper bound of the syntactic path. $(\Phi_i)_{i=0, \dots}$ represent *states* in a dynamical process. It is assumed that such process behave in order to follow optimal paths in some convenient space of partial information states, called the *phase space*.

Models are in fact replaced by *model schemes*. A *model scheme* is simply a set of properties (justifications) which characterises a certain class of models.

If the process has k states, represented by sets of partial information ionic formulae $(\Phi_0, \Phi_1, \dots, \Phi_k)$ and if $M(\Phi_i)$ is the set of model schemes associated with Φ_i , $M(\Phi_0) \times M(\Phi_1) \times \dots \times M(\Phi_k)$ is the *phase space* \mathbb{P} of the process. We then define a *regular path* as a path (x_0, x_1, \dots) traced through \mathbb{P} such that $\forall i$ x_i is minimal in $M(\Phi_i)$ for the cost ordering.

We then adopt the *variational principle*: *Among all paths that it may take through \mathbb{P} , the system traces some regular path.*

The actual value of the regular path traced through phase space may or may not be known.

The variational principle is used to *eliminate* incorrect paths. This approach may be seen as providing a (partial) approximation of Grice's maxims in our framework.

The principle of preferring hard conclusions over soft ones also corresponds to the *lazy evaluation principle* of Piilog, an implementation of logic programming in LPI designed by Rajnovich [5], and amounts to some sort of *least effort principle*.

Example 1. *Il a lu quelque romans de Balzac, il en a même lu un. The background knowledge \mathbf{Q} for this problem is provided by clauses (Q1), (Q2) above. Assume $N = N' = 5$.

Let $\Phi_0 = \{\exists!n.lu(c, n), \exists\gamma. \star(\gamma, heap(c))\}$ represent the first sentence, and let $\Phi_1 = \Phi_0 \cup \{lu(c, 1)\}$ represent both sentences. We assume that $\exists!n.lu(c, n)$ is interpreted as infinite exclusive disjunction $\oplus_{n \in \mathbb{N}} lu(c, n)$.

Let $M(\Phi_0)$ be the set of models schemes of Φ_0 together with \mathbf{Q} . Set $M(\Phi_0)$ contains infinitely many justification ordering minimal model schemes m_n , where $n \in \mathbb{N}$ is the number of novels read, given as follows:

$$\begin{aligned} m_i &= \{lu(c, i), +\star luq(c, 4), \dots, +\star luq(c, i), \models_{soft} heap(c), \dots\} \text{ for } i = 1, \dots, 4. \\ m_5 &= \{lu(c, 5), \models heap(c), \dots\} \\ m_j &= \{lu(c, j), +\star luq(c, 6), \dots, +\star luq(c, j), \models_{soft} heap(c), \dots\} \text{ for } j > 5. \end{aligned}$$

Pattern $m_0 = \{lu(c, 0), +\star luq(c, 4), \dots, +\star luq(c, 1), -\star luq(c, 0), \not\models heap(c), \dots\}$ does not fit the bill, since it does not make $heap(c)$ true in a soft sense. In the cost ordering, m_1 is the maximal element in $M(\Phi_0)$.

Path (x_0, x_1) where $x_0 = m_1 = \{lu(c, 1), +\star luq(c, 4), \dots, +\star luq(c, 1), \models_{soft} heap(c), \dots\}$, and $x_1 = \{\models heap(c), lu(c, 1), \dots\}$ is the only semantically correct trajectory through \mathbb{P} corresponding to syntactic path (Φ_0, Φ_1) . However, path (x_0, x_1) is not regular, since $x_0 \in M(\Phi_0)$ is not minimal for the cost ordering on $M(\Phi_0)$. Δ

Example 2. *Il a lu quelques romans de Balzac, il n'en a même lu aucun. We now have $\Phi'_0 = \Phi_0$ as above, and $\Phi'_1 = \Phi_0 \cup \{lu(c, 0)\}$. Here, in path (x_0, x_1) , one must have $x_0 = m_n$ for some $n \neq 0$ as above, in order to make $heap(c)$ true in a soft sense. But x_1 must make $lu(c, 0)$ true, therefore cannot be an extension of such an x_0 . Therefore, since $heap(c)$ must be false at step x_1 , no trajectory of this system through the phase space makes $heap(c)$ true in a soft sense at step x_0 . In particular, no regular trajectory exists. Δ

5 Generalization to Other Vague Quantifiers in French

5.1 Outline of a General Paradigm for Scalar Implicatures

The axiomatizations above for *quelques* comprehend three components:

1. a range corresponding to situations where we must answer with a hard *yes*. It defines the positive boundary of the predicate *luq*.
2. a second range corresponding to situations where we must answer with a hard *no*. It defines the negative boundary of the predicate *luq*.
3. a third component describing a “soft” domain, corresponding to situations where the sorites axioms must be called upon.

This suggests the following general scheme for vague quantifiers. By λ -abstraction one may define fixpoint equation

$$heap = \lambda\varphi.\Phi$$

where $\Phi = \Phi(\varphi, \theta)$ is given by the set of formulae:

$$\begin{array}{l}
\textbf{Beaucoup: } 0 \xrightarrow[-]{\text{boundary}} \cdot \overset{\text{sorites}}{\vdots} K \xrightarrow[+]{\text{boundary}} \\
\textbf{Pas quelques: } 0 \overset{\text{sorites}}{\vdots} \cdot \xrightarrow[+]{\text{boundary}} \cdot \overset{\text{sorites}}{\vdots} [T \xrightarrow[-]{\text{boundary}}
\end{array}$$

Thus the respective patterns of these examples over the alphabet $\{B^+, B^-, \uparrow, \downarrow\}$ may be summarized as follows:

<i>quantifier</i>	<i>pattern</i>
sorites paradox	$B^- \downarrow B^+$
quelques	$B^- \downarrow B^+ \uparrow$
peu	$\downarrow B^+ \uparrow B^-$
un peu	$B^- \downarrow B^+ \uparrow$
beaucoup	$B^- \downarrow B^+$
pas quelques	$\downarrow B^+ \uparrow B^-$

Let us examine some differences between words like *peu*, *un peu* and *quelques*. Intuitively, *peu* and *un peu* seem quite similar, but they differ in their argumentative orientation, as shown by:

$$Il \text{ a lu peu de livres de Balzac } \textit{DONC} \text{ il ne connaît pas bien Balzac} \quad (32)$$

(he read few books of Balzac, therefore he does not know Balzac well) versus:

$$Il \text{ en a lu un peu } \textit{DONC} \text{ il connaît (un peu) Balzac} \quad (33)$$

(he read a few, therefore he knows (a little) about Balzac)

By (32), we mean that the person has read less than K books, for some rather low K . The set of formulae corresponding to *peu* is:

$$\forall k \varphi(k, T) \rightarrow \neg \textit{heap}(k) \quad (34)$$

$$\exists K \forall k \forall n. 1 \leq n \leq K \rightarrow \theta(k, n) \quad (35)$$

$$\forall k \forall n < 1. \theta(k, n+1) \rightarrow \star(\theta(k, n), \theta(k, n)) \quad (36)$$

$$\forall k \forall n \geq K. \theta(k, n) \rightarrow \star(\theta(k, n+1), \theta(k, n+1)) \quad (37)$$

This corresponds to the topological scale above: the negative boundary is $B^- = \{T\}$, where T is the total number of books, the positive boundary is therefore the opposite one, provided by small values: $B^+ = \{1, 2, \dots, K\}$. Sorites are used for values less than 1 and values greater than K . Because 0 is on the positive orientation, it will be possible to assert:

$$Il \text{ a lu peu de livres de Balzac... il n'en a même lu aucun} \quad (38)$$

Now, by (33), we mean that the person has read more than K books (K small) but less than a certain K' . The set of formulae is then the same as for *quelques*, but a comparison of the two words (as they could appear in the same discourse or uttered by the same locutor etc.), shows that K is systematically lower than the N used in the set of formulae for *quelques* in the same context.

The case of *beaucoup* is similar, using the same scheme as for *un peu* and *quelques*, but different values for N , N' . If we associate with each word w of this family its own interval of positive "hard" conclusions: $B^+ = B_w^+ = \{N_w, N_w + 1, \dots, N'_w\} = [N_w, N'_w]$, we have:

$$N_{unpeu} < N_{quelques} < N_{beaucoup}$$

$$N'_{unpeu} < N'_{quelques} < N'_{beaucoup}$$

but the scales associated with these three words have the same orientation (different from the one of *peu*, and *pas...quelques*). We can actually represent all these scales as subscales of a general one, in such a way that the following fragments of discourse are made possible:

$$Il \text{ a lu un peu de Balzac, il en a lu quelques romans} \quad (39)$$

$$Il \text{ a lu un peu de Balzac, il en a même lu beaucoup} \quad (40)$$

$$Il \text{ a lu quelques romans de Balzac, il en a même lu beaucoup} \quad (41)$$

Let us also observe that from

$$Il \text{ a lu beaucoup de romans de Balzac} \quad (42)$$

it is possible to draw the conclusion:

$$Il \text{ a lu quelques romans de Balzac} \quad (43)$$

but as a "hard" consequence, by following the same reasoning as in section 4 for proving that *il n'a pas lu quelques romans de Balzac* may be deduced from *il a lu quelques romans de Balzac*.

5.3 Further Applications

This also generalizes to scalable adjectives, etc. Statement *The Pathfinder expedition to Mars was expensive* corresponds to scheme

$$0 \xrightarrow[\text{—}]{\text{boundary}} \cdot \text{sorites} \xrightarrow[\text{+}]{\text{boundary}}$$

where B^- corresponds to (depending on the scale) "*the expedition was actually not expensive*" and B^+ corresponds to "*the expedition was actually expensive*."

This regularity suggests sharpening up as follows our approach to the formalization for vague quantifiers in French.

An *argumentation scale* is a linearly ordered set. Such a linear order may or may not have a minimal (resp. maximal) element. For the purpose of the current discussion, we assume that the ordering is discrete, i.e. that it may be embedded into \mathbb{Z} .

A general view of vague quantifiers from the argumentation point of view is as follows. Here the *definiendum* is predicate *heap*, and the *definiens* is predicate φ .

Let \mathbb{A} be an argumentation scale. Let $\varphi(k, n)$ be some given predicate.
 $\exists B^+, B^-, S^\uparrow, S^\downarrow \subseteq \mathbb{A}$ such that

1. $\{B^+, B^-, S^\uparrow, S^\downarrow\}$ is a partition of argumentation scale \mathbb{A} .
2. $\forall k \forall n \in B^+ \varphi(k, n) \rightarrow \text{heap}(k)$ (i.e. $\forall k \forall n \in B^+ \theta(k, n)$.)
3. $\forall k \forall n \in B^- \varphi(k, n) \rightarrow \neg \text{heap}(k)$
4. $\forall k \forall (n+1) \in S^\uparrow \theta(k, n) \rightarrow \star(\theta(k, n+1), \theta(k, n+1))$
5. $\forall k \forall n \in S^\downarrow \theta(k, n+1) \rightarrow \star(\theta(k, n), \theta(k, n))$

Furthermore, $B^+, B^-, S^\uparrow, S^\downarrow$ must be intervals, and thus generate a topology, in the intuitive sense.

Whence, if $\mu h.\Phi(h)$ designates the least defined predicate satisfying $\Phi(h)$, it seem that a general scheme encompassing all these case is given by definition:

$$q = \mu h. \exists B^+, B^-, S^\uparrow, S^\downarrow \subseteq \mathbb{A} . \mathbb{A} = B^+ \uplus B^- \uplus S^\uparrow \uplus S^\downarrow$$

$$\begin{aligned} & \forall k \forall n \in B^+ \theta(k, n) \\ & \forall k \forall n \in B^- \varphi(k, n) \rightarrow \neg h(k) \\ & \forall k \forall (n+1) \in S^\uparrow \theta(k, n) \rightarrow \star(\theta(k, n+1), \theta(k, n+1)) \\ & \forall k \forall n \in S^\downarrow \theta(k, n+1) \rightarrow \star(\theta(k, n), \theta(k, n)) \end{aligned}$$

where one defines locally $\theta(k, n) := (\varphi(k, n) \rightarrow h(k))$. The variations between different vague quantifiers are accounted for in this definition by using as input different partitions of argumentation scale \mathbb{A} .

6 Conclusion

We have attempted to take into account some earlier analyses, particularly by O. Ducrot [2], concerning some very sensitive discourse items which very frequently occur in our everyday language. Such phenomena are very often dealt with by calling up Grice's maxims. It is true that classical pragmatics may be efficiently applied here, but we regret that Grice's maxims have currently no adequate formalisation. This is the reason why we attempt to provide formal tools capable at least to emulate what they actually can do.

Our approach is distinct from two other ones which have been popular in the past:

- the non monotonic logical approach
- the "fuzzy sets" approach

It is distinct from the first one because, as has been said in Section 2, it is not limited to the search of consistent sets of beliefs, it allows computations step by step and thus provides a way of representing dynamic processes. It is distinct from the second one because, besides the fact that it does not use any kind of "spurious" quantification, the undetermination is not seen here as some *de re* property of the (ontological) entities themselves, but as a way of apprehending reality through *language*.

Concerning the latter point, it must be said that the true appropriate way of formulating the issues here presented is probably *dialogical*. We must think of two speakers engaged in a conversation, who try to agree on the sense of words and their applicability to a given situation. Such a conversation could be like the following:

A: *Peter read some novels by Balzac*

B: *what do you mean by "some"? five? six?*

A: *around five*

B: *actually, I know that Peter read three novels, do you still accept he read "some"?*

The appropriateness of "some" thus results from conventions that locutors state themselves, and from some kind of negotiation process according to which they finally agree whether they agree on certain justifications. This is very close to the Wittgensteinian notion of *language game* [6], and calls for a theoretical framework where word meanings would be negotiated according to this pragmatical conception of language. When we reflect on language, we perceive that not only discourse items like *quelques*, *peu*, *un peu*, *beaucoup* etc. are "vague" but ... almost every word!

References

1. G. Chierchia and S. McConnell-Ginet. *Meaning and grammar*. MIT Press, 2001.
2. O. Ducrot. *Les échelles argumentatives*. Editions de Minuit, 1980.
3. L. Horn. *On the semantic properties of logical operators in English*. PhD thesis, 1972.
4. M. A. Nait Abdallah. *The logic of partial information*. EATCS Research Monographs in Theoretical Computer Science. Springer Verlag, 1995.
5. J.J. Rajnovich. *Piilog: Partial information ionic logic programming*. PhD thesis, Department of Computer Science, University of Western Ontario, 2000.
6. L. Wittgenstein *Philosophical Investigations* edited by G.E.M. Anscombe and R. Rhees. Oxford: Basil Blackwell, 1951.

Describing Lambda Terms in Context Unification^{*}

Joachim Niehren¹ and Mateu Villaret²

¹ INRIA Futurs, Lille, France

² IMA-Universitat de Girona, Girona, Spain

Abstract. The constraint language for lambda structures (CLLS) is a description language for lambda terms. CLLS provides parallelism constraints to talk about the tree structure of lambda terms, and lambda binding constraints to specify variable binding. Parallelism constraints alone have the same expressiveness as context unification. In this paper, we show that lambda binding constraints can also be expressed in context unification when permitting tree regular constraints.

Keywords: second-order unification, dominance constraints, computational linguistics, underspecified semantics.

1 Introduction

The *constraint language for lambda structures* (CLLS) is a first-order language for describing lambda terms [5, 6]. CLLS provides parallelism constraints [7] to talk about the tree structure of lambda terms, and lambda binding constraints to specify variable binding. In particular, CLLS models the interaction of parallelism and variable binding in lambda terms correctly.

CLLS supports parallelism constraints to model ellipsis in natural language. These subsume dominance constraints [13, 23, 1, 4], an excellent modeling language for scope underspecification [15]. CLLS features lambda binding constraints in order to analyze the interactions of scope and ellipsis, i.e., parallel lambda binding. Further ingredients of CLLS are anaphoric binding constraints, group parallelism and beta reduction constraints [2].

Parallelism constraints of CLLS alone have the same expressive power as the context equations in *context unification* (CU) [14, 3]. CU is a variant of *linear second-order unification* (LSOU) [9] which restricts second-order unification to unifiers with linear lambda-terms. LSOU and CU only differ in variable binding; this difference can be captured by imposing tree regular constraints [11].

^{*} A previous version of the paper was presented at ICoS4. This research has been partially supported, on the one hand, by the Mostrare project of INRIA Futurs and the LIFL at the Universities of Lille 1 and 3, and on the other hand, by the spanish projects: TIN2004-07672-C03-01 and TIN2004-04343.

The decidability of the satisfiability problems for CU, LSOU, and CLLS – with or without tree regular constraints – are prominent open questions. Decidability of CU was often conjectured (see e.g. [12]). This is because various restrictions [3, 9, 18, 19, 20] make CU decidable, while the analogous restrictions for second-order unification don’t [10]. A decidable fragment of CLLS that can deal with most phenomena raised by scope underspecification and ellipsis was proposed recently [8]. Generally speaking, CLLS is superior in semantic modeling, while CU has advantages for studying expressiveness. This is why this paper investigates the expressiveness of CLLS by comparison to CU.

In this paper, we contribute the first comparison between CLLS and CU which accounts for lambda binding constraints. The motivating question is whether it is possible to describe the interaction of lambda binding and parallelism in CU. In other words: *can parallel lambda binding be expressed in CU similarly to CLLS?* Note that parallel lambda binding was one of the main motivations for introducing CLLS as an alternative to CU. We give a positive answer to the question but at the cost of tree regular constraints.

We show how to encode lambda binding and parallelism constraints in CU with tree regular constraints. Our encoding composes several steps, one of which exploits a recent variant of the famous relationship between monadic second-order logic (MSO) and tree automata [16, 22]. Another step relies on the non-intervenance property of lambda binding constraints, that we exhibit and prove for the first time in the present paper.

Plan. We illustrate parallel lambda binding in CLLS and argue its relevance for underspecified semantic modeling (Sec. 2). We recall tree structures, parallelism, lambda structures, and parallel lambda binding (Sec.3 and 4), the basic notions underlying CLLS, the constraint language for lambda structures (Sec. 5). We exhibit the non-intervenance property of parallel lambda binding in CLLS and prove it to hold (Sec. 6). Due to this property, we can encode parallel lambda binding by using MSO formulas (Sec. 7). Parallelism constraints with MSO formulas have the same expressiveness as CU with tree regular constraints (Sec. 8). We discuss the limitation of our approach with respect to group parallelism (Sec. 9) and conclude with some open questions.

2 Parallel Lambda Binding in Semantics

The prime application of CLLS is the modeling of underspecification in natural language semantics [5, 6]. It improves on previous approaches on analyzing that were based on higher-order unification [21], LSOU [17], and CU. Here, we illustrate why parallel lambda binding is crucial to capture the interactions of scope and ellipsis.

We consider the sentence: *John saw a taxi and so did Bill.* This elliptic sentence has two possible meanings that can be represented in higher-order logic by the following Boolean valued lambda terms:

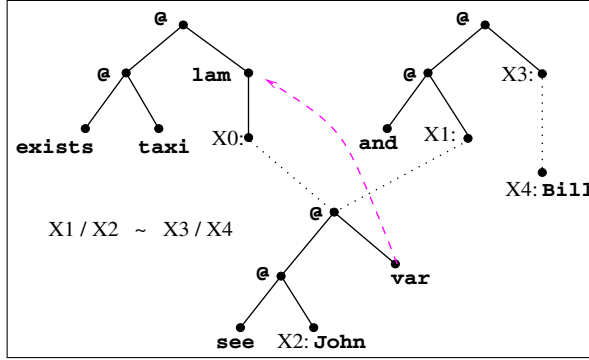


Fig. 1. The graph of a CLLS constraint

1. There exists a taxi t seen by John and Bill:

$$\text{exists taxi } \lambda t. (\text{and } (\text{see john } t) \underline{(\text{see bill } t)})$$

2. There exists a taxi t_1 seen by John and a taxi t_2 seen by Bill.

$$\text{and } (\text{exists taxi } \lambda t_1. (\text{see john } t_1)) \underline{(\text{exists taxi } \lambda t_2. (\text{see bill } t_2))}$$

The ellipses requires the meanings of source and target to be parallel except for the contrasting elements *john* and *bill*. Note that the elided parallel segments, shown underlined, are different in the two readings.

This example illustrates parallel lambda binding: in the first reading, both occurrences of t are bound by the same lambda binder taking scope over both parallel segments, while in the second case, the corresponding variables t_1 and t_2 are bound by distinct lambda binders that correspond to each other in the parallel segments. The parallel parts become equal if we rename the variables t_1 and t_2 to t . But renaming needs to be done carefully without variable capture.

The CLLS constraint in Fig. 1 can be derived by a compositional semantics construction from a parse tree. The parallelism free part describes the meanings of source sentence, the conjunction, and the target parallel element. By means of dotted lines, that express dominance between nodes, it leaves underspecified where to place the fragment with the lambda binder $\lambda((\lambda(\text{exists}, \text{taxi}), \text{lam}(X_0)))$, either above the conjunction $\lambda((\lambda(\text{and}, X_1), X_3))$ or below its first argument that starts at node X_1 .

The parallelism constraint $X_1/X_2 \sim X_3/X_4$ expresses the parallelism requirement of the ellipses. The parallel segments X_1/X_2 and X_3/X_4 must have equal tree structure and parallel binding relations. The meaning of parallel lambda binding is formalized by CLLS's lambda structures (see Section 4).

The lambda terms of both readings satisfy the constraint in Fig. 1. Binding constraints are imposed by dashed edges from **var** to **lam**-labeled nodes, the lambda binding constraints. The single dashed edge in Fig. 1 is satisfied in both

readings: in the first reading, it holds since t is bound by λt , in the second reading it holds since t_1 is bound by λt_1 and since t_2 is bound by λt_2 . No variable names are used in CLLS; this avoids variable renaming and capturing during constraint resolution once and for all.

Parallel lambda binding cannot be easily expressed in CU or LSOU where lambda binding constraints are not available. In CU for instance, the only known approach to express lambda binding is by adding variable names into function symbols lam_t and var_t . Different variables should be named apart, in order to avoid variable capture during constraint resolution. But this is not possible for variables such as t_1 and t_2 above: context equality would impose equal names.

3 Tree Structures and Parallelism

We assume a finite *signature* Σ of function symbols ranged over by f, g . Each function symbol has an arity $\text{ar}(f) \geq 0$.

Finite Trees. A finite (rooted) tree τ over Σ is a ground term over Σ , i.e. $\tau ::= f(\tau_1, \dots, \tau_n)$ where $n = \text{ar}(f) \geq 0$ and $f \in \Sigma$. We identify a node of a tree with the word of positive integers π that addresses this node from the root:

$$\text{nodes}_{f(\tau_1, \dots, \tau_n)} = \{\epsilon\} \cup \{i\pi \mid 1 \leq i \leq n, \pi \in \text{nodes}_{\tau_i}\}$$

The empty word ϵ is called the *root* of the tree, i.e. $\text{root}(\tau) = \epsilon$, while a word $i\pi$ addresses the π node of the i -th subtree of τ . We freely identify a tree τ with the function $\tau : \text{nodes}_\tau \rightarrow \Sigma$ that maps nodes to their label; for $\tau = f(\tau_1, \dots, \tau_n)$:

$$\tau(\pi) = f(\tau_1, \dots, \tau_n)(\pi) = \begin{cases} f & \text{if } \pi = \epsilon \\ \tau_i(\pi') & \text{if } \pi = i\pi' \end{cases}$$

If τ is a tree with $\pi \in \text{nodes}_\tau$ then we write $\tau.\pi$ for the subtree of τ rooted by π , and $\tau[\pi/\tau']$ for the tree obtained by replacing the subtree of τ at node π by τ' .

Dominance and Parallelism. Let τ be a tree with $\pi, \pi', \pi_1, \dots, \pi_n \in \text{nodes}_\tau$. The relation $\pi : f(\pi_1, \dots, \pi_n)$ holds for τ if node π is labeled by f in τ and has the children π_1, \dots, π_n in that order from left to right. This is if $\tau(\pi) = f$ and $\pi_1 = \pi 1, \dots, \pi_n = \pi n$ where $n = \text{ar}(f)$. The *dominance relation* $\pi \triangleleft^* \pi'$ holds for τ if π is an ancestor of π' , i.e. if π is above π' in τ , i.e. if π is a prefix of π' . *Strict dominance* $\pi \triangleleft^+ \pi'$ holds for τ if $\pi \triangleleft^* \pi'$ but not $\pi = \pi'$ in τ . The *disjointness relation* $\pi \perp \pi'$ holds for τ if neither $\pi \triangleleft^* \pi'$ nor $\pi' \triangleleft^* \pi$ in τ .

Definition 1. A segment $\sigma = \pi/\pi_1, \dots, \pi_n$ of a tree τ is a tuple of nodes π, π_1, \dots, π_n of τ such that π dominates all π_i and, all π_i with different index are pairwise disjoint. We call π the root of σ and π_1, \dots, π_n its holes. The nodes of a segment σ of a tree τ lie between the root and the holes of σ in τ :

$$\text{nodes}_\tau(\pi/\pi_1, \dots, \pi_n) = \{\pi' \in \text{nodes}_\tau \mid \pi \triangleleft^* \pi' \text{ and not } \pi_i \triangleleft^+ \pi' \text{ for any } 1 \leq i \leq n\}$$

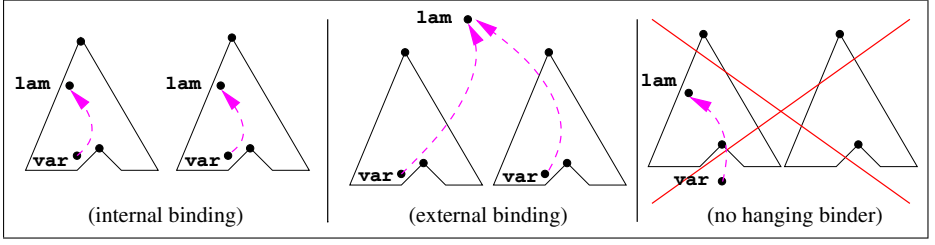


Fig. 2. Axioms of parallel lambda binding

Segment nodes generalize tree nodes in that $\text{nodes}_{\tau.\pi} = \text{nodes}_{\tau}(\pi/)$ for all trees τ and $\pi \in \text{nodes}(\tau)$. The labels of holes do not belong to segments. The *inner nodes* of a segment are those that are not holes:

$$\text{nodes}_{\tau}^{-}(\sigma) = \text{nodes}_{\tau}(\sigma) - \{\pi_1, \dots, \pi_n\} \quad \text{if } \sigma = \pi/\pi_1, \dots, \pi_n$$

Definition 2. A correspondence function between segments σ_1 and σ_2 with the same number of holes of a tree τ is a function $c : \text{nodes}_{\tau}(\sigma_1) \rightarrow \text{nodes}_{\tau}(\sigma_2)$ that is one-to-one and onto and satisfies the following homomorphism conditions:

1. The root of σ_1 is mapped to the root of σ_2 and the sequence of holes of σ_1 is mapped to the sequence of holes of σ_2 in the same order.
2. The labels of inner nodes $\pi \in \text{nodes}_{\tau}^{-}(\sigma_1)$ are preserved: $\tau(\pi) = \tau(c(\pi))$.
3. The children of inner nodes in $\pi \in \text{nodes}_{\tau}^{-}(\sigma_1)$ are mapped to corresponding children in σ_2 : for all $1 \leq i \leq \text{ar}(\tau(\pi))$ it holds that $c(\pi i) = c(\pi)i$.

We call two segments σ_1 and σ_2 of a tree structure τ (tree) *parallel* and write $\sigma_1 \sim \sigma_2$ if and only if there exists a correspondence function between them.

4 Lambda Structures and Parallel Lambda Binding

Lambda structures represent lambda terms uniquely modulo renaming of bound variables. They are tree structures extended by lambda binding edges. The signature Σ of lambda structures contains, at least, the symbols **var** (arity 0, for variables), **lam** (arity 1, for abstraction), and **@** (arity 2, for application).

The tree uses these symbols to reflect the structure of the lambda term. The binding function λ maps **var**-labeled to **lam**-labeled nodes. For example, Fig. 3 shows the lambda structure of the term $\lambda x. (f x)$ which satisfies $\lambda(12) = \epsilon$.

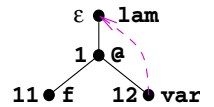


Fig. 3. $\lambda x. (f x)$

Definition 3. A *lambda structure* (τ, λ) is a pair of a tree τ and a total binding function $\lambda : \tau^{-1}(\text{var}) \rightarrow \tau^{-1}(\text{lam})$ such that $\lambda(\pi) \triangleleft^* \pi$ for all **var**-nodes π in τ .

We consider lambda structures as logical structures with the relations of tree structures, lambda binding $\lambda(\pi) = \pi'$, and its inverse relation. Inverse lambda binding $\lambda^{-1}(\pi_0) = \{\pi_1, \dots, \pi_n\}$ states that π_0 binds π_1, \dots, π_n and no other nodes.

Definition 4. Two segments σ_1, σ_2 of a lambda structure (τ, λ) are (binding) parallel $\sigma_1 \sim \sigma_2$ if they are tree parallel so that the correspondence function c between σ_1 and σ_2 satisfies the following axioms of parallel binding (see Fig. 2):

Internal binder. Internal lambda binder in parallel segments correspond: for all $\pi \in \text{nodes}_\tau^-(\sigma_1)$ if $\lambda(\pi) \in \text{nodes}_\tau^-(\sigma_1)$ then $\lambda(c(\pi)) = c(\lambda(\pi))$.

External binder. External lambda binder of corresponding var-nodes are equal: for all $\pi \in \text{nodes}_\tau^-(\sigma_1)$ if $\lambda(\pi) \notin \text{nodes}_\tau^-(\sigma_1)$ then $\lambda(c(\pi)) = \lambda(\pi)$.

No hanging binder. A var-node below a segment cannot be bound by a lam-node within: $\lambda^{-1}(\pi) \subseteq \text{nodes}_\tau^-(\sigma_i)$ for all $i \in 1, 2$ and $\pi \in \text{nodes}_\tau^-(\sigma_i)$.

Note that this definition overloads the notion of parallelism $\sigma_1 \sim \sigma_2$. For tree structures it means tree parallelism and for lambda structures binding parallelism (if not stated differently). The following basic property will be useful to prove Lemma 8 in Sec. 6.

Lemma 5. Parallelism in lambda structures is symmetric: if $\sigma_1 \sim \sigma_2$ holds in a lambda structure then $\sigma_2 \sim \sigma_1$ holds as well.

Proof. Suppose that σ_1 and σ_2 are parallel segments of a lambda structure (τ, λ) and that c is the correspondence function between them. By assumption, c satisfies the axioms of parallel binding. We have to show that the inverse correspondence function c^{-1} also satisfies these axioms.

Internal binder. Let $\pi, \lambda(\pi) \in \text{nodes}_\tau^-(\sigma_2)$ and $\pi' = c^{-1}(\pi)$ be a node in $\text{nodes}_\tau^-(\sigma_1)$. Since $\lambda(\pi')$ dominates π' there remain only two possibilities:

1. Case $\lambda(\pi') \in \text{nodes}_\tau^-(\sigma_1)$. The **internal binder** axiom for c yields $c(\lambda(\pi')) = \lambda(c(\pi')) = \lambda(\pi)$. We can apply the inverse function c^{-1} on both sides and obtain $\lambda(c^{-1}(\pi)) = c^{-1}(\lambda(\pi))$ as required.
2. Case $\lambda(\pi') \notin \text{nodes}_\tau^-(\sigma_1)$. The **external binder** axiom for c implies $\lambda(\pi') = \lambda(c(\pi')) = \lambda(\pi)$. If π' does not belong to the inner nodes of σ_2 then $\lambda(\pi')$ is a **hanging binder** which is not possible. In the same way, we can prove by induction that $(c^{-1})^n(\pi)$ must belong to the inner nodes of σ_2 for all $n \geq 1$. But this is also impossible as trees are finite.

External binder. Suppose that $\pi \in \text{nodes}_\tau^-(\sigma_2)$ while $\lambda(\pi) \notin \text{nodes}_\tau^-(\sigma_2)$. Let $\pi' = c^{-1}(\pi) \in \text{nodes}_\tau^-(\sigma_1)$. Again, there are two possibilities:

1. Case $\lambda(\pi') \in \text{nodes}_\tau^-(\sigma_1)$. The **internal binder** axiom for c yields $c(\lambda(\pi')) = \lambda(c(\pi')) = \lambda(\pi)$ which is impossible since $\lambda(\pi)$ does not belong to the image $\text{nodes}_\tau^-(\sigma_2)$ of c .
2. Case $\lambda(\pi') \notin \text{nodes}_\tau^-(\sigma_1)$. The **external binder** for c implies $\lambda(\pi') = \lambda(c(\pi')) = \lambda(\pi)$ as required.

No hanging binder. This axiom coincides for c and c^{-1} .

5 Constraint Languages

Given the model-theoretic notions of tree structures and lambda structures we can now define logical languages for their description in the usual Tarski'an manner.

We assume an infinite set X, Y, Z of *node variables* and define languages of tree descriptions in Figure 4. A *lambda binding constraint* μ is a conjunction of lambda binding and inverse lambda binding literals: $\lambda(X)=Y$ means that the value of X is a **var**-node that is lambda bound by the value of Y , while $\lambda^{-1}(X) \subseteq \{X_1, \dots, X_m\}$ says that all **var**-nodes bound by the **lam**-node denoted by X are values of one of X_1, \dots, X_m .

A *dominance constraint* is a conjunction of dominance $X \triangleleft^* Y$ and children-labeling literals $X:f(X_1, \dots, X_n)$ that describe the respective relations in some tree structure. We will write $X=Y$ to abbreviate $X \triangleleft^* Y \wedge Y \triangleleft^* X$. Note that dominance constraints are subsumed by parallelism constraints by definition. A *first-order dominance formula* ν is built from dominance constraints and the usual first-order connectives: universal quantification, negation, and conjunction. These can also express existential quantification $\exists X.\nu$ and disjunction $\nu_1 \vee \nu_2$ that we will freely use. Furthermore, we will write $X \neq Y$ instead of $\neg X=Y$ and $X \triangleleft^+ Y$ for $X \triangleleft^* Y \wedge X \neq Y$.

A *parallelism constraint* ϕ is a conjunction of children-labeling, dominance, and parallelism literals $S_1 \sim S_2$. We use *segment terms* S of the form $X/X_1, \dots, X_m$ to describe segments with m holes, given that the values of X and X_1, \dots, X_m satisfy the conditions imposed on the root and holes of segments (Definition 1). Note that a parallelism literal $S_1 \sim S_2$ requires that the values of S_1 and S_2 are indeed segments.

To keep this section self contained let us quickly recall some model theoretic notions. We write $\text{var}(\psi)$ for the set of free variables of a formula ψ of one of the above kinds. A *variable assignment* to the nodes of a tree τ is a total function $\alpha : V \rightarrow \text{nodes}(\tau)$ where V is a finite subset of node variables. A *solution* of a

Lambda binding constraints:

$$\mu ::= \lambda(X)=Y \mid \lambda^{-1}(X) \subseteq \{X_1, \dots, X_m\} \mid \mu_1 \wedge \mu_2$$

First-order dominance formulas:

$$\nu ::= X:f(X_1, \dots, X_n) \mid X \triangleleft^* Y \mid \forall X.\nu \mid \neg \nu \mid \nu_1 \wedge \nu_2$$

Parallelism constraints:

$$\phi ::= X:f(X_1, \dots, X_n) \mid X \triangleleft^* Y \mid S_1 \sim S_2 \mid \phi_1 \wedge \phi_2$$

Segment terms:

$$S ::= X/X_1, \dots, X_m \quad (m \geq 0)$$

Fig. 4. Logical languages for tree and lambda structures

formula ψ thus consists of a tree structure τ or a lambda structure (τ, λ) and a variable assignment $\alpha : V \rightarrow \text{nodes}(\tau)$ such that $\text{var}(\psi) \subseteq V$. Segment terms evaluate to tuples of nodes $\alpha(X/X_1, \dots, X_n) = \alpha(X)/\alpha(X_1), \dots, \alpha(X_n)$ which may or may not be segments. Apart from this, we require as usual that a formula evaluates to true in all solutions. We write $\tau, \alpha \models \psi$ if τ, α is a solution of ψ , and similarly $(\tau, \lambda), \alpha \models \psi$. A formula is *satisfiable* if it has a solution.

Sections 6 and 7 deal with the translation required in the following theorem.

Theorem 6. *Satisfiability of parallelism and lambda binding constraints $\phi \wedge \mu$ can be reduced in non-deterministic polynomial time to satisfiability of parallelism constraints with first-order dominance formulas $\phi' \wedge \nu$.*

Note that the signature is part of the input of the respective satisfiability problems. This means that a formula $\phi \wedge \mu$ over signature Σ can be translated to some formula $\phi' \wedge \nu$ over some other signature Σ' . Section 8 links the result of Theorem 6 to context unification plus tree regular constraints.

6 Non-intervenance Property

The idea behind our translation is to eliminate lambda bindings from the lambda-binding and parallelism constraints, by naming the variable binders. This means that we want to obtain similar parallelism constraints that use *named* labels lam_u and var_u , instead of *anonymous* labels lam and var and lambda-binding constraints.

In order to avoid undesired variable capture, we would like to associate different names to different lambda binders. But unfortunately we cannot always do so in the presence of parallelism: corresponding lam -nodes have to carry the same label lam_u and corresponding var -nodes the same label var_u .

Given that we cannot freely assign fresh names, we are faced with the danger of capturing and have to avoid it. The simplest idea would be to forbid trees where some node with label lam_u intervenes between any two other nodes with labels lam_u and var_u . This restriction can be easily expressed by a closed first-order dominance formula or could also be directly checked by a tree automaton in some tree regular constraint.

Unfortunately, the above restriction is too restrictive and thus not correct, as illustrated by the following example:

$$\text{lam}_u(@(\text{lam}_u(@(\text{lam}_u(a, \text{var}_u)), \text{var}_u)))$$

It can always happen that a corresponding lam_u takes place above of a binding lam_u -node, so that the binding lam_u intervenes between the corresponding lam_u -node and one of the var_u -nodes bound by it. Thus we need a refined *non-intervenance property* stating that no corresponding lam_u may intervene between a lam_u -node and one of the var_u -nodes it binds.

Example 7. The following parallelism constraint that is drawn in Fig. 5 is unsatisfiable:

$$X \triangleleft^+ Y \triangleleft^+ X' \wedge X/X' \sim Y/Y' \wedge Y \triangleleft^+ V \wedge \lambda(V) = X$$

This will be proved by Lemma 8. The problem is that **lam**-node Y must correspond to X but intervene between X and the **var**-node V that X binds.

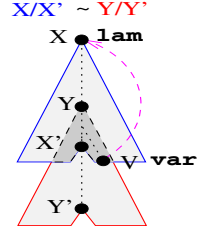


Fig. 5. Intervention

Lemma 8. *Let (τ, λ) be a lambda structure with parallel segments σ and σ' that correspond via the correspondence function c . For all π with $\lambda(\pi) \in \text{nodes}_\tau^-(\sigma)$ it is not the case that $\lambda(\pi) \triangleleft^+ c(\lambda(\pi)) \triangleleft^+ \pi$.*

Proof. We suppose that $\lambda(\pi) \in \text{nodes}_\tau^-(\sigma)$ and $\lambda(\pi) \triangleleft^+ c(\lambda(\pi)) \triangleleft^+ \pi$ and derive a contradiction. The segments σ and σ' must overlap such that the root of σ dominates that of σ' properly.

$$\text{root}(\sigma) \triangleleft^+ \text{root}(\sigma')$$

Notice that π must belong to the inner nodes of segment σ , $\pi \in \text{nodes}_\tau^-(\sigma)$, since otherwise $\lambda(\pi)$ would be a **hanging binder**.

Now suppose π does not belong to the inner nodes of the lower segment $\pi \notin \text{nodes}_\tau^-(\sigma')$. First of all, notice that π must be dominated by a hole of σ' since otherwise $\pi \perp \text{root}(\sigma')$ and the lemma would follow trivially.

We also know that $c(\lambda(\pi)) \triangleleft^+ \pi$ and belongs to the inner nodes of segments σ and σ' , therefore we can apply axiom **internal binder** and we get that $c(\lambda(\pi)) = \lambda(c(\pi))$, to avoid **hanging binder** axiom violation, $c(\pi)$ must belong to the inner nodes of segments σ and σ' , that corresponds to the next case.

Consider now the case that π also belongs to the inner nodes of the lower segment $\pi \in \text{nodes}_\tau^-(\sigma')$. We prove the following property inductively and thus derive a contradiction: For all $\pi \in \text{nodes}_\tau^-(\sigma) \cap \text{nodes}_\tau^-(\sigma')$ it is impossible that:

$$\lambda(\pi) \triangleleft^+ c(\lambda(\pi)) \triangleleft^+ \pi.$$

The proof is by well-founded induction on the length of the word π .

1. Case $\text{root}(\sigma') \triangleleft^* \lambda(\pi) \triangleleft^+ c(\lambda(\pi))$. Let $\pi' = c^{-1}(\pi)$ be an inner node of σ . The length of the word π' is properly smaller than the length of π . Since π' belongs to the inner nodes of σ , the axiom for **internal binder** can be applied to the correspondence function c yielding $c(\lambda(\pi')) = \lambda(c(\pi'))$ and thus $c(\lambda(\pi')) = \lambda(\pi)$. The node $\lambda(\pi')$ must properly dominate both $c(\lambda(\pi'))$ and π' . The address (length) of $c(\lambda(\pi'))$ is smaller than that of π' , so that:

$$\lambda(\pi') \triangleleft^+ c(\lambda(\pi')) \triangleleft^+ \pi'$$

This is impossible as stated by induction hypothesis applied to π' .

2. Case $\lambda(\pi) \triangleleft^+ \text{root}(\sigma') \triangleleft^* c(\lambda(\pi))$. Let $\pi' = c^{-1}(\pi)$ be an inner node of σ . Since π is externally bound outside of σ' , the axiom for **external binder** applies to the inverse correspondence function c^{-1} by Lemma 5 and yields $\lambda(\pi') = \lambda(\pi)$. By now, π' is internally bound in σ . The axiom for **internal binder** applied to correspondence function c yields: $c(\lambda(\pi')) = \lambda(c(\pi'))$ which is $c(\lambda(\pi)) = \lambda(\pi)$. This clearly contradicts $\lambda(\pi) \triangleleft^+ c(\lambda(\pi))$.

7 Elimination of Lambda Binding Constraints

We now give a translation that eliminates lambda binding literals while preserving satisfiability. The procedure is highly non-deterministic and introduces first-order dominance formulas to express consistent naming of bound variables.

$$\begin{aligned} \text{intervene}_{\text{lam}_u}(Y, X) &= \exists Z \exists Z'. Y \triangleleft^+ Z \triangleleft^+ X \wedge Z : \text{lam}_u(Z') \\ \text{bind}_u(X, Y) &= \exists Z (Y : \text{lam}_u(Z) \wedge Z \triangleleft^* X \wedge X : \text{var}_u) \wedge \neg \text{intervene}_{\text{lam}_u}(Y, X) \end{aligned}$$

Fig. 6. Non-intervenance and lambda binding

We impose the non-intervenance property of Lemma 8 when expressing the lambda binding predicate $\text{bind}_u(X, Y)$ in Fig. 6. This is defined by using the predicate $\text{intervene}_{\text{lam}_u}(Y, X)$, that we express via first-order dominance formulas that some lam_u -node intervenes between X and Y .

Guessing Correspondence Classes. Corresponding lam and var nodes clearly have to carry the same node labels. But we have to be a little more careful since we may have several correspondence functions for several pairs of parallel segments. We say that two nodes are in the same correspondence class for a given set of correspondence functions $\{c_1, \dots, c_n\}$ if they belong to the symmetric, reflexive, transitive closure of the common graph of these functions.

Consider for instance tree-structure τ of Fig 7, and correspondence functions c_1 and c_2 defined by $c_1(11) = 12$ and $c_2(12) = 2$. Then, $\mathcal{C}_{\tau, \{c_1, c_2\}} = \{(11, 11), (11, 12), (11, 2), (12, 11), (12, 12), (12, 2), (2, 11), (2, 12), (2, 2)\}$ is the symmetric, reflexive and transitive closure of $\{c_1, c_2\}$ in τ .

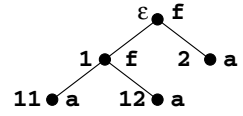


Fig. 7. $f(f(a, a), a)$

Given a parallelism and lambda binding constraint $\phi \wedge \mu$ we consider the set of correspondence functions for pairs of segments that are required to be parallel in ϕ . But how can we know whether two variables of $\phi \wedge \mu$ will denote nodes in the same correspondence class? We want to guess an equivalence relation e between variables of $\phi \wedge \mu$ depending on a solution τ, α for $\phi \wedge \mu$, such that for any two variables X and Y of $\phi \wedge \mu$, $(X, Y) \in e \iff (\alpha(X), \alpha(Y)) \in \mathcal{C}_{\tau, \{c_1, \dots, c_n\}}$, where $\{c_1, \dots, c_n\}$ are the correspondence functions for pairs of segments that are required to be parallel in ϕ . We cannot do it a priori, but we

simply guess it as there are only finitely many possibilities for the finitely many variables.

Translation. We want to guess one of the possible partitions into correspondence classes for variables of ϕ . Instead, we simply guess an equivalence relation on the variables of ϕ , and as our proofs will show, we don't have to express that equivalent variables denote values in the same correspondence class. Let

$$\text{equ}_\phi = \{e \mid e \subseteq \text{vars}(\phi) \times \text{vars}(\phi) \text{ equivalence relation}\}$$

be the set of possible equivalence relations on the variables of ϕ . We write $e(X)$ for the equivalence class of some variable $X \in \text{vars}(\phi)$ with respect to e , but consider equivalence classes of distinct equivalence relations to be distinct. Let

$$\text{names}_e = \{e(X) \mid X \in \text{vars}(\phi)\}$$

be the set names_e of e which contains all equivalence classes of e . Note that names_e is finite for all $e \in \text{equ}_\phi$, and that names_e and $\text{names}_{e'}$ are disjoint for distinct equivalence classes e and e' .

We now fix a constraint $\Phi = \phi \wedge \mu$ and guess an equivalence relation $e \in \text{equ}_\phi$ that determines the translation $[\cdot]_e$ presented in Fig. 8. This translation maps to a parallelism constraint plus first order dominance formulas $\phi' \wedge \nu$ over the following signature Σ_ϕ which extends Σ with finitely many symbols:

$$\Sigma_\phi = \Sigma \uplus \{\text{lam}_u, \text{var}_u \mid u \in \text{names}_e, e \in \text{equ}(\phi)\}$$

The literal $\lambda(X) = Y$ is translated to $\text{bind}_{e(Y)}(X, Y)$ as explained before. This ensures that all corresponding nodes in e are translated with the same name $e(Y)$. The axioms about **external binding** and **no hanging binder** are stated by first-order dominance formulas in the translation of parallelism literals. The first-order formulas are defined in Fig. 9. Note that the axiom of **internal binding** will always be satisfied without extra requirements.

We have to ensure that all var_u -nodes in solutions of translated constraints will be bound by some lam_u -node. Let no-free-var_e be as defined in Fig. 9. We then define the complete translation $[\Phi]$ by:

$$\begin{aligned} [\lambda(X)=Y]_e &= \text{bind}_{e(Y)}(X, Y) \\ [\lambda^{-1}(Y) \subseteq \{X_1, \dots, X_n\}]_e &= \forall X. \text{bind}_{e(Y)}(X, Y) \rightarrow \bigvee_{i=1}^n X = X_i \\ [Y:\text{lam}(Z)]_e &= Y:\text{lam}_{e(Y)}(Z) \\ [X:\text{var}]_e &= \exists Y. \bigvee_{\{Z \mid Z:\text{lam}(Z') \in \phi\}} \text{bind}_{e(Z)}(X, Y) \\ [Y:f(Y_1 \dots, Y_n)]_e &= Y:f(Y_1 \dots, Y_n) \quad \text{if } f \notin \{\text{lam}, \text{var}\} \\ [X \triangleleft^* Y]_e &= X \triangleleft^* Y \\ [S_1 \sim S_2]_e &= S_1 \sim S_2 \wedge \text{external-binder}_e(S_1, S_2) \wedge \\ &\quad \text{no-hang-binder}_e(S_1) \wedge \text{no-hang-binder}_e(S_2) \\ [\Phi_1 \wedge \Phi_2]_e &= [\Phi_1]_e \wedge [\Phi_2]_e \end{aligned}$$

Fig. 8. Naming variable binder for correspondence classes e

$$\begin{aligned}
\text{inside}(X, Y/Y_1, \dots, Y_n) &= Y \triangleleft^* X \wedge (\bigvee_{i \in \{1..n\}} X \triangleleft^+ Y_i) \\
\text{root}(X, Y/Y_1, \dots, Y_n) &= X=Y \\
\text{no-hang-binder}_e(S) &= \bigwedge_{u \in \text{names}_e} \text{no-hang-binder}_u(S) \\
\text{no-hang-binder}_u(S) &= \neg(\exists Y \exists Z. \text{bind}_u(Y, Z) \wedge \neg \text{inside}(Y, S) \wedge \text{inside}(Z, S)) \\
\text{external-binder}_e(S_1, S_2) &= \bigwedge_{u \in \text{names}_e} \text{external-binder}_u(S_1, S_2) \\
\text{external-binder}_u(S_1, S_2) &= \\
&\quad \forall Z_1 \forall Z_2 \forall Y. (\text{bind}_u(Z_1, Z_2) \wedge \text{inside}(Z_1, S_1) \wedge \neg \text{inside}(Z_2, S_1) \wedge \text{root}(Y, S_2)) \\
&\quad \rightarrow (Z_2 \triangleleft^* Y \wedge \neg \text{intervene}_{\text{lam}_u}(Z_2, Y)) \\
\text{no-free-var}_e &= \bigwedge_{u \in \text{names}_e} \forall X. X:\text{var}_u \rightarrow (\exists Y \exists Z. Y:\text{lam}_u(Z) \wedge Y \triangleleft^* X)
\end{aligned}$$

Fig. 9. Auxiliary predicates

$$[\Phi] = \bigvee_{e \in \text{equ}_\phi} [\Phi]_e \wedge \text{no-free-var}_e$$

We want to prove that our translation preserves satisfiability. We split the proof into the following two Lemmas:

Lemma 9. *Let Φ be a conjunction of a parallelism and lambda binding constraint and $e \in \text{equ}(\Phi)$ an equivalence relation on $\text{vars}(\Phi)$. If $[\Phi]_e \wedge \text{no-free-var}_e$ is satisfiable then Φ is satisfiable.*

Let τ be a tree structure and $\alpha : \text{vars} \rightarrow \text{nodes}_\tau$ an assignment with

$$\tau, \alpha \models [\Phi]_e \wedge \text{no-free-var}_e$$

We now define a lambda structure $(p(\tau), \lambda)$ of signature Σ by projecting labels away. The nodes of $p(\tau)$ are the nodes of τ . Let projection $\text{proj} : \Sigma_\phi \rightarrow \Sigma$ be the identity function except that $\text{proj}(\text{lam}_u) = \text{lam}$ and $\text{proj}(\text{var}_u) = \text{var}$ for any $u \in \text{names}_e$. The labels of $p(\tau)$ satisfy for all $\pi \in \text{nodes}_\tau$:

$$p(\tau)(\pi) = \text{proj}(\tau(\pi))$$

We define the lambda binding function $\lambda : p(\tau)^{-1}(\text{var}) \rightarrow p(\tau)^{-1}(\text{lam})$ as follows: Let π be a node such that $p(\tau)(\pi) = \text{var}$. There exists a unique name u such that $\tau(\pi) = \text{var}_u$. We define $\lambda(\pi)$ to be the lowest ancestor of π that is labeled by lam_u . This is the unique node in $p(\tau)$ that satisfies $\text{bind}_u(\pi, \lambda(\pi))$. It exists since we required $\tau, \alpha \models \text{no-free-var}_e$.

It remains to prove that $p(\tau), \lambda, \alpha$ is indeed a solution of Φ , i.e. whether $(p(\tau), \lambda), \alpha$ satisfies all literals of Φ .

- $X \triangleleft^* Y$ in Φ : The dominance relation of τ coincides with that of $p(\tau)$. Since $\tau, \alpha \models X \triangleleft^* Y$ it follows that $(p(\tau), \lambda), \alpha \models X \triangleleft^* Y$.
- $X:f(X_1, \dots, X_n)$ in Φ where $f \notin \{\text{lam}, \text{var}\}$. The labeling relation of τ coincides with that of $p(\tau)$, so there is no difference again.
- $X:\text{var}$ in Φ : Notice that $\text{bind}_e(Y)(X, Y)$ enforces X to be a $\text{var}_e(Y)$ -labeled node in $[\Phi]_e$, which implies $(p(\tau), \lambda), \alpha \models X:\text{var}$ by the definition of p .

- $X:\text{lam}(Z)$ in Φ : Now, the literal $X:\text{lam}_{e(X)}(Z)$ belongs to $[\Phi]_e$. Thus, $\tau, \alpha \models X:\text{lam}_{e(X)}(Z)$ which implies $(p(\tau), \lambda), \alpha \models X:\text{lam}(Z)$ by the definition of p .
- $\lambda(X)=Y$ in Φ : Let $\tau, \alpha \models [\lambda(X)=Y]_e$. By definition of the translation $[\lambda(X)=Y]_e$ this means that $\tau, \alpha \models \text{bind}_{e(Y)}(X, Y)$. In particular, it follows that $\alpha(Y)$ is the lowest $\text{lam}_{e(Y)}$ -labeled ancestor of the $\text{var}_{e(Y)}$ -labeled node $\alpha(X)$. The definition of the lambda-binding relation of $p(\tau)$ yields $(p(\tau), \lambda), \alpha \models \lambda(X)=Y$ as required.
- $\lambda^{-1}(Y) \subseteq \{X_1, \dots, X_n\}$ in Φ : the proof for this literal follows straightforward using similar arguments as for the previous one.

Consider at last, $S_1 \sim S_2$ in Φ : This is the most complicated case. If τ, α satisfies this literal then clearly, $(p(\tau), \lambda), \alpha$ satisfies the correspondence conditions for all labeling and children relations. We have to verify that $(p(\tau), \lambda)$ also satisfies the conditions of parallel binding. Let $c : \text{nodes}_{\tau}^{-}(\alpha(S_1)) \rightarrow \text{nodes}_{\tau}^{-}(\alpha(S_2))$ be the correspondence function between $\alpha(S_1)$ and $\alpha(S_2)$ which exists since $\tau, \alpha \models [\Phi]_e$.

Internal binder. Let $\lambda(\pi_1)=\pi_2$ for some $\pi_1, \pi_2 \in \text{nodes}_{\tau}^{-}(\alpha(S_1))$. By definition of λ , there exists a name u such that $\tau(\pi_1) = \text{var}_u$ and π_2 is the lowest node above π_1 with $\tau(\pi_2) = \text{lam}_u$. Since the labels of the nodes on the path between π_1 and π_2 are equal to the labels of the nodes of the corresponding path from $c(\pi_1)$ to $c(\pi_2)$ it follows that $\tau(c(\pi_1)) = \text{var}_u$, $\tau(c(\pi_2)) = \text{lam}_u$ and that no node in between is labeled with lam_u . Thus, $\lambda(c(\pi_1)) = c(\pi_2)$.

External binder. Suppose that $\lambda(\pi_1)=\pi_2$ for two nodes $\pi_1 \in \text{nodes}_{\tau}^{-}(\alpha(S_1))$ and $\pi_2 \notin \text{nodes}_{\tau}^{-}(\alpha(S_1))$. There exists a name u such that $\tau(\pi_1) = \text{var}_u$ and π_2 is the lowest ancestor of π_1 with $\tau(\pi_2) = \text{lam}_u$. By correspondence, it follows that $\tau(c(\pi_1)) = \text{var}_u$ and that no lam_u -node lies on the path from the root of segment $\alpha(S_2)$ to $c(\pi_1)$. The predicate $\text{external-binder}_u(S_1, S_2)$ requires that π_2 dominates that root of $\alpha(S_2)$ and that no lam_u -node intervenes on the path from π_2 to that root. Thus, π_2 is the lowest ancestor of $c(\pi_1)$ that satisfies $\tau(\pi_2) = \text{lam}_u$, i.e. $\lambda(c(\pi_1)) = \pi_2$.

No hanging binder. Let S be either of the segment terms S_1 or S_2 . Suppose that $\lambda(\pi_1)=\pi_2$ for some nodes $\pi_1 \notin \text{nodes}_{\tau}^{-}(S)$ and $\pi_2 \in \text{nodes}_{\tau}^{-}(S)$. There exists a name $u \in \text{names}_e$ such that $\tau(\pi_1) = \text{var}_u$ and π_2 is the lowest ancestor of π_1 with $\tau(\pi_2) = \text{lam}_u$. This contradicts that τ, α solves $\text{no-hang-binder}_u(S)$ as required by $[S_1 \sim S_2]_e$.

Lemma 10. *If Φ has a solution whose correspondence classes induce the equivalence relation e then $[\Phi]_e \wedge \text{no-free-var}_e$ is satisfiable.*

Let Φ be a conjunction of a parallelism and lambda binding constraint over signature Σ and $(\tau, \lambda), \alpha$ a solution of it. Let $\{c_1, \dots, c_n\}$ be the correspondence functions for the parallel segments $\alpha(S) \sim \alpha(S')$ where $S \sim S'$ belongs to ϕ . Let $c \subseteq \text{nodes}_{\tau} \times \text{nodes}_{\tau}$ be the reflexive, symmetric, and transitive closure of $\{c_1, \dots, c_n\}$, and $e \in \text{equ}(\Phi)$ be the relation $\{(X, Y) \mid (\alpha(X), \alpha(Y)) \in c\}$.

We define $\text{tree}_e(\tau, \lambda)$ as a tree over the extended signature Σ_ϕ whose nodes are those of τ and whose labeling function satisfies for all $\pi \in \text{nodes}_\tau$ that:

$$\text{tree}_e(\tau, \lambda)(\pi) = \begin{cases} \text{lam}_{e(X)} & \text{if } (\pi, \alpha(X)) \in c, \tau(\pi) = \text{lam}, X \in \text{vars}(\Phi) \\ \text{var}_{e(X)} & \text{if } (\lambda(\pi), \alpha(X)) \in c, X \in \text{vars}(\Phi) \\ \tau(\pi) & \text{otherwise} \end{cases}$$

We now prove that $\text{tree}_e(\tau, \lambda), \alpha$ solves $[\Phi]_e$, i.e. all of its conjuncts. This can be easily verified for dominance, labeling, and parallelism literals in $[\Phi]_e$. Notice in particular that corresponding lam-nodes in τ are assigned the same labels in $\text{tree}_e(\tau, \lambda)$. Next, we consider the first-order formulas introduced in the translation of lambda binding and parallelism literals.

1. Case $\text{bind}_{e(Y)}(X, Y)$ in $[\Phi]_e$. This requires either $\lambda(X)=Y$ or $\lambda^{-1}(Y) \subseteq \{X_1, \dots, X_n\}$ or $X:\text{var}$ in Φ . Let's consider the first case, the corresponding cases of $\lambda^{-1}(Y) \subseteq \{X_1, \dots, X_n\}$ in Φ , and of $X:\text{var}$ in Φ are quite similar. It then clearly holds that $\text{tree}_e(\tau, \lambda)(\alpha(X)) = \text{var}_{e(Y)}$ and $\text{tree}_e(\tau, \lambda)(\alpha(Y)) = \text{lam}_{e(Y)}$. Furthermore $\alpha(Y) \triangleleft^+ \alpha(X)$. It remains to show for $\text{tree}_e(\tau, \lambda)$ that no $\text{lam}_{e(Y)}$ -node intervenes between $\alpha(X)$ and $\alpha(Y)$. We do this by contradiction. Suppose there exists π such that $\alpha(Y) \triangleleft^+ \pi \triangleleft^+ \alpha(X)$ and $\text{tree}_e(\tau, \lambda)(\pi) = \text{lam}_{e(Y)}$. By definition of $\text{tree}_e(\tau, \lambda)$ there exists Z such that $(\pi, \alpha(Z)) \in c$ and $e(Y) = e(Z)$. Hence $(\alpha(Y), \alpha(Z)) \in c$ and thus $(\pi, \alpha(Y)) \in c$. But this is impossible by the non-intervenance property shown in Lemma 8: no lam-node such as π that corresponds to $\alpha(Y)$ intervene between $\alpha(Y)$ and the var-node $\alpha(X)$ bound by it.
2. Case $\text{external-binder}_u(S_1, S_2)$ in $[\Phi]_e$ where $S_1 \sim S_2$ in Φ and $u \in \text{names}_e$. By contradiction. Suppose that there exist $\pi_1 \in \text{nodes}_\tau(\alpha(S_1))$, $\pi_2 \notin \text{nodes}_\tau(\alpha(S_1))$ such that $\text{tree}_e(\tau, \lambda)(\pi_1) = \text{var}_u$ and π_2 is the lowest ancestor of π_1 with $\text{tree}_e(\tau, \lambda)(\pi_2) = \text{lam}_u$. Furthermore, assume either not $\pi_2 \triangleleft^* \text{root}(\alpha(S_2))$ or $\text{intervene}_{\text{lam}_u}(\pi_2, \text{root}(\alpha(S_2)))$. The first choice is impossible since the binding axioms were violated otherwise. (The correspondent of an externally bound node must be bound externally). Let π'_1 be the correspondent of π_1 with respect to the parallel segment $\alpha(S_1) \sim \alpha(S_2)$. By Lemma 8 we know that no lam-node corresponding to π_2 can intervene between π_2 and π'_1 and thus between π_2 and $\text{root}(S_2)$. This also contradicts the second choice: $\text{intervene}_{\text{lam}_u}(\pi_2, \text{root}(\alpha(S_2)))$.
3. $\text{no-hang-binder}_e(S)$ in $[\Phi]_e$ where S is either S_1 or S_2 and $S_1 \sim S_2$ in Φ . Let's proceed by contradiction. If it is not satisfied by $\text{tree}_e(\tau, \lambda), \alpha$, then there must exist a name $u \in \text{names}_\phi$ and two nodes π_1, π_2 such that $\text{tree}_e(\tau, \lambda)(\pi_1) = \text{lam}_u$ and $\text{tree}_e(\tau, \lambda)(\pi_2) = \text{var}_u$, even more $\pi_1 \in \text{nodes}_\tau(\alpha(S))$, $\pi_2 \notin \text{nodes}_\tau(\alpha(S))$ and there not exists a third node π_3 between π_1 and π_2 . Then, by Lemma 8, π_1 can not be a corresponding node of the lambda binding node of π_2 , therefore, by definition of $\text{tree}_e(\tau, \lambda)$ $\lambda(\pi_1) = \pi_2 \in \lambda$, but this is impossible because $(\tau, \lambda), \alpha$ must satisfy the **no hanging binder** condition.
4. Finally, we prove that $\text{tree}_e(\tau, \lambda), \alpha$ satisfies no-free-var_e . This is simple.

Proposition 11. *A parallelism and lambda binding constraint $\phi \wedge \mu$ is satisfiable if and only if its translation $[\phi \wedge \mu]$ is.*

8 Context Unification with Tree Regular Constraints

CU is the problem of solving context equations over the algebra of trees and contexts. Let the *hole marker* \bullet be a new symbol. A *context* γ over Σ is a tree over $\Sigma \cup \{\bullet\}$ such that the hole maker occurs at most once. For instance, $C = f(\bullet, a)$ is a context. The application of context C to a tree t over Σ , noted $f(\bullet, a)(t)$ is the tree $f(t, a)$ obtained from C by replacing the hole marker \bullet by t .

In CU we may have first-order variables x denoting trees over Σ and context variables C that denote contexts. The following context equations express the CLLS constraint in Fig 1 except for lambda binding:

$x = C(\text{see@john@var})$	C is the context of the verb in readings x
$C = C_1(\text{exists@taxi@lam}(C_2))$	C contains the quantifier
$C = C_3(\text{and@C}_4\text{@C}_5(\text{bill}))$	and the conjunction
$C_5 = C_2(\text{see@}\bullet\text{@var})$	bill and john do the same

We can enrich context equations by imposing tree regular constraints where \mathcal{A} is a tree-automaton over Σ .

$$x \in L(\mathcal{A})$$

Tree regular constraints can express MSO formulas over dominance constraints, even in the presence of parallelism constraints.

Theorem 12 (Theorem 11 of [16]). *Conjunctions of parallelism constraints with MSO dominance formulas have the same expressiveness as parallelism constraints with tree regular constraints.*

Finally, one can translate parallelism constraints to CU according to [14]. The proof of this paper can be easily extended to tree regular constraints:

Proposition 13 (Extension of [14]). *Parallelism with tree regular constraints have the same expressiveness as CU with tree regular constraints.*

Theorem 14. *Conjunctions of parallelism and lambda binding constraints are satisfaction equivalent to CU equations with tree regular constraints.*

This is a corollary to Theorems 6 and 12 and Proposition 13.

9 Limitations

It is proposed in [2] to extend CLLS by group parallelism in order to deal with beta reduction constraints. The question is now whether *group parallelism* can be expressed in context unification with tree regular constraints. This is a relation between groups of segments $(S_1, \dots, S_n) \sim (S'_1, \dots, S'_n)$ that behaves like a conjunction of parallelism literals $\bigwedge_{i=1}^n S_i \sim S'_i$ but such that hanging binders are defined with respect to groups of segments (S_1, \dots, S_n) resp. (S'_1, \dots, S'_n) .

Unfortunately, we cannot extend the encodings of the present paper. The problem is that group parallelism does not satisfy the non-interveneance property as stated for ordinary parallelism in Lemma 8. Indeed, it is not always possible to name variables consistently in the presence of group parallelism, so that corresponding binder of parallel groups are named alike. In other words, binding parallelism cannot be reduced to tree parallelism by naming binders. This is illustrated by the lambda structure in Fig. 10 which satisfies the group parallelism constraint:

$$(X_1/X_2, X_4/X_5) \sim (X_2/X_3, X_3/X_4)$$

Even though the lam-node X_2 corresponds to X_1 , X_2 intervenes between X_1 and its bound var-node X_6 . We thus cannot name these corresponding nodes alike.

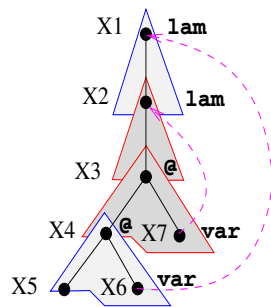


Fig. 10. Group Parallelism

10 Conclusion and Future Work

We have shown that the lambda-binding constraints of CLLS can be expressed in CU with tree regular constraints. The proof relies on the non-interveneance property of parallel lambda binding in CLLS that we establish. We leave it open whether all of CLLS can be translated into CU, in particular group parallelism, beta reduction, or anaphoric binding constraints. Another open question is how to characterize the decidable well-nested fragment of parallelism constraints [8] in a decidable fragment of CU.

References

1. R. Backofen, 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. Manuel Bodirsky, Katrin Erk, Alexander Koller, and Joachim Niehren. Underspecified beta reduction. In *ACL*, pages 74–81, 2001.
3. Hubert Comon. Completion of rewrite systems with membership constraints. *Symbolic Computation*, 25(4):397–453, 1998. Extends on a paper at ICALP’92.
4. Denys Duchier and Claire Gardent. Tree descriptions, constraints and incrementality. In *Computing Meaning*, Linguistics and Philosophy, pages 205–227. 2001.
5. Markus Egg, Alexander Koller, and Joachim Niehren. The constraint language for lambda structures. *Logic, Language, and Information*, 10:457–485, 2001.
6. Katrin Erk, Alexander Koller, and Joachim Niehren. Processing underspecified semantic representations in the constraint language for lambda structures. *Journal of Research on Language and Computation*, 1(1):127–169, 2002.
7. Katrin Erk and Joachim Niehren. Parallelism constraints. In *RTA ’00*, volume 1833 of *LNCS*, pages 110–126, 2000.
8. Katrin Erk and Joachim Niehren. Well-nested parallelism constraints for ellipsis resolution. In *EACL*, pages 115–122, 2003.

9. Jordi Levy. Linear second-order unification. In *RTA*, volume 1103 of *LNCS*, pages 332–346, 1996.
10. Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Information and Computation*, 159:125–150, 2000.
11. Jordi Levy and Mateu Villaret. Linear second-order unification and context unification with tree-regular constraints. In *RTA*, pages 156–171, 2000.
12. Jordi Levy and Mateu Villaret. Context unification and traversal equations. In *RTA'01*, volume 2051 of *LNCS*, pages 167–184, 2001.
13. Mitchell P. Marcus, Donald Hindle, and Margaret M. Fleck. D-theory: Talking about talking about trees. In *Proceedings of the 21st ACL*, pages 129–136, 1983.
14. Joachim Niehren and Alexander Koller. Dominance constraints in context unification. In *3rd LACL'98 (Grenoble, France)*, volume 2014 of *LNAI*, 2001.
15. Joachim Niehren and Stefan Thater. Bridging the gap between underspecification formalisms: Minimal recursion semantics as dominance constraints. In *41st Meeting of the Association of Computational Linguistics*, pages 367–374, July 2003.
16. Joachim Niehren and Mateu Villaret. Parallelism and tree regular constraints. In *LPAR'02*, volume 2514 of *LNAI*, pages 311–326, 2002.
17. Manfred Pinkal. Radical underspecification. In *Proceedings of the 10th Amsterdam Colloquium*, pages 587–606, 1996.
18. Manfred Schmidt-Schauß. A decision algorithm for distributive unification. *Theoretical Computer Science*, 208:111–148, 1998.
19. Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. In *CADE-16*, *LNAI*, pages 67–81, 1999.
20. Manfred Schmidt-Schauß and Klaus U. Schulz. Decidability of bounded higher-order unification. In *Computer Science Logic*, volume 2471 of *LNAI*, 2002.
21. Stuart Shieber, Fernando Pereira, and Mary Dalrymple. Interaction of scope and ellipsis. *Linguistics & Philosophy*, 19:527–552, 1996.
22. J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1967.
23. K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18:481–518, 1992.

Category Theoretical Semantics for Pregroup Grammars

Anne Preller

LIRMM/CNRS, 34392 Montpellier, France
preller@lirmm.fr

Abstract. We describe the derivations in a pregroup grammar as the 2-cells of a free compact 2-category defined by the grammar. The 2-cells of this category are the intermediary parsing structures necessary for a semantic interpretation when pregroups are used in natural language processing. The construction of the free compact 2-category also provides another cut-free axiomatisation of compact bilinear logic.

1 Introduction

This paper is an introduction to joint work with Joachim Lambek on the free compact 2-category generated by an arbitrary category \mathbf{C} .

Pregroups have been introduced in [1]. In linguistic applications, one has recourse to the *free* pregroup generated by a partially ordered set of *basic types*. A *pregroup* is a partially ordered monoid in which each element a has both a *left adjoint* a^ℓ and a *right adjoint* a^r , such that

$$a^\ell a \rightarrow 1 \rightarrow a^\ell a, \quad aa^r \rightarrow 1 \rightarrow a^r a,$$

where the arrow denotes the partial order.

For example, look at the following English phrases:

horses and roses

$$\underline{c_2} \ (\underline{n^r} \ n_2 \ \underline{n^\ell}) \ \underline{c_2} \quad \rightarrow \ n_2$$

pictures of horses

$$= \underline{c_2} \ (\underline{n_2^r} \ n_2 \ \underline{o^\ell}) \ \underline{c_2} \quad \rightarrow \ n_2$$

Mary saw (pictures of horses) and roses

(1.1)

$$\begin{aligned} & n_1 (\pi^r s_2 o^\ell) \ c_2 \ (\underline{n_2^r} \ n_2 \ \underline{o^\ell}) \ c_2 \ (\underline{n^r} \ n_2 \ \underline{n^\ell}) \ c_2 \\ = & \underline{n_1 (\pi^r s_2 o^\ell) c_2} \ (\underline{n_2^r} \ n_2 \ \underline{o^\ell}) \ c_2 \ (\underline{n^r} \ n_2 \ \underline{n^\ell}) \ c_2 \rightarrow s_2 \end{aligned}$$

Mary saw pictures of (horses and roses)

(1.2)

$$n_1 (\pi^r s_2 o^\ell) \ c_2 \ (\underline{n_2^r} \ n_2 \ \underline{o^\ell}) \ c_2 \ (\underline{n^r} \ n_2 \ \underline{n^\ell}) \ c_2$$

$$= \underbrace{n_1(\pi^r s_2 o^\ell)}_{c_2} \underbrace{(n_2^r n_2 o^\ell)}_{c_2} \underbrace{(n^r n_2 n^\ell)}_{c_2} c_2 \rightarrow s_2$$

Here we have employed the following basic types: s_2 (statement in the past tense), o (direct object), π (subject), n (noun-phrase when number does not matter), n_1 (singular noun phrase), n_2 (plural noun phrase), and c_2 (plural count noun). We also postulate

$$c_2 \rightarrow n_2 \rightarrow n, n_1 \rightarrow n, n \rightarrow o, n \rightarrow \pi$$

to determine the partial order among basic types, so that e.g.

$$c_2 n^r \rightarrow n n^r \rightarrow 1, o^\ell n_2 \rightarrow o^\ell o \rightarrow 1, n_1 \pi^r \rightarrow \pi \pi^r \rightarrow 1.$$

Note that we have assigned to each English word a *type*, namely a string of *simple types* of the form

$$\dots a^{\ell\ell}, a^\ell, a, a^r, a^{rr} \dots$$

where a is any basic type. In the example, *pictures*, *horses*, *roses*, *Mary* have been assigned basic types whereas

$$\text{saw: } \pi^r s_2 o^\ell, \text{ and: } n^r n_2 n^\ell, \text{ of: } n_2^r n_2 o^\ell.$$

The two derivations of

$$n_1(\pi^r s_2 o^\ell) c_2 (n^r n o^\ell) c_2 (n^r n_2 n^\ell) c_2 \rightarrow s_2$$

have different meanings, and must be represented differently in a discourse structure. This suggests that we must be able to distinguish between different derivations and therefore should take the arrow to denote not just derivability, but the actual derivation. Therefore the two distinct derivations above might be thought of as morphisms in a certain category, where the categorical structure introduces equality on derivations. As we shall see, the derivations are 2-cells in a compact 2-category with one 0-cell, freely generated by a given partially ordered set or, more generally, category \mathbf{C} .

We construct a compact 2-category $T(\mathbf{C})$, freely generated by \mathbf{C} . The 2-cells of $T(\mathbf{C})$, the so called *transitions*, are labelled graphs and generalise derivations in pregroup grammars. They can also be described as labelled transition systems [2]. Horizontal composition models parallelism, vertical composition models temporal composition of these systems. Our transition systems are given in normal form, i.e. they have initial and final, but no intermediary states. Otherwise said, the 2-cells can be generated without vertical composition. The fact that every 2-cell is equal to a 2-cell in normal form is the categorical version of what logicians call “cut-elimination”. The normal form corresponds to a cut-free axiomatisation of compact bilinear logic different from that given in [3]. Our proof also provides a linear algorithm for computing composition in $T(\mathbf{C})$.

2 Compact 2-Categories Recalled

The concept of a 2-category generalises the notion of the category of natural transformations $t : F \rightarrow G$ between functors $F : \mathbf{M} \rightarrow \mathbf{N}$, $G : \mathbf{M} \rightarrow \mathbf{N}$. Here the categories \mathbf{M} and \mathbf{N} are the 0-cells, F and G the 1-cells and t is a 2-cell. The usual definition of natural transformations requires the commutativity of the following diagram, where $f : A \rightarrow B$ is a given arrow in the category \mathbf{M} ,

$$\begin{array}{ccc} FA & \xrightarrow{Ff} & FB \\ tA \downarrow & \text{\textit{tf}} \swarrow & \downarrow tB \\ GA & \xrightarrow{Gf} & GB \end{array}$$

that is the equality

$$tB \circ Ff = Gf \circ tA = tf, \text{ for } f : A \rightarrow B, t : G \rightarrow F, \quad (2.1)$$

where \circ denotes the composition of 2-cells. It is reasonable to denote the diagonal by tf .

Now this equality is valid in any 2-category where A , B , F and G are arbitrary 1-cells, and FA , tA , tf denotes *horizontal composition*.

This horizontal composition is to be distinguished from the *vertical composition*

$$s \circ t : F \xrightarrow{t} G \xrightarrow{s} H,$$

the usual composition of 2-cells. The two compositions are related by the equality

$$(s \circ t)(g \circ f) = sg \circ tf, \quad (2.2)$$

Mac Lane's so-called *interchange law* [4].

If we identify B with 1_B and F with 1_F , we see that (2.1) is a special case of (2.2). But (2.2) can also be deduced from (2.1) and the *distributive laws*

$$(s \circ t)C = sC \circ tC, \quad F(g \circ f) = Fg \circ Ff. \quad (2.3)$$

As a consequence of (2.1), note that

$$1_{FA} = 1_F 1_A = 1_F A \circ F 1_A = F 1_A \circ 1_F A. \quad (2.4)$$

A 2-category is said to be *compact*, if every 1-cell has both a left and a right adjoint. A 1-cell G is said to be the *right adjoint* $G = F^r$ of 1-cell F or F the *left adjoint* $F = G^\ell$ of G , if there are 2-cells $\varepsilon_G : G^\ell G \rightarrow 1$ and $\eta_G : 1 \rightarrow GG^\ell$ such that

$$G\varepsilon_G \circ \eta_G G = 1_G, \quad \varepsilon_G G^\ell \circ G^\ell \eta_G = 1_{G^\ell} \quad (2.5)$$

or, identifying 1_G with G ,

$$G\varepsilon_G \circ \eta_G G = G, \quad \varepsilon_G G^\ell \circ G^\ell \eta_G = G^\ell.$$

As in linguistic applications, call ε_G a *contraction* and η_G an *expansion*.

All the usual properties of adjoints, familiar from the category of (small) categories remain valid in an arbitrary 2-category. For example, adjoints are unique up to isomorphism (see e.g. [4]). This implies in particular that

$$G^{\ell r} \simeq G \simeq G^{r\ell}. \quad (2.6)$$

Note that if H has a left adjoint H^ℓ with universal 2-cells ϵ_G and η_G , then GH has a left adjoint $H^\ell G^\ell$ with universal 2-cells

$$\epsilon_{GH} = \epsilon_H \circ H^\ell \epsilon_G H, \quad \eta_{GH} = G \eta_H G^\ell \circ \eta_G. \quad (2.7)$$

In particular, it follows that

$$(GH)^\ell \simeq H^\ell G^\ell \quad \text{and} \quad (GH)^r \simeq H^r G^r. \quad (2.8)$$

For any 2-cell $f : F \rightarrow G$, one can define 2-cells $f^\ell : G^\ell \rightarrow F^\ell$ and $f^r : G^r \rightarrow F^r$ by

$$f^\ell = \epsilon_G F^\ell \circ G^\ell f F^\ell \circ G^\ell \eta_F. \quad (2.9)$$

$$f^r = F^r \epsilon_{G^r} \circ F^r f G^r \circ \eta_{F^r} G^r \quad (2.10)$$

We note that $f F^\ell \circ \eta_F = G f^\ell \circ \eta_G$ and $\epsilon_G \circ G^\ell f = \epsilon_F \circ f^\ell F$.

Hence we introduce the *generalised contraction* ϵ_f as a common name:

$$\epsilon_f = \epsilon_G \circ G^\ell f = \epsilon_F \circ f^\ell F : G^\ell F \rightarrow 1. \quad (2.11)$$

Similarly, we may define a *generalised expansion*

$$\eta_f = f F^\ell \circ \eta_F = G f^\ell \circ \eta_G : 1 \rightarrow G F^\ell \quad (2.12)$$

It follows that

$$f^{r\ell} = f = f^{\ell r} \quad (2.13)$$

$$(g \circ f)^\ell = f^\ell \circ g^\ell, \quad (g \circ f)^r = f^r \circ g^r. \quad (2.14)$$

In the following we concentrate on the so called a *strictly monoidal categories*, i.e. 2-categories with only one 0-cell. As there is only one 0-cell, horizontal composition is defined for arbitrary 1-cells and in view of (2.1), we may assume that the same holds for arbitrary 2-cells.

3 Transitions

For a given category \mathbf{C} , we introduce a category $T(\mathbf{C})$ in which the 2-cells are labelled graphs, called *transitions*, and show that it is the compact strictly monoidal category freely generated by \mathbf{C} .

As \mathbf{C} is to be embedded in the free category, the objects A, B, \dots of \mathbf{C} , are identified with 1-cells, and the arrows of \mathbf{C} with 2-cells such that composition in \mathbf{C} becomes vertical composition in $T(\mathbf{C})$. Let

$$\dots, A^{(-2)}, A^{(-1)}, A^{(0)}, A^{(1)}, A^{(2)}, \dots$$

stand for

$$\dots, A^{\iota}, A^{\ell}, A, A^r, A^{rr}, \dots$$

The 1-cells of $T(\mathbf{C})$ are strings

$$\Gamma = A_1^{(z_1)} \dots A_n^{(z_n)}, \quad z_i \in \mathbb{Z}, \quad A_i \in |\mathbf{C}|,$$

where the empty string represents the unit 1. Following pregroup terminology, 1-cells of the form $A^{(z)}$ are called *simple types* and strings of simple types are called *types*. Using letters A, B for simple types, we refer to the integer z such that $A = A^{(z)}$ as the *marker* of A and to A as the *base* of A .

Right and left adjoints are defined in the meta-language

$$\begin{aligned} (A_1^{(z_1)} \dots A_n^{(z_n)})^{\ell} &= A_n^{(z_n-1)} \dots A_1^{(z_1-1)}, \\ (A_1^{(z_1)} \dots A_n^{(z_n)})^r &= A_n^{(z_n+1)} \dots A_1^{(z_1+1)}. \end{aligned}$$

In particular

$$(A^{(z)})^{\ell} = A^{(z-1)}, \quad (A^{(z)})^r = A^{(z+1)}.$$

It is customary in pregroup grammars to represent contractions of simple types as under-links:

$$\varepsilon_A : A^{\ell} A \rightarrow 1 \quad \underbrace{A^{\ell} A}_A \quad \text{or} \quad \underbrace{A^{\ell} A}_A.$$

By analogy, we introduce over-links for expansions of simple types:

$$\eta_A : A A^{\ell} \rightarrow 1 \quad \overbrace{A A^{\ell}}^A \quad \text{or} \quad \overbrace{A A^{\ell}}^A.$$

Representing an arrow $s : A \rightarrow B$ of \mathbf{C} as a vertical link

$$\begin{array}{c} A \\ \downarrow s \\ B \end{array},$$

we generalize this to

$$\begin{array}{ccccccccc} A^{(-2)} & B^{(-1)} & A^{(0)} & B^{(1)} & A^{(2)} & & A^{(z)} & A^{(z)} \\ \downarrow s^{(-2)} & \downarrow s^{(-1)} & \downarrow s^{(0)} & \downarrow s^{(1)} & \downarrow s^{(2)} & \text{with} & \downarrow & \text{for} & \downarrow 1_A^{(z)} \\ B^{(-2)} & A^{(-1)} & B^{(0)} & A^{(1)} & B^{(2)} & & A^{(z)} & A^{(z)}. \end{array}$$

Again, $\dots, s^{(-2)}, s^{(-1)}, s^{(0)}, s^{(1)}, s^{(2)}, \dots$ stands for $\dots, s^{\iota}, s^{\ell}, s, s^r, s^{rr}, \dots$. It is convenient to think of $s^{(z)}$ as a copy of s and declare $s^{(z)} : A^{(z)} \rightarrow B^{(z)}$ if either

$s : A \rightarrow B$ and z even or $s : B \rightarrow A$ and z odd. We use $s : A \rightarrow B$ for $s^{(z)} : A^{(z)} \rightarrow B^{(z)}$ and refer to it as a *simple arrow*. We call the integer such that $s = s^{(z)}$ the *marker* and the arrow s the base of s . If $s = s^{(z)} : A \rightarrow B$, $t = t^{(z)} : B \rightarrow C$ we define

$$\begin{aligned} t \circ s &= (t \circ s)^{(z)}, \text{ if } z \text{ is even} \\ &= (s \circ t)^{(z)}, \text{ if } z \text{ is odd.} \end{aligned}$$

Another convenient notation concerning simple arrows is

$$\begin{aligned} s^\ell &= (s^{(z)})^\ell = s^{(z-1)} \\ s^r &= (s^{(z)})^r = s^{(z+1)}. \end{aligned}$$

It follows from these definitions that $(t \circ s)^\ell = s^\ell \circ t^\ell$ and $(t \circ s)^r = s^r \circ t^r$.

The idea is to extend this graphical representation of contractions, expansions and simple arrows to all 2-cells of the free category, using *links* labelled by simple arrows. Horizontal composition is represented by the disjoint union of sets of links. For example,

$$\begin{array}{c} A \\ | \\ \overbrace{A \ A^\ell} \\ | \\ A \end{array} \text{ stands for } \eta_A A : A \rightarrow AA^\ell A, \quad \text{and} \quad \begin{array}{c} A \ A^\ell \ A \\ \overbrace{\hspace{1cm}} \\ | \\ A \end{array} \text{ for } A\epsilon_A : AA^\ell A \rightarrow A.$$

The vertical composition must then satisfy $A\epsilon_A \circ \eta_A A = A$, i.e. we must identify

$$\begin{array}{c} A \\ | \\ \overbrace{A \ A^\ell \ A} \\ | \\ A \end{array} \quad \text{and} \quad \begin{array}{c} A \\ | \\ A \end{array}$$

This is completely general: the composite path identifies with the link through its endpoints. For $s : B \rightarrow A$, we represent the generalised contraction

$$\epsilon_s = \epsilon_A \circ A^\ell s = \epsilon_B \circ s^\ell A : A^\ell B \rightarrow 1 \text{ by } \overbrace{A^\ell \ B}_s$$

and then must define vertical composition such that

$$\begin{array}{c} A^\ell \ B \\ | \quad | \\ \overbrace{A^\ell \ A} \end{array} \quad = \quad \begin{array}{c} A^\ell \ B \\ s^\ell \quad | \\ \overbrace{B^\ell \ B} \end{array} \quad = \quad \overbrace{A^\ell \ B}_s.$$

Similarly, we represent the generalised expansion

$$\eta_s = sB^\ell \circ \eta_B = As^\ell \circ \eta_A : 1 \rightarrow AB^\ell \text{ by } \overbrace{A \quad B}^s.$$

Definition 1

A *transition* $g : A_1 \dots A_m \rightarrow B_1 \dots B_n$ over \mathbf{C} is a set of labelled links $\{i, k\}$

$$\begin{array}{ccc} A_1 \dots A_i \dots A_m & & A_1 \dots A_i \dots A_k \dots A_m, \\ \downarrow s_{ik} & & \underbrace{\hspace{1.5cm}}_{s_{ik}} \\ B_1 \dots B_k \dots B_n & & B_1 \dots B_i \dots B_k \dots B_m. \end{array}$$

where i and k are distinct¹ positions in the domain $A_1 \dots A_m$ or the codomain $B_1 \dots B_n$. The following conditions must hold:

A) The labels of the links have the form $s_{ik} = s^{(z)}$, $z \in \mathbb{Z}$ and s in \mathbf{C} , such that

1) If $\{i, k\}$ is *vertical*, i.e. i in the domain and k in the codomain, then

$A_i = A^{(z)}$ and $B_k = B^{(z)}$ and either $s : A \rightarrow B$ if z is even, or $s : B \rightarrow A$ if z is odd.

(This says that s points downward, i.e. from the domain to the codomain, if z is even, and upward if z is odd).

2) If $\{i, k\}$ is an *under-link*, i.e. i and k are in the domain, and if $i < k$,

then $A_i = B^{(z-1)}$ and $A_k = A^{(z)}$ and either $s : A \rightarrow B$ if z is even, or $s : B \rightarrow A$ if z is odd.

(This means that in under-links, s has its tail at the base with the even marker and its head at the base with the odd marker).

3) If $\{i, k\}$ is an *over-link*, i.e. i and k are in the codomain, and if $i < k$,

then $B_i = B^{(z)}$ and $B_k = A^{(z-1)}$ and either $s : A \rightarrow B$ if z is odd, or $s : B \rightarrow A$ if z is even.

(Hence in over-links, s is directed from the base with the odd marker to the base with the even marker).

B) Every position i in $A_1 \dots A_m$ or $B_1 \dots B_n$ is endpoint of exactly one link.

C) Links may not cross, i.e.

(i) If $\{i, k\}$ and $\{j, l\}$ are vertical links and $i < j$, then $k < l$.

(ii) If $\{i, k\}$ is an under-link, and $i < j < k$, then j belongs to an under-link $\{j, l\}$ such that $i < l < k$.

(iii) idem for over-links

¹ Hence the integer i denoting the position of A_i in the domain is to be considered different from the integer i denoting the position of B_i in the codomain!

$$\frac{A}{B} \Big|_s$$
$$\begin{array}{ccccccc}
 A & B^{\ell}B & C^{\ell} & \underbrace{C^{\ell\ell}B^{\ell}}_{s^{\ell}} & \underbrace{A^{\ell}BC}_{r} \\
 | & & & & \\
 t & & & & \\
 | & & & & \\
 D & \underbrace{B \quad C^{\ell}}_s & & &
 \end{array}$$
$$t\eta_s \varepsilon_B = t\varepsilon_B \eta_s$$

$$\begin{array}{c} A \xrightarrow{B^\ell B} \\ \downarrow t \\ D \xrightarrow[B \quad C^\ell]{} \end{array}$$

$$\boldsymbol{\varepsilon}_{s^\ell} \boldsymbol{\varepsilon}_{t^\ell} = \underbrace{C^{\ell\ell} B^\ell}_{s^\ell} \underbrace{A^{\ell\ell} D^\ell}_{t^\ell} C$$

$$\left[C^\ell \quad \underbrace{C^{\ell\ell} B^\ell}_{s^\ell} \quad \underbrace{A^{\ell\ell} D^\ell}_{t^\ell} C \right].$$

is obtained from g by adding a new under-link from B^ℓ to A labelled s .

For $s : A \rightarrow B$, $h : 1 \rightarrow \Delta$

$$\eta_s(h) : 1 \rightarrow B \Delta A^\ell$$

is obtained from h by adding a new over-link from B to A^ℓ labelled s .

For example,

$$\begin{aligned} \epsilon_C(\epsilon_s, \epsilon_t) &= \frac{C^\ell \quad \frac{C^{\ell\ell} B^\ell}{s^\ell} \quad \frac{A^{\ell\ell} D^\ell}{t^\ell} C}{\frac{s}{B \quad C^\ell}}, \\ \eta_s(1) &= \frac{s}{B \quad C^\ell} \quad \text{and} \quad \epsilon_t = \epsilon_t(1) = \frac{D^\ell}{t} A. \end{aligned}$$

The context permitting, we continue to write η_s for $\eta_s(1)$, ϵ_t for $\epsilon_t(1)$ etc.

Horizontal composition and the operators ϵ_s and η_s are sufficient to generate all transitions from the one-link transitions. This can be captured by the following construction rules:

Definition 3 (Normal form of transitions)

A transition in *normal form* is defined inductively as follows

$$\begin{aligned} \text{(Unit)} & \quad \frac{}{1 : 1 \rightarrow 1 \text{ normal}} \\ \text{(Horizontal composition)} & \quad \frac{f : \Gamma \rightarrow \Delta \text{ normal} \quad g : \Lambda \rightarrow \Theta \text{ normal}}{fg : \Gamma \Lambda \rightarrow \Delta \Theta \text{ normal}} \\ \text{(Simple)} & \quad \frac{}{s^{(z)} : A^{(z)} \rightarrow B^{(z)} \text{ normal}} \\ \text{(Contraction)} & \quad \frac{f : \Gamma \rightarrow 1 \text{ normal}}{\epsilon_{s^{(z)}}(f) : A^{(z-1)} \Gamma B^{(z)} \rightarrow 1 \text{ normal}} \\ \text{(Expansion)} & \quad \frac{f : 1 \rightarrow \Delta \text{ normal}}{\eta_{s^{(z)}}(f) : 1 \rightarrow B^{(z-1)} \Delta A^{(z)} \text{ normal}} \end{aligned}$$

where (Simple), (Contraction) and (Expansion) are restricted to either $s : A \rightarrow B$ and z even or $s : B \rightarrow A$ and z odd.

Lemma 1 (Normal form)

Every transition $f : \Gamma \rightarrow \Delta$ can be generated from 1 and the simple transitions with the help of horizontal composition and the operators ϵ_s and η_s .

Lemma 2 (Uniqueness)

If all 1 's are cancelled in the horizontal composition and parentheses are grouped to the right, the horizontal normal form is unique.

The proof is straight forward as the example below may illustrate.

$$\begin{array}{c}
A \quad B^\ell B \quad C^\ell \quad \underbrace{C^{\ell\ell} B^\ell}_{s^\ell} \quad \underbrace{A^{\ell\ell} D^\ell}_{t^\ell} \quad C \\
\downarrow t \\
D \quad B \quad C^\ell.
\end{array}
\quad = \quad t \varepsilon_B(1) \eta_s(1) \varepsilon_C(\varepsilon_{s^\ell} \varepsilon_{t^\ell}).$$

This representation of a transition in normal form does not involve vertical composition. We show below that the set $T(\mathbf{C})$ of transitions over \mathbf{C} is closed under (the yet to be defined) vertical composition. It will follow that $T(\mathbf{C})$ is the free compact strictly monoidal category generated by \mathbf{C} .

There is an obvious candidate for *vertical composition*, as we see by “connecting” transitions vertically:

$$\begin{array}{c} A \\ s \downarrow \\ B \\ t \downarrow \\ C \end{array}$$
 should become

$$\begin{array}{c} A \\ t \circ s \downarrow \\ C \end{array}$$

and connecting



$$\begin{array}{cccc} & \text{---} & & \\ & | & & \\ A^\ell & & A^\ell & A^{\ell\ell} \\ & | & & \\ A & & A & \end{array}$$

$g =$

with
 $f =$

$$\begin{array}{cc} \begin{array}{c} A^\ell \\ \text{---} \\ A \end{array} & \begin{array}{c} A \\ \downarrow \\ A^\ell \end{array} \\ & \begin{array}{c} A^\ell \\ \downarrow \\ A^{\ell\ell} \end{array} \end{array}$$

the connected graph

$g; f =$

 should become
 $g \circ f =$


Definition 4 (Vertical Composition)

Let $f : \Gamma \rightarrow \Delta$ and $g : \Delta \rightarrow \Lambda$ be transitions, denote $g;f$ the graph obtained by connecting f with g at Δ . Then $g \circ f$ is the graph obtained from $g;f$ by replacing each composite path with no proper extension by a single link through its endpoints, labelled by the composition of the labels.

In the examples above, the labels of the links are identities, among them the identity of $A^{\ell\ell}$ and the identity of A^{ℓ} . Hence it is not obvious that successive labels can always be composed. However, this will follow from the following lemma, which also establishes that transitions are closed under vertical composition.

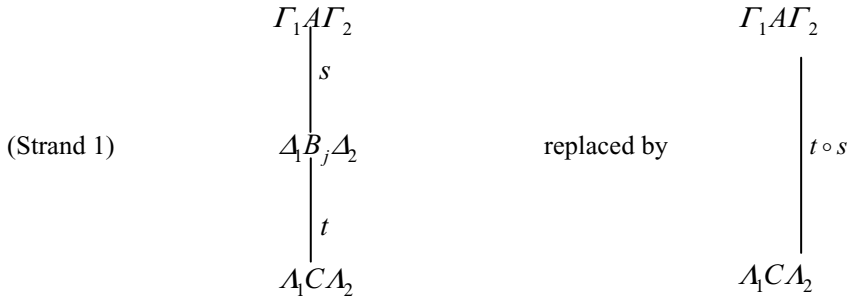
Lemma 3 (Combing)

Let $f : \Gamma \rightarrow \Delta$ and $g : \Delta \rightarrow A$ be transitions, then $g \circ f$ is a transition of domain Γ and codomain A .

Proof: Use induction on the length m of the *intermediary* string Δ . If $m=0$, then Δ is empty, f has only under-links, g only over-links. Hence all paths in $g;f$ have length 1 and $g \circ f = g;f = gf$. For the induction step, assume that Δ is non-empty and that the property holds for all transitions connected at an intermediary Δ' shorter than Δ . Note that every path of length at least 2 goes through a position in Δ . In the following argument, we choose a section of a path through such a position consisting of two or three consecutive links. This section will be called a *strand* and be replaced by a single link, with the same endpoints. There are eight different strands to be considered:

Case 1:

Suppose Δ has a position j such that f has a vertical link $s : A \rightarrow B_j$ and g a vertical link $t : B_j \rightarrow C$.

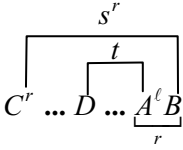


Case 2 (Strand 2)-(Strand 7)

If Δ does not have such a position, assume first that g has at least one under-link. Then there is a position j in Δ such that $\{j, j+1\}$ is an under-link of g . Let $r : B \rightarrow A$ be the label of this under-link, hence $B_j = A^\ell$ and $B_{j+1} = B$. Let Δ' be obtained from Δ by omitting $B_j B_{j+1}$ and g' from g by omitting the link $\{j, j+1\}$. Clearly, g' is a transition from Δ' to A .

Next, consider the links $\{i, j\}$ and $\{j+1, k\}$ of f . Note that two consecutive positions $j, j+1$ in Δ cannot simultaneously form an over-link of f and an under-link of g . Indeed, the former would imply that the marker of B_j is greater than the marker of B_{j+1} , whereas the latter would imply the contrary. Hence, i and k are both different from j and $j+1$. We obtain f' from f by omitting the two links $\{i, j\}$ and $\{j+1, k\}$ and adding the new link $\{i, k\}$. For each strand, we verify that the labels of $\{i, j\}$, $\{j, j+1\}$ and $\{j+1, k\}$ can be composed, providing thus the label for $\{i, k\}$. Then the maximal paths² of $g;f$ identify with the maximal paths of

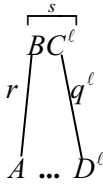
² i.e. paths which have no proper extension.

(Strand 7)  replaced by $\overbrace{(t \circ r \circ s)}^r C^r \dots D \dots$

Case 3:

Remains the case where g has no under-links. As we are in the case where no position in Δ belongs both to a vertical link in g and to vertical link in f , the latter must have over-links. Hence there is a position j such that $\{j, j+1\}$ is an over-link of f , i.e. if

$$q : D \rightarrow C, s : C \rightarrow B, r : B \rightarrow A$$

(Strand 8)  is replaced by $\overbrace{A \dots D^l}^{r \circ s \circ q}$.

Note that the vertical composition of two transitions can be computed in time proportional to the number of links in the transitions. Indeed, it suffices to follow a maximal composite path exactly once, computing on the way the composite of the successive labels.

Corollary

$T(\mathbf{C})$ is a compact strictly monoidal category.

Proof: Vertical composition is clearly associative, the identity $1_\Gamma : \Gamma \rightarrow \Gamma$ consists of the obvious vertical links through corresponding simple types. The label of the link connecting i in the domain to i in the codomain is the identity of the simple type A_i . If the context permits, we write Γ instead of 1_Γ . Then the equality (2.1)

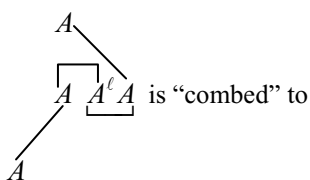
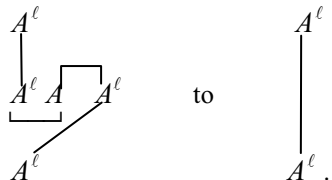
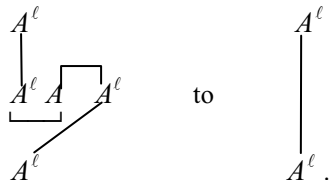
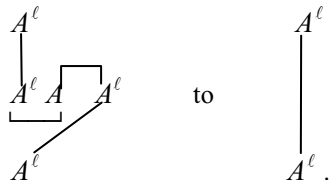
$$g\Lambda \circ \Delta f = \Theta f \circ g\Gamma = gf, \text{ for } f : \Gamma \rightarrow \Lambda, g : \Delta \rightarrow \Theta$$

is straightforward.

Compactness follows, if

$$A\varepsilon_A \circ \eta_A A = A \text{ and } \varepsilon_A A^\ell \circ A^\ell \eta_A = A^\ell$$

holds. By (2.7), it is enough to verify this for all simple types A :

 is “combed” to  and  to .

Proposition

$T(\mathbf{C})$ is the free strictly monoidal compact category generated by \mathbf{C} .

The proof is beyond the scope of this paper and can be found in [6].

The Combing Lemma is the categorical version of cut-elimination in compact bilinear logic, established in [3]. Indeed, the categorical equality defines an equivalence relation on proofs such that transitions are cut-free representatives of equivalence classes. Besides providing a graphical representation of cut-free proofs, the categorical result tells us more: not only can we derive from $f : \Gamma \rightarrow \Delta$ and $g : \Delta \rightarrow \Lambda$ the existence of a $h : \Gamma \rightarrow \Lambda$, but also show that the new $h : \Gamma \rightarrow \Lambda$ is equivalent to $g;f$. The axiomatisation of compact bilinear logic corresponding the normal form of transitions is the following

(Unit)

$$\frac{}{1 \vdash 1}$$

(Horizontal composition)

$$\frac{\Gamma \vdash \Delta \quad \Lambda \vdash \Theta}{\Gamma\Lambda \vdash \Delta\Theta}$$

(Simple)

$$\frac{}{A^{(z)} \vdash B^{(z)}}$$

(Contraction)

$$\frac{\Gamma \vdash 1}{A^{(z-1)} \Gamma B^{(z)} \vdash 1}$$

(Expansion)

$$\frac{1 \vdash \Delta}{1 \vdash B^{(z-1)} \Delta A^{(z)}}$$

where (Simple), (Contraction) and (Expansion) are restricted to the cases where either $A \rightarrow B$ and z even or $B \rightarrow A$ and z odd.

The rule corresponding to vertical composition is

$$(Cut) \quad \frac{\Gamma \vdash \Delta \quad \Delta \vdash \Lambda}{\Gamma \vdash \Lambda}.$$

Clearly, the inference rules of the systems S respectively S' for compact bilinear logic in [3] can be derived with the rules above and vice versa.

4 Example of a Semantic Interpretation

We sketch an interpretation of examples (1.1) and (1.2) into predicate logic without explicitly defining the underlying algorithm. The idea behind the algorithm is to process simultaneously syntactic and semantic content associated to words. The algorithm will use semantical as well as syntactical types. Composition of transitions realizes the passage from syntactic analysis to meaning. The category of semantical transitions is generated from a multi-graph where two nodes may have more than one edge between them in opposition to the partially ordered set of basic syntactical types.

For example, the semantical basic type e of *entities*³ may have the different arrows $p : e \rightarrow e$, $h : e \rightarrow e$ etc. where p is short for *pictures*, h for *horses* etc. That is to say, the words of the dictionary are used as labels. The algorithm computes the composition of the semantical transition and the trace of the syntactical transition and transforms the result into a formula of predicate logic. We assume equality, functional symbols and two sorts of constants or variables x, y, \dots and X, Y, \dots . The latter are to be interpreted as sets of individuals of the universe of discourse, the former as singleton sets, i.e. as individuals. Proper names and count nouns are interpreted by unary relational symbols, like *mary*, *picture*, *horse*, *rose* for *Mary*, *picture*, *horse*, *rose*. A word with a compound type including the sentence type is interpreted by a relational symbol. If the basic type reduces to π or o , the whole type corresponds to a functional symbol or constant, where the number of non-basic simple types in the compound type gives the arity. Functional symbols are to be interpreted as partially defined functions. For example

saw: $\pi^r s, o^\ell, e^r se^\ell$, *saw*(,); *and*: $n^r n, n^\ell, e^r ee^\ell$, *and*(,); *of*: $n_r^r n, o^\ell, e^r ee^\ell$, *of*(,).

The syntactical analysis

$$\begin{array}{c}
 \text{Mary saw (pictures of horses) and roses} \\
 = \underbrace{n_1(\boldsymbol{\pi}^r \ s_2 \ o^\ell)}_{\text{NP}} \underbrace{c_2}_{\text{VP}} \underbrace{(n_2^r \ n_2 \ o^\ell)}_{\text{NP}} \underbrace{c_2}_{\text{VP}} \underbrace{(n^r \ n_2 \ n^\ell)}_{\text{NP}} \underbrace{c_2}_{\text{VP}}
 \end{array}$$

Fig. 1

yields a semantical *trace*

$$\begin{array}{ccccccc} e(e^r s e^\ell) & e(e^r e e^\ell) & e(e^r e e^\ell) & e(e^r e e^\ell) \\ \underbrace{\hspace{1.5cm}} & \underbrace{\hspace{1.5cm}} & \underbrace{\hspace{1.5cm}} & \underbrace{\hspace{1.5cm}} \\ \underbrace{\hspace{3.5cm}} & & & \\ s & & & \end{array}$$

Fig. 2

which is connected to the semantical *declaration*

The labels of the semantical declaration are found reading the words from left to right. For the basic type in the word's type, declare a constant and a new variable except in the types of the connectors where the variable is omitted.⁴

³ In a more sophisticated example, it may be necessary to use more than one semantic type for entities.

⁴ Declaring the semantical type associated to the connector with a “hard-wired” link $\frac{of}{\overline{of}}$ without a variable has the same effect as replacing $\exists u(u=f(x) \wedge \phi(u))$ by $\phi(f(x))$ in first order logic. I would like to thank Claudia Casadio for telling me about her insight that words like relative pronouns have types with over-links which propagate equality of individuals.

The logical structure of the expression is computed by “combing”, i.e. following the paths from endpoint to endpoint, starting with the left upper simple type. To find the corresponding logical expression, it suffices to read the labels in the order they are composed.

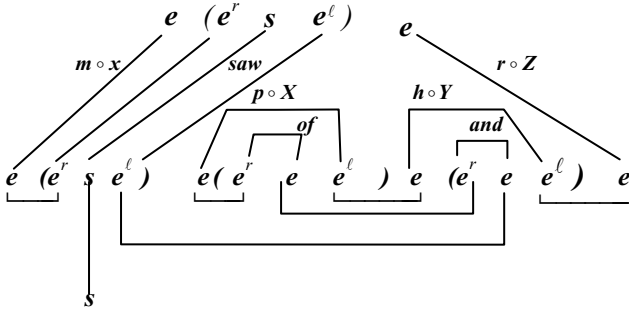


Fig. 3

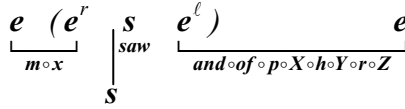


Fig. 4

Reading the labels from left to right and replacing prefix notation by infix notation where necessary we get

$$\text{mary}(x) \wedge \text{saw}(x, \text{and}(\text{of}(X, Y), Z)) \wedge \text{picture}(X) \wedge \text{horse}(Y) \wedge \text{rose}(Z) .$$

Whereas, the second reduction of our example

Peter saw pictures of (horses and roses)

$$\underbrace{n_1(\pi^r s_2 o^l) c_2}_{\text{Peter}} \underbrace{(n_2^r n_2 o^l) c_2}_{\text{saw}} \underbrace{(n^r n_2 n^l) c_2}_{\text{pictures of (horses and roses)}}$$

Fig. 5

is connected to the same semantical declaration

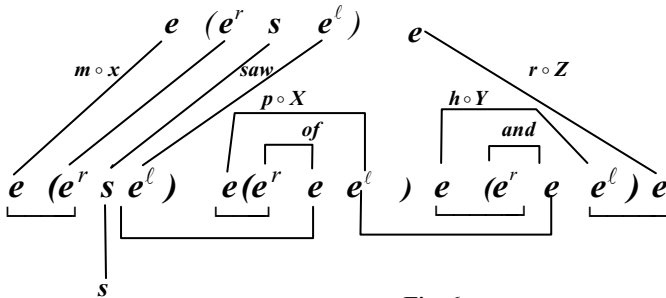


Fig. 6

which combs to

$$\begin{array}{c}
 \begin{array}{c} e \quad (e^r) \\ \hline m \circ x \end{array} \quad \begin{array}{c} s \\ \hline saw \end{array} \quad \begin{array}{c} e^\ell \\ \hline of \circ p \circ X \circ and \circ h \circ Y \circ r \circ Z \end{array} \quad e \\
 \hline
 s
 \end{array}$$

Fig. 7

and hence is interpreted by

$$\text{mary}(x) \wedge \text{saw}(x, \text{of}(X, \text{and}(Y, Z))) \wedge \text{picture}(X) \wedge \text{horse}(Y) \wedge \text{rose}(Z) .$$

5 Conclusion

In this paper, we have concentrated on the categorical properties of derivations with pregroup grammars by embedding these derivations into the compact 2-category of transitions. The categorical properties of transitions intervene in the computation of the meaning associated to a derivation. The algorithm we have sketched has two essential features: It adds a semantical transition to the derivation in *linear* time and then uses the *linear* algorithm of categorical composition to compute the logical formula associated to the analysed sentence. Hence the complexity of computing the meaning(s) of a sentence remains that of finding the correct derivation(s). The properties of such an interpreting algorithm depend essentially on the types associated to words in the pregroup dictionary. The language fragments analysed so far involve syntactical types similar to those of our example, but some standardisation of the pregroup dictionaries may be necessary before a linear algorithm of discourse representation via pregroup grammars becomes possible.

References

1. Joachim Lambek, Type Grammar revisited, in A. Lecomte et al., editors, Logical Aspects of Computational Linguistics, Springer LNAI 1582, pp.1 –27, 1999
2. Samuel Eilenberg, *Automata, Languages and Machines*, vol. A and B, New York: Academic Press, 1972 and 1976
3. Wojciech Buszkowski, Cut elimination for the Lambek calculus of adjoints, in *Abrusci et al. ed.*, Papers in formal linguistics and logic, Bologna: Bulzoni, 2002
4. Saunders Mac Lane, *Categories for the working mathematician*, Springer Verlag, New York 1971
5. Joachim Lambek, Bicategories in algebra and linguistics, in: T. Ehrhard et al., editors, *Linear Logic in computer science*, Cambridge University Press 2004
6. Anne Preller, Joachim Lambek, Free compact 2-categories, rapport de recherche 11439, LIRMM, 2004

Feature Constraint Logic and Error Detection in ICALL Systems

Veit Reuer and Kai-Uwe Kühnberger

University of Osnabrück, Institute of Cognitive Science,
Katharinenstr. 24, 49069 Osnabrück, Germany
{vreuer, kkuehnbe}@uos.de

Abstract. In this paper, an extension of feature constraint logic is presented which allows the coding of errors in feature structures. This is achieved by adding a designated feature to the feature logic with special properties resulting in an expansion of the underlying feature logic. The framework will be formally developed and applied in an ICALL system that allows errors of learners of a foreign-language. Furthermore the system provides an analysis of such errors.

1 Introduction

The recognition of errors made by foreign-language learners is a task that has been tackled with same success using methods from Computational Linguistics. Some advantages of intelligent computer-assisted language learning systems (ICALL systems) which incorporate advanced error-recognition are summarized below:

- An ICALL-system can give precise feedback about the momentary performance of the learner.
- The analyses can be used for learner modeling and individual tutoring.
- Dialog-style exercises can be implemented (but need robust yet sensitive analyses of ill-formed sentences in order to continue a dialog).

From a Computational Linguistics perspective, CALL can also be seen as an interesting field to test its methods. In current research, two main aspects are examined: One aspect concerns the usage of Computational Linguistics' technology to analyze and interpret user input. A second aspect is the use of linguistic resources such as (annotated) corpora and lexical databases for presenting relevant information to language learners.

This paper describes some aspects of the error-recognition module of an interactive ICALL-system for German. Using the system, the language learner is invited to produce complete written sentences in small question-answering-tasks. This setting challenges the learner to use language interactively in order to enhance the development of the so-called communicative competence. Emphasis is put on the possibility of giving adequate feedback to the learner if a

morpho-syntactically ill-formed sentence is encountered. As mentioned, for example, in Menzel and Schröder [12] a program usually does not support both requirements at the same time. That is, if free formed input is allowed, the system is not able to do a detailed analysis of the input. On the other hand, if the error-recognition capabilities are advanced enough to give satisfying feedback, the choice of exercises is in most cases quite limited. Thus the main goal in developing an ICALL-system should be to provide a stimulating environment *and* to give an adequate feedback. Preconditions for this are sensitive parsing and error recognition, which are handled partially by using the method described below.

The plan for the paper can be summarized as follows: In Section 2, several approaches to error recognition in feature structures are discussed and the strengths and weaknesses are evaluated. Section 3 gives an informal and intuitive idea of classical feature constraint logics and ideas of how to incorporate errors in such frameworks by a modified unification procedure as the main structure building process. Section 4 presents a classical formal account for representing feature constraint logics. In Section 5, an expansion of feature constraint logics is developed by adding a designated error feature, a substitution operation, and a modified unification operation. Section 6 presents a practical application of the framework developed in Section 5 in the context of an ICALL-system. Finally, some concluding remarks in Section 7 are added.

2 Error Analysis

Two general strategies for error handling can be distinguished. First, there are the so-called robust parsing methods. These try to continue parsing past a position that cannot be handled by the grammar, without considering the type and exact location of the error. The main purpose is to achieve a result for as much of the input as possible. This usually means a result yielding the largest possible chunks (e.g. Jensen et al. [10]). Second, “sensitive” strategies are being developed for specifically locating and analyzing errors in the input. With the help of some type of correction method, the parsing process continues across the error position and yields a complete description of the input, usually including the position and the type of error. In a system aimed at both determining the grammaticality according to a given grammar and providing as much feedback about an error as possible, only the second type of parsing method can be adopted.

There are two strategies for identifying errors: Either the grammar is extended with additional rules covering the various cases of erroneous input; or the parsing algorithm itself is modified to allow for error recognition, so that the grammar and lexicon need to cover “correct” language only. We call the first concept “anticipation-based” and the second concept “anticipation-less”. In the following we would like to discuss a few aspects of these two approaches to error recognition. An example from Schwind [21] demonstrates one major disadvantage of the first approach. The mal-rule in the phrase structure grammar is specifically designed to describe an error a French native speaker might make when learning German: this is to position the adjective after the noun in

a noun phrase (*le maillot jaune* vs. *das gelbe Trikot*). However, speakers with other mother tongues may produce other errors not covered by this approach. New rules have to be added to the grammar for many other cases.¹ This in turn leads to enormous efficiency problems.

However, two advantages should be noted regarding the anticipation-based approach. One is that the most efficient parsing algorithms can be chosen, as the grammars typically are only extended but not changed in their form. A second advantage is the possibility to be able to distinguish between ungrammatical input on the one hand, and unparsable input on the other, where unparsable input is meant to be input not covered by the grammar. In most cases this means that the feedback to the learner can be stated with more confidence regarding the location and the type of the error.

The anticipation-less approach transfers the load of recognizing and handling errors in the input to the parsing algorithm. An example of this is the approach presented in Menzel [13], where a “model-based” error-diagnosis is characterized by its complexity. Specifically, in an agreement situation, every feature of every lexical item is checked by an individual function in order to allow for a precise localization of a possible error. One important advantage of anticipation-less parsing is the recognition of certain errors “anywhere” if they can be identified at a single position. For example, an agreement error between the subject and a verb should be recognized not only if the subject is in the standard position but also if for some reason the subject is displaced, e.g. by topicalization. A second advantage is the possibility for an independent development of a grammar and a lexicon. They can be engineered in a way such that they generate only descriptions for correct sentences of a language. This also allows the integration of “foreign” data, e.g. a large lexical database that may have been developed in other contexts.

In order to decrease the processing load, the search space for finding a solution may be minimized in anticipation-less concepts. The evaluation in Lee et al. [11] is an example (for a certain type of robust parsing): Using a phrase structure grammar with only 192 rules, the parsing algorithm generates in average 12,000 edges per sentence in the chart with heuristics, and even 25,000 edges without. A different approach is chosen in Schröder et al. [19] and Fouvry [8]. In these two cases constraints are weighted. This allows for concomitant robust parsing and a solution with the “smallest” error measure. Additionally, constraints can be marked with a weight 0, which effectively makes them so-called hard constraints. Solutions with this kind of constraint-clash are not considered for further analysis. However, there is no evaluation whatsoever with respect to a possible feedback to the learner about the error in these two approaches.

Our aim is to avoid the anticipation of errors in the grammar and to use a “sound” grammar theory, namely LFG, which has been used successfully for the description of various linguistic phenomena (e.g., for German, [3] and [1]). Furthermore LFG is based on concepts that can easily be implemented in efficient

¹ A similar case is described in Schneider and McCoy [18].

parsers, and, as it is based on the functional/lexical perspective, it may be readily understood by a language learner. Further indications for the usefulness of LFG in a language learning scenario can be drawn from the use of LFG in language descriptions such as Schwarze's grammar for Italian [20], Thomann's CALL-program for Arabic [23], and Rypa and Feuerman's ICALL-program CALLE [16]. Note that even though other grammar theories are not excluded by these arguments as such, there seems to be a general preference for LFG if any major theory is used at all.

The structural elements of LFG are annotated PS-rules and complex feature-structures (called f-structure in LFG), which mainly contain information about the functional structure of a sentence. The f-structure is built up by using annotations on the PS-rules as guidelines. An extended version of Earley-based chart parsing which is known to be relatively fast, is used for processing PS-rules in our approach. However, in this paper we will concentrate on the recognition of errors in the f-structure. The unification module contains a list of features that restricts the error location in a given f-structure mainly to agreement-features. This corresponds to a two-valued constraint-ranking; either the values of corresponding attributes may clash and the error location and values are stored in the f-structure or the attributes are considered "hard" and the unification fails in case of an error. Note that this approach differs from ideas such as those of Vogel and Cooper [24] and Carpenter [4], that are also able to handle clashing values but have no mechanism to store the information concerning which values actually did not match. However, in a language learning scenario this information is crucial for adequate feedback to the learner.

3 Feature Constraint Logic and Modified Unification

In the last decades, feature constraint logics have become more and more important for applications in Computational Linguistics. Feature constraint formalisms are strongly related to unification-based grammars. Examples of corresponding grammar formalisms that are based on constraint logics are HPSG [15], LFG [2], [5], or Definite Clause Grammars [14]. Whereas HPSG and LFG are grammar formalisms used in (computational) linguistics, Definite Clause Grammars are important for programming languages (e.g. PROLOG). In particular with respect to computational issues and the development of efficient algorithms, feature constraint logics seem to have many advantages. In this paper, we focus on the formal properties of the underlying feature logic in a LFG-like approach.

The idea to use feature constraint logics in LFG is to code grammatical objects like *subject*, *accusative* etc. as functional properties, called features. Because features can be considered as attribute-value pairs, the corresponding representation of, for example, a sentence can be associated with a rooted feature graph where the nodes represent abstract objects and the edges represent features. Because of the fact that well-formedness constraints of the grammar strongly restrict the possible feature graphs, a grammatically erroneous sentence cannot be represented as a well-formed feature graph.

We would like to show how a mechanism can be employed to locate and store errors found in the so-called f-structure. In LFG, the f-structures contain mainly agreement-information but also functional and subcategorizational information. The first two types of features in particular can be used to identify certain types of errors and to give feedback to the learner about these errors. The unification algorithm has been changed to incorporate information about clashing values in the f-structure in order to inform the learner precisely about his/her error. The basic idea is to add a list of attribute-value pairs to every feature structure resulting from a unification with mismatching atomic values.

Several suggestions for the treatment of clashing values were proposed. In Schwind [21], plain disjunctive features are used to keep the clash-information. However, all consequences using disjunctions must be addressed, e.g. the impossibility to use disjunctions in the lexicon for specifying lexical properties. Furthermore, additional principles (“case filter”) must be applied to the resulting structures to select the most sensible one. Neither default unification such as presented in Carpenter [4] nor an application of grammar checking as in Sgvall Hein [17] can be used in the context of language learning, where precise feedback is asked for. In both cases, only one of the clashing values is retained in the feature structure. The crucial information about the error in the construction of the sentence cannot be presented to the learner. Furthermore, there have been several approaches to robust parsing within the HPSG framework such as Vogel and Cooper [24], Fouvry [8], or Foster [6]. Vogel and Cooper’s, as well as Fouvry’s, approach also suffers from the shortcomings mentioned above. It is not clear, how the information about the clash can be kept in the feature-structure. More importantly, Vogel and Cooper suggest a handling of the situation in case of “structure-sharing”, by which “the logic of the resulting feature structure is not well understood” [24–p. 213]. A similar problem is encountered in Foster’s concept. Here, the clashing information is also integrated into the feature structure as in our approach, however, the method adopted by Foster is not monotonic, in contrast to our concept. Additionally, she deals only with agreement errors, which would be a shortcoming in the language learning scenario. To summarize these considerations: To our knowledge there have not been any proposals that integrate the information about clashing values from unification into the resulting feature structure and maintain the desirable properties of feature logics such as monotonicity.

4 Classical Feature Constraint Logic

4.1 Features and Constraints

The following development of classical feature constraint logic is relatively closely related to the development in Smolka [22] and approaches mentioned in this work. Although the following formal development is slightly more general as needed for implementing LFG, we chose the relatively abstract account of feature algebras to introduce feature constraint logic.

We assume that a finite set Con of constants, a finite set $Feat$ of features, and an infinite set Var of variables are given. The denotation of symbols is defined as usual: constants are denoted by letters a, b, c, \dots , features by letters f, g, h, \dots , and variables by letters x, y, z, \dots . The following definition specifies a feature algebra.

Definition 1. A feature algebra \mathfrak{A} is a pair $\langle D, I \rangle$ where D is a non-empty set and I is an interpretation function defined on constants by $I : Con \rightarrow D$ and on features by $I : Feat \rightarrow \mathfrak{P}(D \times D)$ such that the following conditions hold:

- (i) If $\langle a, b \rangle \in I(f)$ and $\langle a, c \rangle \in I(f)$ then $b = c$
- (ii) If $I(a) = I(b)$ then $a = b$
- (iii) For all $a \in Con$ there is no $d \in D$ such that $\langle a, d \rangle \in I(f)$

The first condition specifies that features are interpreted as (partial) functions, the second condition corresponds to the commonly used unique name assumption for constants, and the third condition prohibits an application of features to constants, i.e. constants are considered to be atomic.

For applications in computational linguistics and grammar theory, feature graphs play an important role mainly because of their convenient diagrammatic properties. The following definition makes this concept precise.

Definition 2. A feature graph is either a graph without edges, i.e. a pair $\langle a, \emptyset \rangle$ where $a \in Con$ is the root of the graph or a graph with edges $\langle x, E \rangle$ where $x \in Var$ is the root of the graph and E is a finite set of (feature-labeled) edges of the form yfs where $y \in Var$, $f \in Feat$, and $s \in Con \cup Var$ such that the following three conditions are satisfied:

- (i) Edges are uniquely defined: If $yfs \in E$ and $yft \in E$ then $s = t$.
- (ii) The graph is connected: If $yfs \in E$ then E contains a path of edges from the root x to y .
- (iii) The graph is acyclic: If $yfs \in E$ then there is no path of edges leading from s to y .

According to Definition 2 a feature graph is a rooted, connected, acyclic, and directed graph.² The edges of a feature graph are labeled with features and the nodes with variables or constants. Notice that we require that the set of edges E is finite.

Subgraphs of a given graph G are defined as usual, namely as an embedding of the subgraph into the matrix graph. Definition 3 makes this idea precise.

Definition 3. Given two graphs G and G' where the root of G is $x \in Con \cup Var$, G is called a subgraph of G' iff there is a homomorphic embedding $h : G \rightarrow G'$ such that: $h(x) = x$ and for all edges $yfs \in G : h(yfs) = yfs$.

² Alternative definitions of feature graphs allow cyclic graphs as well, compare for example Smolka [22].

Obviously, Definition 3 induces a partial order relation \leq on subgraphs of a given graph G' by defining: $G \leq G'$ if and only if G is a subgraph of G' . Notice further that for every graph G' and variable or constant x occurring in G' there is a uniquely defined maximal subgraph G of G' with root x .

Following [22], the collection of all feature graphs can be used to define a new structure called a feature graph algebra. It turns out that this feature graph algebra is itself a feature algebra (compare Fact 5).

Definition 4. Assume a feature algebra $\mathfrak{A} = \langle D, I \rangle$ is given. A feature graph algebra \mathcal{F} is a pair $\langle \mathbf{D}_{\mathfrak{A}}, I_{\mathfrak{A}} \rangle$ such that:

- (i) $\mathbf{D}_{\mathfrak{A}}$ is the set of all feature graphs
- (ii) For $a \in \text{Con}$: $I_{\mathfrak{A}}(a) = \langle a, \emptyset \rangle$
- (iii) $I_{\mathfrak{A}}(f) = \langle G', G \rangle$ iff $xf s \in G'$ where x is the root of G' and G is the maximal subgraph with root s in G'

Fact 5. (Smolka) A feature graph algebra satisfies the properties of a feature algebra.

Proof: Given a feature graph algebra \mathcal{F} the required conditions (ii) and (iii) of Definition 1 are clearly satisfied. For condition (i) of Definition 1 consider $I_{\mathfrak{A}}(f) = \langle G', G_1 \rangle$ and $I_{\mathfrak{A}}(f) = \langle G', G_2 \rangle$. Assume $G_1 \neq G_2$. Because G_1 and G_2 are maximal subgraphs and because of the fact that maximal subgraphs are uniquely defined, the root s_1 of G_1 and the root s_2 of G_2 are not equal. Then, the corresponding feature graph G' does not satisfy condition (i) of Definition 2. Hence, we have a contradiction. Therefore, $G_1 = G_2$. q.e.d.

Usually feature graphs are considered to be preordered by a subsumption relation \preceq determining the amount of information coded in the feature graph. The importance of the subsumption relation \preceq can be traced back to the fact that \preceq is used for defining the unification operation on feature graphs. Following Smolka [22] we want to build this subsumption relation bottom up by defining first subsumption on feature algebras and extending this to feature graphs. Definition 6 specifies a subsumption preorder \preceq on a feature algebra \mathfrak{A} .

Definition 6. Assume $\mathfrak{A} = \langle D, I \rangle$ is a feature algebra. A subsumption preorder \preceq on \mathfrak{A} is a preorder defined on D satisfying the following property: $a \preceq b$ iff there is a partial mapping $\phi : D \rightarrow D$ such that the following conditions (i) – (iii) hold.³

- (i) For all $c \in \text{Con}$ it holds $\phi(I(c)) = I(c)$
- (ii) If $f \in \text{Feat}$, $d \in D$, and $(I(f))(d)$ as well as $\phi(d)$ are defined, then $\phi[(I(f))(d)]$ and $(I(f))(\phi(d))$ are defined such that $(I(f))(\phi(d)) = \phi[(I(f))(d)]$
- (iii) $\phi(a) = b$

³ According to Definition 1 a feature f is interpreted as a (partial) function. We write $(I(f))(d)$ as the result of mapping $d \in D$ under $I(f)$ in the feature algebra \mathfrak{A} .

The intuition of Definition 6 is to make the concepts of generality and specificity precise in the following sense: if $a \preceq b$ then a is more general than b (or b is more specific than a). Notice that \preceq is not a partial order on D , because it does not hold in general that if $a \preceq b$ and $b \preceq a$ then $a = b$. A natural generalization of Definition 6 concerns paths. Given a feature algebra $\mathfrak{A} = \langle D, I \rangle$ we define a path p as a concatenation of features such that $p = f_n \circ f_{n-1} \circ \dots \circ f_1$. Notice that features are partial functions, hence paths are crucially compositions of functions, i.e. $I(p)$ is the composition of $I(f_n), \dots, I(f_1)$. If p is empty, then the interpretation $I(p)$ is the identity function on D . Hence, the interpretation of paths is recursively induced by the interpretation of features: $(I(p))(a) = b$ iff $\langle a, b \rangle \in I(p)$. As already mentioned above, Definition 6 can similarly be extended to paths: if $\phi : D \rightarrow D$ is a partial homomorphism relative to a given feature algebra $\mathfrak{A} = \langle D, I \rangle$, $d \in D$, and p is a path, then it holds: if $\phi(d)$ and $(I(p))(d)$ is defined, then $\phi[(I(p))(d)]$ and $(I(p))(\phi(d))$ is defined and $\phi[(I(p))(d)] = (I(p))(\phi(d))$.

4.2 The Subsumption Relation on Feature Graph Algebras

Using the machinery developed in Subsection 4.1 we give a characterization of the subsumption preorder \preceq on a feature graph algebra.

Fact 7. (Smolka) *Assume $G = \langle x_G, E_G \rangle$ and $G' = \langle x_{G'}, E_{G'} \rangle$ are feature graphs. Using Definition 6 a preorder on feature graphs can be induced: $G \preceq G'$ iff there exists a mapping $\psi : \text{Var}_G \cup \text{Con}_G \rightarrow \text{Var}_{G'} \cup \text{Con}_{G'}$ such that it holds:*

- (i) $\psi(x_G) = x_{G'}$
- (ii) For all $a \in \text{Con}_G : \psi(a) = a$
- (iii) If $xf s \in E_G$ then: $\psi(x)f\psi(s) \in E_{G'}$

Proof: “ \Rightarrow ” Assume G and G' are given according to the Fact. We define a partial endomorphism $\phi : D \rightarrow D$ according to Definition 6 such that $\phi(G) = G'$. We define $\psi : \text{Var}_G \cup \text{Con}_G \rightarrow \text{Var}_{G'} \cup \text{Con}_{G'}$ such that every element $s \in \text{Var}_G \cup \text{Con}_G$ is mapped to $\psi(G_s)$ where G_s is the maximal subgraph generated by s . Condition (i) is satisfied because it holds: if x_G is the root of G , then $\psi(x_G)$ is the root of G' since $\phi(G_{x_G}) = G'_{\psi(x_G)}$. Condition (ii) is obvious and condition (iii) holds because of the following reasoning provided that $xf s \in E_G$ holds:

$$G'_{\psi(s)} = \phi(G_s) = \phi[(I(f))(G_x)] = (I(f))(\phi(G_x)) = (I(f))(G'_{\psi(x)})$$

Hence, $\psi(x)f\psi(s) \in E_{G'}$.

“ \Leftarrow ” Assume ψ is given as above. We need to show that a partial endomorphism $\phi : D \rightarrow D$ satisfies the conditions in Definition 6. We define ϕ as follows: $\phi(G_x) = G'_{\psi(x)}$. Then it holds: $\phi(G) = \phi(G_x) = G'_{\psi(x)} = G'$. Therefore $\phi(I(x)) = I(x)$. We need to show that $\phi[(I(f))(G_x)] = (I(f))(\phi(G_x))$ for given and defined f . Assume $xf s \in E_G$, then $\psi(x)f\psi(s) \in E_{G'}$. Hence:

$$\phi[(I(f))(G_x)] = \phi(G_s) = G'_{\psi(s)} = (I(f))(G'_{\psi(x)}) = (I(f))(\phi(G_x)). \quad \text{q.e.d.}$$

Although the presented classical account suffices to give grammar theories such as LFG a sound formal basis, problems arise if we try to apply this framework to ICALL systems. The reason is that such systems cannot deal with errors. The following section presents a solution to this problem by the introduction of a designated error feature *err* with some special properties.

5 An Extension of Feature Constraint Logic for Error Recognition Systems

5.1 Introductory Example

In this subsection, we would like to present an intuitive example prior to the introduction of the formal specifications of the extended approach. Consider the sentence “Ich habe jetzt eine Unfall gesehen”. The sentence is almost correct except for the agreement between the determiner and the noun in the object NP. “gesehen” calls for an Accusative object however “Unfall” is Accusative Masculin whereas “eine” is Accusative Feminin. As mentioned above the resulting structure generated by an ICALL system should ideally describe a corrected version of the sentence without neglecting the error, i.e. the clashing feature values. Figure 1 shows the (simplified) corresponding lexical information for “eine” and “Unfall” in a LFG-style representation.

“eine”	“Unfall”
$\left[\begin{array}{ll} \text{def} & : - \\ \text{gen} & : f \\ \text{num} & : \text{sg} \\ \text{case} & : \text{acc} \end{array} \right]$	$\left[\begin{array}{ll} \text{pred} & : \text{'ACCIDENT'} \\ \text{gen} & : m \\ \text{num} & : \text{sg} \\ \text{case} & : \text{acc} \end{array} \right]$

Fig. 1. Lexical entries for “eine” and “Unfall”

Notice that the words “eine” and “Unfall” can also have different properties which we ignore here. The unification of these two feature structures should fail in the normal case because the values of the “gen”-feature clash. In our approach however the clash of these atomic values leads to the insertion of a designated feature named “err” containing the relevant information about the clash as elements of a set. Figure 2 demonstrates the mechanism. The resulting feature structure is the description of the phrase “eine Unfall” in object position including the information that a clash between values of a feature took place. In order to build the resulting feature structure the parsing continues as usual, however the error is encoded in the structure itself and can be used for feedback to the user.

$$\begin{array}{c} \left[\begin{array}{l} \text{def} \quad : - \\ \text{gen} \quad : f \\ \text{num} \quad : \text{sg} \\ \text{case} \quad : \text{acc} \end{array} \right] \sqcap_{err} \left[\begin{array}{l} \text{pred} \quad : \text{'ACCIDENT'} \\ \text{gen} \quad : m \\ \text{num} \quad : \text{sg} \\ \text{case} \quad : \text{acc} \end{array} \right] = \left[\begin{array}{l} \text{def} \quad : - \\ \text{gen} \quad : f \\ \text{num} \quad : \text{sg} \\ \text{case} \quad : \text{acc} \\ \text{pred} \quad : \text{'ACCIDENT'} \\ \hline \text{err} \quad : \left\{ \left[\text{gen} : m \right] \right\} \end{array} \right]
\end{array}$$

Fig. 2. Example of a unification with mismatching values for the gender feature

5.2 Adding an Error Feature to Feature Constraint Logic

In order to adjust the previously defined feature constraint logic to sensitive error recognition, we propose to integrate a designated additional error feature denoted by *err*. This feature has certain different properties in comparison to standard features of the classical theory. Since the unification process as the main structure building process should entirely be based on the definition of the subsumption relation we start off by providing a suitable definition of substitution and the relevant modifications of Fact 7.

Definition 8. Assume a feature graph G_{\oplus} extended by a designated feature *err* is given where *err* is defined by the nodes x and $\{[f_1 : y_1], \dots, [f_n : y_n]\}$. A substitution Θ on G_{\oplus} is a contravariant pair of substitution functions $\Theta = \langle \Theta_i^{\wedge}, \Theta_i^{\vee} \rangle$ such that the following two conditions hold:

- (i) Θ_i^{\wedge} substitutes x_i by y_i , provided that $i \in \{1, \dots, n\}$, $x f_i x_i \in G_{\oplus}$, $x_i \in \text{Con}$, and $y_i \in \text{Con}$.
- (ii) Θ_i^{\vee} substitutes y_i by x_i , provided that $i \in \{1, \dots, n\}$, $x f_i x_i \in G_{\oplus}$, $x_i \in \text{Con}$, and $y_i \in \text{Con}$.

The intuition of Definition 8 is that an application of a substitution to a feature graph including a *non-empty* error feature *err* is a bidirectional substitution of the value of a feature f in *err* by a value of the feature f occurring in the feature graph elsewhere.

The subsumption relation \preceq needs to be slightly modified in order to incorporate substitutions. Because the subsumption relation \preceq is defined on the domain D of a feature algebra \mathfrak{A} we use Θ to induce a substitution operation $\bar{\Theta} = \langle \bar{\Theta}^{\wedge}, \bar{\Theta}^{\vee} \rangle$ on D by the following two conditions (a) and (b):

- (a) $\bar{\Theta}^{\wedge}$ substitutes $I(x_i)$ by $I(y_i)$ iff Θ^{\wedge} substitutes x_i by y_i .
- (b) $\bar{\Theta}^{\vee}$ substitutes $I(y_i)$ by $I(x_i)$ iff Θ^{\wedge} substitutes y_i by x_i .

Using $\bar{\Theta}$ we can reformulate Definition 6 for feature structures including an error feature *err*.

Definition 9. Assume $\mathfrak{A} = \langle D, I \rangle$ is a feature algebra including a designated error feature *err*. A subsumption preorder \preceq on \mathfrak{A} is a preorder defined on D

such that it holds: $a \preceq b$ iff there is a partial mapping $\phi : D \rightarrow D$ such that the following conditions hold:

- (i) For all $c \in \text{Con}$ it holds $\phi(I(c)) = I(c)$
- (ii) If $f \in \text{Feat} \setminus \{\text{err}\}$ and $(I(f))(d)$ and $\phi(d)$ are defined, then there exists a (possible empty) substitution $\bar{\Theta}$ such that $\bar{\Theta}[\phi[(I(f))(d)]]$ as well as $(I(f))[\bar{\Theta}(\phi(d))]$ are defined and the following equation holds:

$$\bar{\Theta}[\phi[(I(f))(d)]] = (I(f))[\bar{\Theta}(\phi(d))]$$
- (iii) $\phi(a) = b$

Given the substitution operation Θ of Definition 8 and the definition of subsumption according to Definition 9 we can extend Fact 7 to feature graphs including error features as follows:

Fact 10. Assume $G_{\oplus} = \langle x_{G_{\oplus}}, E_{G_{\oplus}} \rangle$ and $G'_{\oplus} = \langle x_{G'_{\oplus}}, E_{G'_{\oplus}} \rangle$ are feature graphs extended by designated error features err_1 and err_2 , respectively. The feature err_1 is defined by the nodes z and $\{[f_1 : y_1], \dots, [f_n : y_n]\}$ and the feature err_2 is defined by the nodes z' and $\{[f'_1 : y'_1], \dots, [f'_m : y'_m]\}$. A preorder on feature graphs is induced by: $G_{\oplus} \preceq G'_{\oplus}$ iff there exists a mapping $\psi : \text{Var}_{G_{\oplus}} \cup \text{Con}_{G_{\oplus}} \rightarrow \text{Var}_{G'_{\oplus}} \cup \text{Con}_{G'_{\oplus}}$ such that it holds:

- (i) $\psi(x_{G_{\oplus}}) = x_{G'_{\oplus}}$
- (ii) For all $a \in \text{Con}_{G_{\oplus}}$: $\psi(a) = a$
- (iii) If $xfs \in E_{G_{\oplus}}$ then: $\psi(x)f\psi(s) \in E_{G'_{\oplus}}$
- (iv) If $xfs \in E_{G_{\oplus}}$, $\psi(x)fs' \in E_{G'_{\oplus}}$, and $\psi(x)f\psi(s) \notin E_{G'_{\oplus}}$ then there exists a substitution Θ on G'_{\oplus} such that $\psi(x)f\Theta(\psi(s)) \in E_{G'_{\oplus}}$ and furthermore: $\{y_1, \dots, y_n\} \subseteq \Theta[\{y'_1, \dots, y'_m\}]$

Proof: The properties (i)-(iii) are similar to Fact 7. Condition (iv) requires that there is a substitution replacing a feature by a member of the error feature in order to establish the subsumption relation and the error list of G_{\oplus} is included in the error list of G'_{\oplus} after the application of the substitution Θ . It is straightforward to show that this is induced by Definition 9. q.e.d.

Notice that the definition of substitution is restricted to the application of Θ to features with constants as values, therefore substitutions need not be extended to paths in constraint feature graphs. Nevertheless another extension is straightforward. This concerns multiple substitutions on feature graphs. Provided that argument-value pairs are disjoint multiple substitutions are simply iterated applications of the involved substitutions $\Theta_1, \dots, \Theta_n$.

5.3 Using Substitutions for Unification

In this subsection, we want to use feature structures extended by an error feature err for unification. Unification should be defined as usual based on the subsumption order. However the following issues need clarification:

- In order to make the unification process homogeneous the building of equivalence classes of feature graphs is necessary.
- Classically, two inconsistent feature graphs cannot be unified. In our approach, two inconsistent feature graphs can be unified by incorporating the inconsistent information in the designated feature *err*.
- In a unification process, the union of the error lists of two feature graphs needs to be computed (modulo substitutions).

Concerning the first point we define an equivalence relation on feature graphs, collapsing feature graphs that subsume each other in an equivalence class.

Definition 11. *Given a feature graph algebra \mathcal{F} , an equivalence relation \sim on feature graphs is defined as follows:*

$$G_{\oplus} \sim G'_{\oplus} \quad \Leftrightarrow \quad G_{\oplus} \preceq G'_{\oplus} \wedge G'_{\oplus} \preceq G_{\oplus}$$

We denote the equivalence class of a feature graph G_{\oplus} with $[G_{\oplus}]_{\sim}$. Clearly Definition 11 can be used to define a modified subsumption relation \preceq_{\sim} that induces a partial order relation on equivalence classes of feature graphs. More formally we define:

$$[G_{\oplus}] \preceq_{\sim} [G'_{\oplus}] \quad \Leftrightarrow \quad G'_{\oplus} \preceq G_{\oplus}$$

The modified subsumption relation \preceq_{\sim} will be used for the modified unification of two feature graphs. The following Fact 12 specifies some properties of the relation \preceq_{\sim} .

Fact 12. *Given a feature graph algebra $\mathcal{F} = \langle \mathbf{D}_{\mathfrak{B}}, I_{\mathfrak{B}} \rangle$ with error feature *err* the following claims hold:*

- (i) *The subsumption relation \preceq_{\sim} is reflexive, antisymmetric, and transitive, i.e. it induces a partial order relation.*
- (ii) *For every pair of feature graphs $[G_{\oplus}]$ and $[G'_{\oplus}]$ the greatest lower bound $[G_{\oplus}] \sqcap [G'_{\oplus}]$ exists.*

Proof: (i) Clearly $[G_{\oplus}] \preceq_{\sim} [G_{\oplus}]$ holds, hence \preceq_{\sim} is reflexive. Transitivity follows easily by reducing \preceq_{\sim} to \preceq . For antisymmetry assume that $[G_{\oplus}] \preceq_{\sim} [G'_{\oplus}]$ and $[G'_{\oplus}] \preceq_{\sim} [G_{\oplus}]$. Then it must also hold: $G_{\oplus} \preceq G'_{\oplus}$ as well as $G'_{\oplus} \preceq G_{\oplus}$ by the definition of \preceq_{\sim} . But then, both feature graphs are in the same equivalence class by Definition 11: $G_{\oplus} \in [G_{\oplus}]$ and $G'_{\oplus} \in [G_{\oplus}]$. Hence, antisymmetry holds as well.

(ii) Assume two feature graphs $[G_{\oplus}]$ and $[G'_{\oplus}]$ are given. Consider a feature graph $[\bar{G}_{\oplus}]$ specified by the following properties:

- $[\bar{G}_{\oplus}]$ contains all edges $xf s$ that are occurring in $[G_{\oplus}]$ and $[G'_{\oplus}]$.
- Each edge $xf s$ that occurs either in $[G_{\oplus}]$ or in $[G'_{\oplus}]$ is also in $[\bar{G}_{\oplus}]$.
- If there is an edge $xf s \in [G_{\oplus}]$ and an edge $xf s' \in [G'_{\oplus}]$ and $s \neq s'$ then $xf s \in \bar{G}_{\oplus}$ and $[f : s']$ is written in the error list of \bar{G}_{\oplus} .
- $[\bar{G}_{\oplus}]$ does not contain any other edges, values, or constants.

Obviously $[\bar{G}_\oplus]$ is a lower bound of both feature graphs $[G_\oplus]$ and $[G'_\oplus]$. Assume $[H_\oplus]$ is an arbitrary lower bound of $[G_\oplus]$ and $[G'_\oplus]$. If $[H_\oplus]$ contains a feature g or a value s that is not contained in $[\bar{G}_\oplus]$, then $[H_\oplus]$ cannot be greater than $[\bar{G}_\oplus]$. If all features and values are occurring in both feature graphs $[\bar{G}_\oplus]$ and $[H_\oplus]$, then by substitution both graphs subsume each other and therefore are in the same equivalence class. Hence $[\bar{G}_\oplus]$ is the greatest lower bound of $[G'_\oplus]$ and $[G_\oplus]$. q.e.d.

We will write $[\mathbf{D}]_{\mathfrak{B}}$ for set of all equivalence classes of feature graphs relative to a given feature algebra \mathfrak{B} including the designated *err* feature.

Corollary 13. *The structure $\langle [\mathbf{D}]_{\mathfrak{B}}, \preceq \rangle$ is a semilattice.*

Proof: Follows directly from Fact 12. q.e.d.

The unification of two feature graphs can now be considered as the computation of the greatest lower bound of the two graphs with respect to the partial order relation \preceq .

Definition 14. *The unification of two feature graphs $[G_\oplus]$ and $[G'_\oplus]$ is defined as the greatest lower bound $[G_\oplus] \sqcap [G'_\oplus]$ in the structure $\langle [\mathbf{D}]_{\mathfrak{B}}, \preceq \rangle$.*

The unification operation defined in Definition 14 corresponds exactly to the usual definition of unification induced by the modified subsumption relation \preceq resulting in a partially ordered structure $\langle [\mathbf{D}]_{\mathfrak{B}}, \preceq \rangle$.

A remark concerning structure sharing should be added: As already mentioned in [24], the introduction of structure sharing may cause some difficulties, because of the propagation of the error to many feature value pairs. In our approach, the substitution Θ applied to a structure shared object $f : [1]a$ would iteratively be applied further to all feature-value pairs that are coreferenced with $f : [1]a$. By application of a substitution Θ the indices are not mapped into the error list. Rather indices are strictly connected to the corresponding feature.

With the proposed formal machinery we can handle occurring clashes in feature structures as well as the storing and subsequent analysis of the information contained in the extended feature structure. The following section roughly describes a scenario for an application.

6 Practical Application

In order to assess the value of the proposed mechanism, the annotated Heringer-Corpus [9] containing 7107 sentences with errors produced by learners of German as a foreign language was analyzed with respect to frequently occurring error types. Note that the Heringer-Corpus in general does not only include morphosyntactic errors, but also orthographic and morphological errors. The analysis showed that the distribution of morphosyntactic errors is strongly limited to certain positions and error types, e.g. missing prepositions account for

only 1.5% of all errors and may be neglected in a first approach. With regard to feature structures this means that the occurrences of errors can be limited to features which actually make sense to a language learner: “Classical” features such as Case, Number, and Gender, but also features like Auxiliar Type and for German adjective agreement the feature Inflection Type. Certain other features may not be allowed to fail as they are probably less of a linguistic nature but build in to “streamline” the grammar.

The analysis of error types further revealed that around 35% of all errors should be dealt with inside the feature structure in a LFG-type analysis. In an evaluation 75 sentences with morphosyntactic errors were randomly chosen from the Heringer-Corpus and another 75 were collected from trials with the mentioned ICALL-system. These sentences were manually error-tagged and contained 96 errors which should be recognized in the f-structure of the analysis.

Table 1. Evaluation of errors in f-structure (correct / a.o. / none / fail)

Error-Type	Heringer + ICALL	Total
Agreement/Government		
with/in Subject	11 / 3 / 4 / –	18
with/in Object	17 / 6 / 3 / 1	27
with/in Prep.-Object	20 / 1 / 1 / 3	25
POS-error	– / – / 2 / 1	3
Verb form	11 / – / 2 / –	13
Auxiliar	9 / – / – / 1	10
Total	68 / 10 / 12 / 6	96
%	71 / 10 / 13 / 16	100

As Table 1 shows, 68 errors were correctly identified by the proposed mechanism compared to the manual tagging with a very simple preference scheme preferring the result with less clashes. 10 errors were identified among other non preferred ones with an identical error measure. For 12 errors the resulting f-structure with the least clashes did not show the expected error and for 6 errors the analysis failed because of either an arbitrary threshold of 20,000 edges in the chart or because a final edge could not be generated. As the grammar was designed to cover the correct versions of the sentences following [7], no false positive occurred. In summary 71% of the errors were identified correctly and another 10% were identified among others which shows the value of our approach.

As an example for feedback generation consider again the German sentence “Ich habe jetzt eine Unfall gesehen”. The f-structure in Figure 3 is a highly abbreviated result at the S-node of the actual parsing process. In order to determine which concrete words are affected by the feature mismatch the tree has to be traversed down to the node marked with the annotation “obj”. In the current case all words below this node need to be examined with the features affected by the mismatch. As it turns out “eine” has the feature combination [gen : f, num : sg, case : acc] whereas “Unfall” has the features [gen : m, num : sg, case : acc]. This

$$\left[\begin{array}{lcl} \text{pred} & : & \text{'SEHEN' } \langle \text{subj, obj} \rangle \\ \text{subj} & : & \left[\text{pred} : \text{'PRO'} \right] \\ \text{obj} & : & \left[\begin{array}{lcl} \text{gen} & : & \text{f} \\ \text{pred} & : & \text{'ACCIDENT'} \\ \hline \text{err} & : & \{ [\text{gen: m}] \} \end{array} \right] \end{array} \right]$$

Fig. 3. Feature structure including *err* feature

can then be reported to the learner in an extensive message stating that *there is an error in the Object NP, namely a mismatch in Gender between the determiner “eine” being Feminine and the noun “Unfall” being Masculine*. A correction of the phrase would need, first, a heuristics in order to determine which lexical item needs to be corrected (in this case the determiner) and, second, a new lookup in the lexicon for a determiner matching all relevant features except for the gender feature. Note that straight forward correction strategies for a substantial amount of cases cannot easily be determined as shown e.g. for German in [13].

7 Conclusions

In this paper, we have shown that an extension of constraint feature logics can be achieved by adding a designated error feature *err* and a substitution Θ defined on feature graphs such that errors can be homogeneously stored and tracked. By defining a modified subsumption relation inducing a partial order on equivalence classes of feature graphs a straightforward definition of unification can be achieved storing all information about the clash. This is motivated by applications of feature grammars for ICALL systems where error detection, error analysis, and error feedback are essential for the language learner. On the practical side there is a natural extension to grammar checking systems. Furthermore we believe that the approach can be used for the modeling of robust parsing mechanisms of humans in a formally sound manner.

References

1. Berman, J.: Clausal Syntax of German. CSLI Publications, Stanford (2003)
2. Bresnan, J.: Lexical-Functional Grammar. Oxford: Blackwell Publisher (2001)
3. Butt, M., King, T. H., Niño, M., Segond, F.: A Grammar Writer’s Cookbook. CSLI Publications, Stanford (1999)
4. Carpenter, B.: Skeptical and Credulous Default Unification with Applications to Templates and Inheritance. In: Briscoe, T., Copestake, A., Paiva, V. (eds.): Inheritance, Defaults and the Lexicon. Cambridge University Press, Cambridge (1993) 13–37
5. Dalrymple, M., Kaplan, R. M., Maxwell, J. T., Zaenen, A. (eds.): Formal Issues in Lexical-Functional Grammar, CSLI Publications, Stanford (1995)

6. Foster, J.: A Unification Strategy for Parsing Agreement Errors. In: Pilière, C. (ed.): *Proceedings of the 5th ESSLI 2000 Student Session*, Birmingham (2000)
7. Foster, J.: Parsing Ungrammatical Input: An Evaluation Procedure. In: *Proceedings of LREC 2004*, Lisbon (2004) 2039–2042
8. Fouvry, F.: Constraint relaxation with weighted feature structures. *Papers from IWPT2003, 8th International Workshop of Parsing Technologies*, Nancy (2003)
9. Heringer, H. J.: *Aus Fehlern lernen*. Augsburg (1995) – CD-ROM for Win9x/NT
10. Jensen, K., Heidorn, G. E., Miller, L. A., Ravin, Y.: Parse Fitting and Prose Fixing: Getting Hold on Ill-formedness. *Computational Linguistics* **9**(3-4) (1983) 147–160
11. Lee, K. J., Kweon, Cheol J., Seo, J., Kim, G. C.: A Robust Parser Based on Syntactic Information. In: *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Dublin (1995) 223–228
12. Menzel, W., Schröder, I.: Constraint-based Diagnosis for Intelligent Language Tutoring Systems. *Fachbereich Informatik, Universität Hamburg*, Report Nr. FBI-HH-B-208-98 (1998)
13. Menzel, W.: Modellbasierte Fehlerdiagnose in Sprachlehrsystemen. Niemeyer, Tübingen (1992)
14. Pereira, F., Warren, D.: Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence* **13** (1980) 231–278
15. Pollard, C., Sag, I.: *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago (1994)
16. Rypa, M., Feuerman, K.: CALLE: An Exploratory Environment for Foreign Language Learning. In: Holland, V. M., Kaplan, J. D., Sams, M. R. (eds.): *Intelligent Language Tutors*, Earlbaum, Mahwah, NJ (1995) 55–76
17. Sågval Hein, A.: A Grammar Checking Module for Swedish. *Uppsala University*, Uppsala (1998) – SCARRIE Deliverable 6.6.3
18. Schneider, D., McCoy, K. F.: Recognizing Syntactic Errors in the Writing of Second Language Learners. In: *Proc. 17th Int. Conference on Computational Linguistics (COLING)*, Montreal (1998) 1198–1204
19. Schröder, I., Menzel, W., Foth, K., Schulz, M.: Modeling Dependency Grammar with Restricted Constraints. In: *T.A.L.*, **41**(1) (2000) 113–142
20. Schwarze, Ch.: *Grammatik der italienischen Sprache*. Niemeyer, Tübingen (1995)
21. Schwind, C.: Sensitive Parsing: Error Analysis and Explanation in an Intelligent Language Tutoring System. In: *Proc. 12th Int. Conference on Computational Linguistics* (1988) 608–613
22. Smolka, G.: Feature Constraint Logics for Unification Grammars, *Journal of Logic Programming* **12** (1992) 51–87
23. Thomann, J.: LFG as a Pedagogical Grammar. In: *Proceedings of LFG02*, Athens (2002) 366–372
24. Vogel, C., Cooper, R.: Robust Chart Parsing with Mildly Inconsistent Feature Structures. In: Schöter, A., Vogel, C. (eds.): *Nonclassical Feature Systems Vol. 10*. Edinburgh University (1995) 197–216

Linguistic Facts as Predicates over Ranges of the Sentence

Benoît Sagot

INRIA-Rocquencourt, Projet Atoll,
Domaine de Voluceau, Rocquencourt B.P. 105,
78 153 Le Chesnay Cedex, France

Abstract. This paper introduces a novel approach to language processing, in which linguistic facts are represented as predicates over ranges of the input text, usually, but not limited to, ranges of the current sentence. Such an approach allows to build non-linear analyses with a polynomial parsing complexity that take into account simultaneously and with the same technical status morphological, syntactical and semantical properties, this list being non limitative. Classical analyses, such as constituency trees, dependency graphs, topological boxes and predicate-arguments semantics are then obtained as partial projection of a complete analysis. The formalism presented here is based upon Range Concatenation Grammars (hereafter RCG), and has been successfully implemented, thanks to a previously existing RCG parser and a syntactico-semantical grammar for French.

1 Introduction

The definition of an adequate formalism for natural language processing consists in the search of an optimal balance between linguistic validity and computational efficiency. Moreover, a newly defined formalism can be considered interesting only if it is really implemented, with a complete parser and a large-coverage grammar for an example language, and if it shows interesting properties that are not present in other formalisms, or at a more expensive computational cost.

In this paper, we introduce a new formalism that, we think, has all the above-mentioned properties. This formalism relies on the hypothesis that linguistic properties are best described as predicates over continuous ranges of the input sentence¹ or of a bounded amount of extra tokens (e.g. contextual syntagms²). Range Concatenation Grammars (hereafter RCGs), introduced by Boullier, provide a appropriate basis to develop such a formalism. In the remainder, we will present RCGs and their most important properties. Then we will

¹ In itself this idea is not new, and has been used for example in Datalog grammars [1] or in the constraint logic programming implementations of Property Grammars [2]. However, as made clear later, our formalism relies also on three fundamental properties of RCGs: range concatenation, non-linearity and parsing time polynomiality.

² This possibility, already implemented, will not be treated in this paper.

introduce our formalism, called *Meta-RCG*, whose grammars can be converted into RCGs. Finally we will present a fragment of our grammar for French through two examples, in order to show how morphological, syntactical and semantical properties interact continuously with each other, leading to global analyses from which classical analyses (constituency tree, dependency graph, topological boxes, predicate-arguments semantics) can be extracted.

It is worth saying that one of the most important motivations for this work is to develop a formalism in which the two following major constraints are satisfied:

- Morphology, syntax, and, more originally, lexical semantics (and if possible more general semantics) are dealt with *simultaneously* during parsing; in particular, there must be no artificial boundary between a syntactic backbone, syntactic features (or decorations) and lexical semantics.
- Parsing must be computationally tractable, and we make the hypothesis that we can assume a polynomial parsing time.

We discuss these hypotheses more deeply at the beginning of section 3.

2 Range Concatenation Grammars

Range Concatenation Grammars (RCGs) have been introduced by Boullier (see for example [3], and applications thereof in [4]). They defined a class of languages, Range Concatenation Languages (RCLs), that covers exactly *PTIME*, the class of all languages recognizable in deterministic polynomial time. Therefore, RCGs are more powerful than all Mildly Context-Sensitive formalisms, such as Linear Context-Free Rewriting Systems (LCFRS) or Multi-Component Tree-Adjoining Grammars (MC-TAGs), while remaining computationally tractable. We shall now define formally RCGs, following [3].

2.1 Positive RCGs

A *positive RCG* is a 5-tuple $G = (N, T, V, P, S)$ in which:

- N is a finite set of *predicate names*,
- T is a finite set of *terminal symbols*,
- V is a finite set of *variable symbols* such as $T \cap V = \emptyset$,
- $S \in N$ is the *axiom*,
- P is a finite set of *clauses*, which are defined hereafter.

A clause C has the form $\psi_0 \rightarrow \psi_1 \dots \psi_j \dots \psi_m$, where $m \geq 0$ and each ψ_j is a *predicate* of the form $A(\alpha_1, \dots, \alpha_i, \dots, \alpha_p)$, where $p \geq 1$ is its *arity*, $A \in N$, and each α_i is an *argument* of A . Each argument α_i of A has the form $X_1 \dots X_l \dots X_q$, where each X_l is in $V \cup T$. The *left-hand part* of C is ψ_0 , its *right-hand part* is $\psi_1 \dots \psi_j \dots \psi_m$. The predicates of the right-hand side form a set of predicates, which means that order does not matter and that duplicating a predicate is useless. In the following, we will abusively denote a predicate by its name, thus

speaking of the *predicate* A . The arity of the axiom has to be 1, and the arity of a predicate A is fixed. We call A -clause a clause whose left-hand side predicate is A . We define the arity of an A -clause by the arity of A , and the arity of a grammar by the maximal arity of its predicates. An RCG of arity k is a k -RCG.

As said before, the definition of a language by an RCG relies on the notion of range of the input string. Given a string $w = a_1 \dots a_n$ of terminal symbols ($w \in T^*$), each pair of integers (i, j) such as $0 \leq i \leq j \leq n$ is called a *range* of w and is denoted $\langle i..j \rangle_w$ or $i..j$ if w can be omitted, i and j being respectively the *lower* and *upper bound* of the range, and $j - i$ its size. If $i = j$, the range is *empty*. Two ranges are equal if and only if their lower and upper bounds are respectively equal³. A range $\langle i..j \rangle_w$ corresponds to a substring of w , namely $a_{i+1} \dots a_j$. The concatenation of two ranges $i..j$ and $k..l$ is defined if and only if $j = k$ and is in this case the range $i..l$.

Variable symbols and terminal symbols denote ranges. A terminal symbol t denotes a range of length 1 corresponding to a substring of w that is the symbol t . The *concatenation* XY of two variable or terminal symbols X and Y ($X, Y \in V \cup T$) denotes the range resulting from the concatenation of the ranges denoted respectively by X and Y , and is therefore defined if and only if these ranges can be concatenated. We often (abusively) denote by *the range* X , where X is a variable symbol, the range that is denoted by X .

Given $w \in T^*$, we call *w-instantiated predicate* or simply *instantiated predicate* a predicate in which all variable and terminal symbols have been replaced by ranges of w , and all ranges of the same argument of the same predicate have then been replaced by the range resulting from their concatenation. If this is possible, the predicate is said to be *instantiable* by w . For example, if the length of w is 3 ($w = a_1 a_2 a_3$), a possible instantiation for the predicate $A(XY, a_3, a_2 Y)$ is $A(0..3, 2..3, 1..3)$. We define in the same way *instantiated clauses*. The arguments of a predicate (but not different variables of the same argument) can of course be replaced by discontinuous, or even overlapping ranges, since the same variable can occur in several arguments of several predicates of a clause. This is denoted as the *non-linearity* of RCGs, and allows to express different points of view or properties on a given range that interact with each other. As we will see below, this is a major advantage of RCGs for linguistic use. More generally, it is because of this non-linearity that RCGs have the necessary expressive power to cover all *PTIME*.

For a given positive RCG G and an input string w , a binary *derive* relation, denoted by $\xRightarrow{G, w}$, and having as operands sets of instantiated predicates, by the following. Let $\Gamma_1 \gamma \Gamma_2$ be an instantiated right-hand side of some clause (and thus a set of predicates, as said before), where γ is also the left-hand side of the instantiated clause $\gamma \rightarrow \Gamma$. We have then $\Gamma_1 \gamma \Gamma_2 \xRightarrow{G, w} \Gamma_1 \Gamma \Gamma_2$.

A string $w \in T^*$ of length n is recognized by the grammar G if and only an empty list of predicates can be derived from the instantiated predicate $S(0..n)$

³ Therefore, if $i_1 \neq i_2$, the ranges $i_1..i_1$ and $i_2..i_2$, while both empty, are not equal.

(S being the axiom), i.e. if and only if $S(0..n) \xrightarrow[G,w]{+} \varepsilon$, the binary relation $\xrightarrow[G,w]{+}$ being the transitive closure of $\xrightarrow[G,w]$.

For example, let us consider the non semi-linear language $L = \{a^{2^p} \mid p \geq 0\}$. The following positive RCG recognizes this language:

$$\begin{aligned} S(XY) &\rightarrow S(X) \text{ EQ}(X, Y) \\ S(a) &\rightarrow \varepsilon \\ \text{EQ}(aX, aY) &\rightarrow \text{EQ}(X, Y) \\ \text{EQ}(\varepsilon, \varepsilon) &\rightarrow \varepsilon \end{aligned}$$

Indeed, this grammar recognizes the input string a thanks to the second clause. And an input string consisting of 2^p times a ($p \geq 1$) is decomposed by the first clause in two ranges that must have the same length (predicate EQ^4) and denote a (same) substring that has to be in L as well (of length 2^{p-1}).

2.2 Negative RCGs

Positive RCGs cover *PTIME*. Therefore, the set of languages that can be recognized by a positive RCG is closed under complementation. For this reason, it is only for practical reasons, and not to increase the expressing power of the formalism, that negative predicate can be introduced.

Indeed, we call *negative predicate* a predicate marked as such, either by a bar over the predicate ($\overline{A(\dots)}$), or by the symbol $!$ in front of it ($!A(\dots)$), the intended meaning being based on "negation by failure": the empty list of instantiated predicates can be derived from an instantiated negative predicate if and only if it can not be derived from its positive counterpart.

We call negative RCG an RCG that has at least one clause containing in its right-hand part at least one negative predicate. A negative RCG is consistent if, for any $w \in T^*$, there is no w -instantiated predicate $A(\dots)$ such that the empty list of predicates can be derived from both $A(\dots)$ and $!A(\dots)$.

2.3 Closure Properties

It can be shown easily [3] that RCLs are closed under union, concatenation, Kleene iteration, and, more interestingly, intersection and complementation. The grammars recognizing the operand languages need not be modified. One or two more suffices. In a rather informal way, and with S_1 and S_2 being the axioms of RCGs recognizing respectively the languages L_1 and L_2 , we can get the closures by adding the following clauses, S being the axiom of the resulting language:

⁴ The notation ε groups in a rather confusing way two different things, as visible in the last clause of the grammar given as example: it can denote either an empty string, i.e. an element of T^* , or an empty list (or set) of predicates.

Union	$S(X) \rightarrow S_1(X)$ $S(X) \rightarrow S_2(X)$
Concatenation	$S(XY) \rightarrow S_1(X) S_2(Y)$
Intersection	$S(X) \rightarrow S_1(X) S_2(X)$
Kleene iteration	$S(\varepsilon) \rightarrow \varepsilon$ $S(XY) \rightarrow S_1(X) S(Y)$
Complementation	$S(X) \rightarrow !S_1(X)$

2.4 Parsing

Range Concatenation Languages can be recognized and analysed in polynomial time. More precisely, let $|G|$ be the *size* of the k -RCG G , defined as the sum of the number of right-hand side predicates over all its clauses, and l the maximum number of right-hand side predicates in the longest clause. In [3], Boullier gives an algorithm that is $O(|G|n^{2k(1+l)})$ in time.

Moreover, Boullier has developed an efficient RCG-parser that has been already used to build TAG and MC-TAG parsers after an appropriate conversion step, with excellent efficiency results [7].

3 Introducing Meta-RCGs

Range Concatenation Grammars are a powerful though efficient formalism that can be seen as logic programming on ranges of the input string. For this reason, and for others (see [5]), it is a suitable basis to develop a linguistic formalism, but is not satisfying as such. Indeed, a linguistic formalism is almost always twofold, since on the one hand it builds the structure of the sentence, and on the other hand it computes features on this structure. In most formalisms, such as TAGs, LFGs, HPSGs, Dependency Grammars or Categorical Grammars, these two aspects are processed with different operators or even different mechanisms, not necessary simultaneous. In some cases, this leads to imprecise or excessive expressive power for the resulting formalism, and, if the separation between these two aspects is too strict, this also leads to problems concerning, for example, combinatorial explosion, error recovery algorithms, or automatic learning (dealing with unknown words).

Moreover, the limit between the structural backbone (often purely syntactic) and the features computed over it (often referred to as *decorations*), is hardly linguistically justified. Its position is not precisely and uniquely defined: a given formalism or implementation of a formalism can implement a linguistic property in the backbone, an other one treating it as a feature. On the other side, the linearity of most formalisms, i.e. their non-ability to reuse several times the same range of the sentence, makes it impossible to include additional information inside the grammar, such as lexical semantic constraints, in a satisfying way (i.e. not only as a limited set of "semantic" features that are not avoidable for linguistic reasons, but really as a complete set of semantic predicates).

For all these reasons, it seems appropriate to implement, if it is possible, all linguistically-motivated "decorations" inside the grammatical formalism that describes the structural backbone. Furthermore, we make the hypothesis that parsing natural language is possible in a polynomial time⁵. Thus, our linguistic formalism can be seen as decorated RCGs that can be compiled into pure RCGs. In the remaining of this section, we shall define our formalism, called Meta-RCGs (or MRCG), which defines over RCGs these decorations and the way they can be converted into RCG predicates and/or arguments. The borderline between the structure and the decorations is defined precisely by the following: a decoration is a property of (a portion of) an analysis, and the structure retains all properties of ranges.

The main idea underlying this approach is that the non-linearity of RCGs allows to treat as structural predicates (and not as decorations) several different linguistic facts over the same ranges. For example, syntactic and lexical semantic facts are used simultaneously. However, in a linguistic grammar, the analysis of a given range of the input string can be ambiguous. Therefore, the use we make of non-linearity has to be able to guarantee that all facts expressed in a global analysis about a given range are compatible: we have to prevent the apparition of analyses where such a range is analysed in a first way in one part of the global analysis, and in an other incompatible way in an other part of the analysis. However, it is not possible in a polynomial way to label the complete analysis of a range for later identification and re-use. Therefore, it is necessary to define a polynomial amount of information that will be exported by ranges to other parts of the analysis. This is what is done in our formalism, where this amount of information regroups the *heads* of a syntagm and *features* (decorations) associated to it, which will be made accessible at different parts of the analysis thanks to *contexts*. The remainder of the analysis of this syntagm will be inaccessible from the "outside".

Let $G_{RCG} = (N, T, V, P, S)$ be a classic RCG, as defined above. We will define an associated Meta-RCG (hereafter MRCG) G_{MRCG} extending G_{RCG} (the extension concerns grammars, not necessarily associated languages). For this, we will first go through preliminary remarks and new definitions.

3.1 Heads

While controversial by many aspects, the notion of *head of a syntagm* is widespread in linguistic literature, and has been intensively used by many grammatical frameworks, such as HPSG [6], but also LFG, Dependency Grammars, and others. We introduce heads (and coordinating items separating them) by ex-

⁵ Even the parsing of the *syntax only* of natural language probably needs the expressive power of RCGs, i.e. all *PTIME*. Indeed, it can be shown (see for example [5] and references therein) that some specific phenomena in some languages are beyond the expressive power of Mildly Context-Sensitive languages, and thus by formalisms as powerful as LCFRS (e.g. Chinese numbers, genitives in old Georgian, scrambling in German, or multiple verbs coordinates in Dutch).

tending the notion of argument in the following way: we call *MRCG-argument*, or more simply *argument* one of the following items:

- an (RCG-)argument, i.e. as said before the concatenation of elements of $V \cup T$,
- a *head-coordination argument* or *hc-argument*, i.e. an element of V followed by the operator $^{\wedge}$ or the operator $^!$,
- a *single-head argument*, or *sc-argument*, i.e. an element of V followed by the operator $^+$,
- a *head-adding argument* or *ha-argument*, i.e. an argument of the form $V_1^+ V_2^+ V_3^{\wedge}$.

An argument that is not a classical RCG-argument is called a *syntagmatic argument*. The intended meaning is the following. If **Syntagm** is a range denoting a syntagm, **Syntagm** $^{\wedge}$ denotes a pair of lists, the first one being the list of its heads and the second one the list of the coordinating items that are between them. Hence, if the substring corresponding to **Syntagm** is "an apple or a pear", a reasonable grammar will give an analysis such as **Syntagm** $^{\wedge}$ includes a list of heads covering "apple" and "pear" and a list of coordination items covering "or". The argument **Syntagm** $^!$ is the same thing, but with a number of heads (respectively coordination items) that is exactly 1 (respectively 0). A single-head argument V^+ creates a syntagmatic argument made out of a list of heads containing only one element, V (thus, it has to be of length 1), and an empty list of coordinating items. Finally, a ha-argument $V_1^+ V_2^+ V_3^{\wedge}$ is a syntagmatic argument whose heads list is the concatenation of V_1 and the heads list of V_3^{\wedge} , and whose coordination items list is the concatenation of V_3 and the coordination items list of V_3^{\wedge} .

For example, and for illustration purposes only, a clause analysing (recursively) as a nominal group⁶ a simple coordination of basic nominal groups could look like the following:

$$\begin{aligned} & \text{NP}(\text{Det Head Coord Np2, Head}^+ \text{Coord}^+ \text{Np2}^{\wedge}) \\ \rightarrow & \text{NOUN}(\text{Head}) \text{DET}(\text{Det, Head}) \text{COORD}(\text{Coord}) \text{NP}(\text{Np2, Np2}^{\wedge}). \end{aligned}$$

3.2 Features and Homonym Numbers

A *feature* (or *attribute*) can be defined as a (finite) vector $\mathbf{F} = (f_1, \dots, f_n)$. A *constant value* of the feature \mathbf{F} is an element of \mathbf{F} . A *variable value* of the feature \mathbf{F} is the concatenation of the operator $\$$ and an element of a set of variable feature-values symbol, V_f (for example, if $\mathbf{g} \in V_f$, $\$ \mathbf{g}$ is a valid variable value for \mathbf{F}). We call *features list* a list of feature names. An *attribute-value pair*, or *av-pair*, is of the form $\mathbf{F} = v$, where \mathbf{F} is the name of a feature and v is a constant or variable value of \mathbf{F} . Finally, an *attribute-value pairs list*, or *av-list*, is a sequence of av-pairs in which an attribute can appear at most once in an av-pair.

We call homonym number a vector similar to a feature, defined as $\mathbf{HN} = (0, \dots, h_{max})$. First, we define a *terminal argument* as an argument that has

⁶ We do not use here the phrase *noun phrase* for a reason that will be explained later.

at most one variable, and that denotes a range of maximal length 1. The role of a homonym number is then to associate some terminal arguments of some predicates a special number that allows to distinguish between two homonymous terminals (i.e. words). For each predicate, a special function *homonymous-args* gives the list of the positions of its arguments that have such an homonym number. But these numbers are not apparent in MRCG-clauses.

3.3 Contexts

We define a *contextual item* as an element **Ctxt** of the set V of variable symbols possibly followed by the operator / or by the operator \wedge , and possibly followed by the operator : and a features list. The role of contextual items is to modelize long-distance dependencies. For this reason, and although declarative as our whole formalism, the intended meaning of contexts is more easily described with an operational point of view. Long distance dependencies will be treated in 2 steps that share common points with the SLASH feature of HPSG: at one point of the analysis, a contextual item is built out of the concerned syntagm and "pushed" into the predicate-dependant *context* of the concerned predicate, **A**. All predicates related to **A** that can accept this context and for which no new value is explicitly given will inherit this value, thus transporting the context to other parts of the analysis. At some (arbitrarily distant) other point of the analysis, this contextual item, its heads and/or its features can be put into arguments (or "dropped", or "popped") and used.

Hence both operators, with the following meanings: the / operator means that the features list associated to the range **Ctxt** (this features list is in this case mandatory) is pushed into the context, but not **Ctxt** itself. The operator \wedge means that the range **Ctxt** is a syntagm, and that its heads and coordination items, i.e. **Ctxt** \wedge , have also to be pushed along with the range **Ctxt** in the context.

The percolation of contextual items from one point of the analysis to another is allowed by a special function that needs to be defined, called *context-of*. This function is defined over the set N of predicate names and, for a given predicate, returns a list of contextual items. The meaning of this function is the following: given a predicate name **A**, the contextual items in *context-of*(**A**) are associated to all occurrences of **A** predicates. Suppose that, in a given clause, the same contextual item **Ctxt** (or its heads, or some of its associated features) is associated to more than one predicate. All such predicates that are on the right-hand side and that explicitly re-define the value of **Ctxt** will get this value. And all such predicates that are on the right-hand side and that do not re-define the value of **Ctxt** will share its value with the one of the left-hand side predicate. On the left-hand side, the value of **Ctxt** (or of **Ctxt** \wedge or of some of its features) can be equated to a real range (or to a feature value, if appropriate). For example, if a contextual item **Ctxt** is associated to both predicate names **A** and **B**, and if no specific equation redefines a new value of **Ctxt** for predicate **B**, then a clause such as **A**(**X**) \rightarrow **B**(**X**) will ensure that the contextual item denoted by **Ctxt** is passed to the predicate **B** in the same way as the range denoted by **X**. We define only one operation on contexts, denoted by the operator =. When applied to a contextual

range or contextual heads/coordinations, it is a range-equality operator. When applied to contextual features, it is a value-equality operator. The feature **F** of a contextual item **Ctxt** is accessible through a special dot operator with the following syntax: **Ctxt.F**. The distinction between a "push" and a "drop" (or "pop") lies only in the fact that the = operation is done on the left-hand side predicate ("drop") or on a right-hand side predicate ("push"), a direct use of **Ctxt** as a standard range in the right-hand side corresponding also to a "drop".

Finally, we call *context-value pair*, *cv-pair* or *contextual equation*, an expression of the form **Ctxt=Range** or **Ctxt.F=v**. We will see examples when we have defined the complete MRCG syntax.

3.4 MRCG

MRCG clauses are an extension of RCG clauses, since predicates are replaced by meta-RCG predicates of the following form:

$$A(\alpha_1, \dots, \alpha_i, \dots, \alpha_p) [\phi_1 \dots \phi_j \dots \phi_q] \{\kappa_1 \dots \kappa_k \dots \kappa_r\},$$

where α_i are MRCG-arguments, ϕ_j are av-pairs (equations on features), and κ_k are cv-pairs (contextual equations). The features part and the contextual part are facultative. For example, using **LDDep** as a shortcut for "long-distance dependency syntagm", here is a valid MRCG-clause, assuming that *context-of(NP)* = (**LDDep**[^]:**number**,**case**):

$$\begin{aligned} &VP(Verb, Verb^+) \{LDDep.case=accusative\} \rightarrow \\ &\quad VERB(Verb) \\ &OBJ(LDDep, LDDep^, Verb) [number=LDDep.number] \{LDDep=0\}, \end{aligned}$$

where **LDDep=0** is syntactic sugar for "LDDep is a 0-length range". The meaning of such a clause is the following: we can build with the range **Verb** a **VP** in a context where we have a long distance dependency **LDDep** with an accusative case if **Verb** is a verb, and if we can make of **LDDep** the direct object of this verb (this object having the same number than the one of **LDDep**), in a context where no long distance dependency is available any more for use.

3.5 Conversion from MRCG to RCG

As said before, the MRCG formalism can be converted into a strongly-equivalent RCG. This allows to use Boullier's efficient RCG parser, but also to avoid problems explained in the first paragraphs of this section that arise when the analysing process is decomposed in more than one step. However, the details of this conversion are not very interesting, and will not be presented here. We have designed and realized such a converter, that we call *MRCG-compiler*. As a side effect, this compiler produces an information file about the conversion that can be used to rebuild the MRCG-analysis of a sentence from the RCG-analysis given by the RCG parser.

3.6 Grammar and Lexicon

An important part of the design of a grammatical formalism for natural language is the design of the interface between the grammar and the lexicon. Although it

is possible, there is no reason to limit an RCG to be lexicalized, i.e. to include in every clause at least one left-hand side argument that has at least one terminal symbol. However, a lot of information is given by the terminals, here words⁷, that has to be both represented and used in a way which, as for all parts of the formalism, has to be both computationally efficient and linguistically acceptable.

Several options could be thought of, taking advantage of the properties of closure of RCGs:

1. Compile a huge grammar with as many terminals as there are different inflected forms in the language, leading to an enormous grammar,
2. Compile separately the non-lexicalized part of the grammar and one or more lexical modules with only lexicalized clauses, either with the RCG parser generator or with a specific module that profits from the hierarchical structure of lexical information, uses appropriate algorithms to deal with the enormous amount of inflected forms and that is able to append the RCG parse forest with consistent sub-forests.
3. Compile the non-lexicalized part of the grammar, and, for each sentence, generate and compile dynamically the set of lexicalized clauses involving terminals present in the input sentence.

Independently of this choice, lexicalized clauses can be represented either as such, or be computable as the result of an inheritance process inside an ontology.

4 Parsing French with Meta-RCGs: Two Examples

As said before, the aim of the Meta-RCG formalism is to allow the design of grammars that take into account at the same time and with the same technical status morphological, syntactical and (lexical) semantics information. This is achieved by different predicates, thanks to the non-linearity of RCGs and hence Meta-RCGs.

We have developed, and still do, a large-coverage grammar for French language in the Meta-RCG formalism. To show how this formalism can be used for what it has been designed for, we will show how two sentences are analysed. The first sentence, *Un avocat mange un avocat* ("A lawyer eats an avocado"), shows how homonym numbers and lexical semantics work together to fully disambiguate ambiguous inflected forms. The second sentence, *Pierre veut une bière et dormir* ("Pierre wants a beer and [wants to] go to bed"), is an example of heterogeneous coordination (between a noun phrase and an infinitive verb phrase) and of subject control verb. Of course, both for place and complexity reasons, the whole analyses will not be shown, but only simplified parts of them to present the involved mechanisms.

⁷ Since we want to include lexical semantic properties inside the grammar, the traditional approach of a lot of formalisms, which associate to all words a *category*, whose value is used as terminal symbol in the grammar, is not satisfactory. It would lose too much information or would require such a big amount of categories that there would not be any advantage over the direct use of words as terminals.

In the following, instantiated MRCG-clauses will respect the following convention: a range $\langle i..j \rangle_w$ covering the substring s will be represented as $s_{i..j}$. If this range has an homonym number h , it will be represented as $s_{i..j}^h$. If it has heads, the corresponding terminals will be underlined, and its homonym number will be indicated as an exponent.

4.1 Lexical Ambiguity and Semantics: *Un Avocat Mange un Avocat*

Let us consider the following sentence:

- (1) Un avocat mange un avocat
A lawyer eats an avocado

This sentence is syntactically extremely simple. We use it as an example only to give an insight into the top-level clauses of our grammar and to show the role of homonym numbers. Moreover, grammar rules will be simplified. In particular, predicates and clauses dealing with non-existent topological places (e.g. sentence modifiers, adverbs, clitics, long-distance dependencies and so on) will be ignored. Finally, for space reasons, features will not be displayed, except when absolutely necessary, and terminals are abbreviated by their first characters followed by a dot when necessary. Some parts of the analysis, that are not very important for its global understanding, are replaced by textual descriptions printed in *italics*. This being said, here is how our grammar analyses this sentence (VK stands for *verbal kernel*, VC for *verbal complex*, and *dth=act* resp. *pass* for *diathesis=active* resp. *passive*):

```

PHRASE(un av. mange un av.0..5) → VSEM_IND(mange2..30)
                                     PHRASE2(un av. mange2..30 un av.0..5).
VSEM_IND(mange2..30) → VERBE(mange2..30)
                                     LEX(mange2..30) [mode=Ind] .
VERBE(mange2..30) → .
LEX(mange2..30) [mode=Ind] → .
PHRASE2(un av. mange2..30 un av.0..5) → SUBJECT(un avocat1..2, mange2..30)
                                     VP(mange2..30 un avocat2..5) {Subj=un av.1..20}.
VP(mange2..30 un avocat2..5) → VK(un avocat2..5, un avocat2..5).
VK(mange2..30, un avocat2..5) → VC(mange2..30, 3..3, 3..3)
                                     ROLES(2..2, 2..2, 2..2, un av.2..5, mange2..30).
ROLES(2..2, 2..2, 2..2, un av.2..5, m.2..30) → OBJECT(un avocat3..5, m.2..30)
                                     ROLES(un avocat3..5, 2..2, 2..2, 5..5, m.2..30).
```

Before going on with the remainder of the analysis, it is necessary to clarify the meaning of the arguments of the predicate ROLES. In fact, the 3 first arguments (they are 5 in the real grammar) denotes the syntactico-semantic roles associated with the verb. The first role is the accusative one, the second role is the dative one, and the last one is the genitive one. The recursive predicate ROLES "eats" at each call a complement in the list of not-yet-parsed complements (4th argument), analyses it, and, if it fills a not-yet-filled role, fills the corresponding argument of the right-hand side call of ROLES with it. When the range of not-yet-parsed

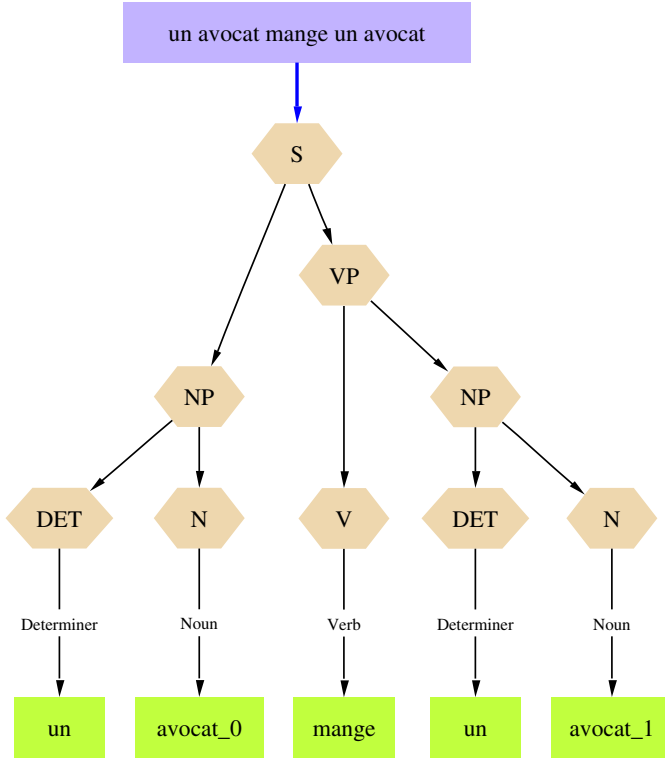


Fig. 1. Constituency view of the analysis of sentence (1)

arguments is empty, roles filling is completed, and verifications can be done about mandatory or impossible roles. This being said, here is how the analysis goes on:

$\text{ROLES}(\text{un } \underline{\text{avocat}}_{3..5}^1, 2..2, 2..2, 5..5, m, \text{ }_{2..3}^0) \rightarrow \text{True because } \text{mange}^0 \text{ is a transitive verb and the first argument of ROLES, i.e. the accusative role, is filled by } \text{un } \underline{\text{avocat}}_{3..5}^1$
 $\text{SUBJECT}(\text{un } \underline{\text{avocat}}_{0..2}^0, \text{mange}_{2..3}^0) \rightarrow \text{SUBJ_SEM}(\text{avocat}_{1..2}^0, \text{mange}_{2..3}^0) \text{ NP}(\text{un } \underline{\text{avocat}}_{0..2}^0).$
 $\text{SUBJ_SEM}(\text{avocat}_{1..2}^0, \text{mange}_{2..3}^0) [\text{d.}=\text{act}] \rightarrow \text{AGENT}(\text{avocat}_{1..2}^0, \text{mange}_{2..3}^0)$
 $\text{AGENT}(\text{avocat}_{1..2}^0, \text{mange}_{2..3}^0) \rightarrow \text{ANIMATED}(\text{avocat}_{1..2}^0).$
 $\text{OBJECT}(\text{un } \underline{\text{avocat}}_{3..5}^1, \text{mange}_{2..3}^0) \rightarrow \text{OBJ_SEM}(\text{avocat}_{4..5}^1, \text{mange}_{2..3}^0) [\text{dth}=\text{pass}] \text{ NP}(\text{un } \underline{\text{avocat}}_{3..5}^1).$
 $\text{OBJ_SEM}(\text{avocat}_{4..5}^1, \text{mange}_{2..3}^0) [\text{d.}=\text{pas}] \rightarrow \text{PATIENT}(\text{avocat}_{4..5}^1, \text{mange}_{2..3}^0)$
 $\text{PATIENT}(\text{avocat}_{4..5}^1, \text{mange}_{2..3}^0) \rightarrow \text{EDIBLE}(\text{avocat}_{4..5}^1).$
 $\text{NP}(\text{un } \underline{\text{avocat}}_{0..2}^0) \rightarrow \text{True because } \text{un } \underline{\text{avocat}}_{1..2}^0 \text{ is a valid noun phrase}$
 $\text{NP}(\text{un } \underline{\text{avocat}}_{3..5}^1) \rightarrow \text{idem}$

As can be seen, the process can be summed up as follows:

- Identify the semantic part of the main verb (the past participle if there is one or more auxiliaries), and make of it the head of the sentence,
- Identify the whole verbal kernel (verbal components, clitics, and adverbs that are inbetween),
- Analyse the subject, which is a noun phrase (NP clause ; noun phrases can be infinitives or propositions) and a semantic argument of the verb (SUBJ_SEM clause), and put it in the context (not used in this sentence),
- Identify one after the other all post-verbal complements (here only one), and analyse it both as a noun phrase (NP clause) and as a semantic argument of the verb (*_SEM clauses).

Of course, this simplified presentation doesn't explain how are treated attributes of the subject or of a complement, clitics, long-distance dependencies, modifiers, relatives, and so on. It is only the basic skeleton of the grammar.

For illustration purposes, we give in Figure 1 the constituency tree extracted automatically from the global analysis.

4.2 Heterogeneous Coordination and Control Verb: *Pierre Veut Une Bière et Dormir*

Let us consider the following sentence:

- (2) Pierre veut une bière et dormir
 Pierre wants a beer and [wants to] sleep

As said before, this sentence exemplifies two phenomena: heterogeneous coordination between a nominal syntagm (*une bière*) and an infinitive (*dormir*), and subject control verb (*veut*, which controls the subject *Pierre* for the infinitive *dormir*). We will not give the whole analysis of this sentence, but only the most interesting parts.

First, we will see how the heterogeneous coordination is treated. In fact, we make the distinction between a noun phrase and a nominal group. For us, a *noun phrase* is a phrase that plays the role which is canonically fulfilled by a phrase built around a noun. Such a phrase is a *nominal group*. Thus, a nominal group, an infinitive or a proposition are noun phrases. This distinction makes it possible to explain several facts, including the fact that an infinitive and a nominal group can indeed be coordinated, as in (2). It can also deal with the fact that, for example, several verbs accept an infinitive as a direct object. Predicates such as OBJ_SEM can nevertheless constraint the object of a verb to be an infinitive, or on the contrary prevent it from being an infinitive. But in our grammar, this is a matter of constraints over the object, and not a fundamental difference between different kinds of syntagms, at least at the level of the *object* relation. Hence the following analysis for this relation, in which contexts are not shown, since they will be useful and thus shown later (NG stands for *nominal group*):

OBJECT(une bière_{3..4} et dormir_{5..6}) → OBJ_SEM(bière_{3..4}⁰, veut_{1..2}⁰)
 OBJ_SEM(dormir_{5..6}⁰, veut_{1..2}⁰)
 NP(une bière_{3..4}⁰ et dormir_{5..6}⁰) .

OBJ_SEM(bière_{3..4}⁰, veut_{1..2}⁰) → NOUN(bière_{3..4}⁰) .

OBJ_SEM(dormir_{5..6}⁰, veut_{1..2}⁰) → [See below]

NP(une bière_{3..4}⁰ et dormir_{5..6}⁰) → NP(une bière_{2..4}⁰)
 NP(dormir_{5..6}⁰)
 COORD(et_{4..5}⁰)
 [other predicates
 for features processing] .

NP(une bière_{2..4}⁰) → NG(une bière_{2..4}⁰) .

NG(une bière_{2..4}⁰) → [Standard analysis for a nominal group]

NP(dormir_{5..6}⁰) → INFINITIVE(dormir_{5..6}⁰) .

INFINITIVE(dormir_{5..6}⁰) → [See below]

In our grammar, any noun phrase can potentially be an infinitive. The fact that *veut* is a subject control verb has only one impact: it allows the predicate $\text{OBJ_SEM}(\text{dormir}_{5.6}^0, \text{veut}_{1..2}^0)$ to be true. Thus, the noun phrase that is the object of *veut*⁰ can really be an infinitive. The analysis of this infinitive, here *dormir*, uses the contextual item **Subject** in the following way. As for the sentence studied in the previous paragraph, the analysis of the subject, here Pierre_{0..1}⁰ (see the **PHRASE2**-clause in the previous example), "pushes" this syntagm in a contextual item named **Subj**. Then, the predicate **INFINITIVE**, which

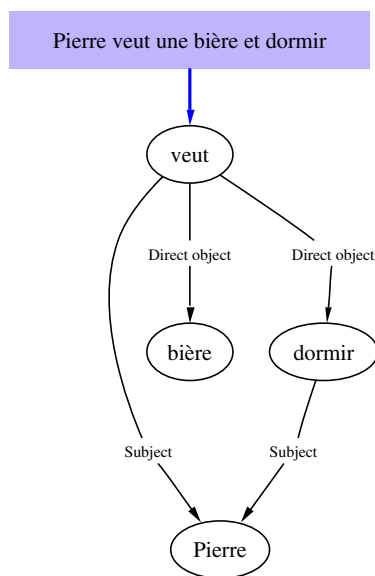


Fig. 2. Dependency view of the analysis of sentence (2)

inherits this context through respectively VP, VK, ROLES, OBJECT and NP, uses it to build the syntactico-semantic dependency thanks to the following clause (VK_INF stands for *verbal kernel of an infinitive*, and the context of INFINITIVE is now shown):

$$\text{INFINITIVE}(\underline{\text{dormir}}_{5..6}^0) \{ \text{Subj}=\underline{\text{Pierre}}_{0..1}^0 \text{ Subj.gender=masc Subj.number=sing} \} \\ \rightarrow \text{VK_INF}(\underline{\text{dormir}}_{5..6}^0, \underline{\text{Pierre}}_{0..1}^0) [\text{gender=masc number=sing}] .$$

For illustration purposes, we give in Figure 2 the dependency graph extracted automatically from the global analysis.

5 Conclusion

We have presented a new formalism, called *Meta-RCG*, and based on Range Concatenation Grammars. This formalism, thanks to the non-linearity of RCGs, allows the development of grammars that implement linguistic facts as predicates over ranges of the input sentence. These facts can deal for example with morphology, syntax, lexical semantics, combinaisons thereof. The non-linearity makes it unnecessary to put in costly and numerous features all the linguistic information that doesn't fit an insufficient backbone. Moreover, the very concept of predicates over ranges of the input string seems very intuitive from a linguistic point of view.

We have shown thanks to two small examples the way linguistic Meta-RCGs can be developed. As said before, we are currently developing such a grammar for French. The development is already quite advanced, and we can deal with phenomena like subject, object or indirect-object control verbs, raising verbs, infinitives, completives (either modifiers or arguments), light verbs, homogeneous and heterogeneous coordination, participial modifiers, attributes of the subject or of the object, auxiliaries, arguments of nouns ("Peter's departure") or adjectives, negation (which can be discontinuous in French), relatives (including relatives in "dont" that can modify the subject or the object of an arbitrarily deep verb inside the relative) and many other less complicated phenomena. Parsing times are very satisfying⁸. We are currently using the EUROTRA corpus of French sentences [8], which gives a good set of simple to very complicated sentences: we proceed sentence by sentence in an exhaustive manner, and modify the grammar so as to get only the appropriate parse (or the appropriate parses if the sentence is really ambiguous). We have currently reached the second half of the file, but we have already implemented several phenomena that appear later on in this corpus. Moreover, we have developed filters that allow to project our analysis

⁸ We use a "benchmark-sentence" which has simultaneously a relative in "dont" that modifies the coordinated object of a verb of the relative, with a coordinated subject, and which is an object-controlled verb inside a completive. Parsing time, depending on the state of the grammar, has oscillated in the last months between 0.4 and 2 seconds. This sentence is the following: *Paul aime la Normandie dont je sais que Pierre regarde Marie et Paul mange une pomme et une poire verte.*

into different views, including (for now) a constituency tree, a dependency graph, topological boxes, and predicate-arguments semantics.

For all these reasons, we believe that we have designed a formalism that virtually satisfies the constraints given in the introduction of this paper to characterize an interesting formalism. Further work includes the continuation of the extension of our grammar for French thanks to this EUROTRA corpus, a more precise definition of the abstract foundations of our formalism, an in-depth linguistic analysis of its properties and of the grammars it allows to write, and a more precise definition of the interface between lexicon and grammar. It also includes the extension of our grammar to other components of the linguistic analysis of a sentence, such as discourse analysis or Montague-like semantics.

References

1. Dahl, V., Tarau, P. and Huang Y.-N.: Datalog Grammars. In: GULP-PRODE 2 (1994) 268–282
2. Blache, P.: Parsing with Constraint Graphs: a Flexible Representation for Robust Parsing. In Di Sciullo, A.M., ed.: *Grammars and NLP LNCS*, Springer-Verlag (2001)
3. Boullier, P.: Range Concatenation Grammars. In Bunt, H., Carroll, J., Satta, G., ed.: *New developments in parsing technology*. Kluwer Academic Publishers (2004) 269–289
4. Boullier, P.: Counting with range concatenation grammars. *Theoretical Computer Science* **293** (2003) 391–416
5. Sagot, B., Boullier, P.: Les RCG comme formalisme grammatical pour la linguistique. In: *Proceedings of TALN '04, Fez, Marocco* (2004) 403–412
6. Pollard, C., Sag, I.A.: *Head-Driven Phrase Structure Grammar*, University of Chicago Press and CSLI Publications (1994)
7. Barthélémy, F., Boullier, P., Deschamp, P., Villemonte de La Clergerie, É.: Guided Parsing of Range Concatenation Languages. In: *Proceedings of ACL '01, Toulouse, France* (2001) 42–49
8. Danlos, L., Laurens, O.: *Présentation du Projet Eurotra et des grammaires d'Eurotra-France*. Technical Report n 1, Université Paris 7 - Talana/LISH (1991)

How to Build Argumental Graphs Using TAG Shared Forest: A View from Control Verbs Problematic

Djamé Seddah¹ and Bertrand Gaiffe²

¹ Loria, Nancy

`djame.seddah@loria.fr`

² Loria/Inria Lorraine, Nancy

`bertrand.gaiffe@loria.fr`

B.P.239 -F 54506 Vandoeuvre-lès-Nancy, France

Abstract. The aim of this paper is to describe an approach to semantic representation in the Lexicalized Tree Adjoining Grammars (LTAG)[1] paradigm. We show how to use all the informations contained in the two representation structures provided by the LTAG formalism in order to provide a dependency graph.

1 Introduction

A LTAG grammar consists of a large lexicon that associates each lemma to a set of elementary trees. Each elementary tree has one “main” anchor (instantiated by a lemma) which is generally a minimal semantic unit. Each tree reflects the argument structure of the main anchor. The tree’s leaf nodes must describe the logical arguments of the predicate implied by the anchor. TAG provides two operations to manipulate elementary trees : adjunction and substitution. The substitution operation corresponds to context free derivation : an *initial tree* with a root labeled “X” can be substituted in a substitution node of the same category (Substitution nodes have to be leaves of elementary trees). The adjunction operation gives TAG additional power over Context Free grammar. It is optional and recursive. This operation inserts an *auxiliary tree* into another tree. The inserted tree must have a *foot node* which has the same category of the root node and of the insertion node.

All elementary trees are extended projections of lexical items and contain all syntactic arguments of the lexical anchors. In fact, when a given LTAG grammar follows principles[2] such as the predicate-argument co-occurrence principle and minimal semantic unit principle, the syntactic arguments also correspond to semantic arguments. Moreover the LTAG framework provides two kinds of representations : a derivation tree which strictly records the resulting operation from the parsing of a given sentence, and a derived tree which is the final syntactic representation of the analysis. Each node of the derivation tree is labeled

by a Gorn address¹ telling us where the operation took place in the dominating tree.

Let us consider the following sentence :

- (1) Jean aime beaucoup Marie
JEAN LOVES MUCH MARIE

A toy grammar that analyzes this sentence is shown in figure 1 and the resulting derived and derivation tree on the figure below. Each substitution node is labeled as X_j , where X corresponds to its category and j to its argumental position within its respective tree.

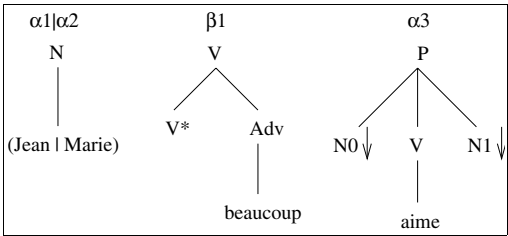


Fig. 1. Elementary trees

We see in figure 2 that the derivation tree² mirrors the predicate-argument structure.

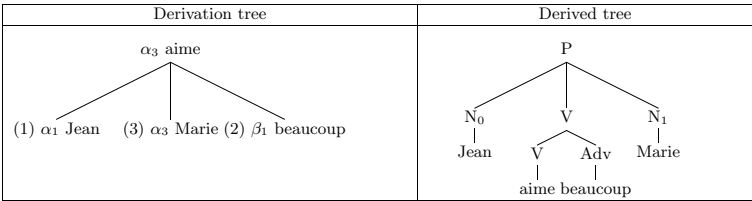


Fig. 2. Derivation tree and derived tree of “Jean aime beaucoup Marie”

In an ideal world it would be possible to work from this structure to build a compositional semantic representation in the spirit of Montague grammar. In

¹ the root has the address 0, the i^{th} child of the root has address i and for all other nodes : the i^{th} child of the nod with address j has address $j.i$
² For the sake of simplicity, each tree whose name begin by β , resp α , is governed by adjunctions, resp. substitution.

fact, the LTAG formalism was conceived to make the interface between semantic and syntax easy to manipulate such that one should only have to associate λ -terms to initial trees to obtain a logical formulae or a correct predicative structure³. The derivation tree would simply guide the application operation on λ -terms, so that we get at its root the result of our λ -term computation. If we base our argumentation upon the properties of substitution nodes, their mandatory completion, we could see these node as the exact counterparts of argumental variables. The main properties of adjunction are that it is non predictable and optional, so we have to consider the tree where the adjunction took place as an argument of the function associated to the auxiliary tree. Thus the derivation tree seen in fig 2 could be interpreted as follows⁴ with simplified λ -terms :

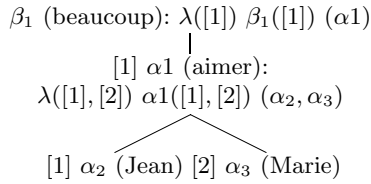


Fig. 3. Syntax-semantic interface in an ideal world

The problem with this kind of syntax-semantic interface is it only works on a very small subset of language, only for simple predicative structure : 1) because there is no distinction being made between predicative and modifier adjunction (the first one inverses the dependency arc between two trees), 2) because there is no any proper mechanism to resolve scope ambiguities and, 3) because the semantic link between a subject and a verb complemener when dealing with control-verbs is not taken into account.

An extensive amount of literature exists on these subjects [3, 4, 5] and the solutions proposed can be divided in two sets : in one solution the derivation tree is considered unusable and the syntax-semantic interface is built on the derived tree [6, 7] ; in the other solution, the information provided by the derivation tree need to be enriched and thus the LTAG formalism is modified, if not replaced by Multi Component TAG [8] to deal with modifier scope information in the case of [9].

³ The use of a Categorical Grammar(CG) could simplify a syntax-semantic interface because of the direct connection between CG's combination rules and λ -calculus functional application, so applying β -reduction while reducing CG rules will result in a correct λ -term. CG and LTAG are known to be very close (same generative power, both lexicalized formalism and a limited number of possible operations), but TAG derivation trees are almost direct representations of predicative structures and for most applications, this structure is what we need.

⁴ In this *mini*-model we replaced classical λ -calculus variables (x, y, \dots) by square bracket argumental position.

Although these solutions are considered elegant for solving scope ambiguities by putting together flat semantics and underspecification, they cannot represent missing argumental links while staying in the strict LTAG formalism. This is why the derived trees has been used.

Using derived trees, LTAG feature structures are used to rebuild predicative structure or logical formulae. The main argument against this method is that every event which occurs during the semantic process is bound to a node where a derivation would have occurred anyway (such as substitution node or adjunction node) so the feature structures added to the derived tree would be redundant with the derivation tree. Moreover, in the solution proposed by [6], the number of feature structures is not bound so the weak generative capacity of the formalism is extended as noted by [10].

In fact, we think that the proper way to use the strength of LTAG is to build our semantic from both the derivation tree and the derived tree through the use of what we call “Derivation Forests”.

In this paper, we will not discuss the problems with generating proper logical formulae, instead we will discuss how to provide a dependency graph whose main characteristics are to make missing argumental links appear and to provide all analyses within a single one structure. We will use the phenomenon of control-verb to guide our demonstration.

2 Control-Verb Problematic

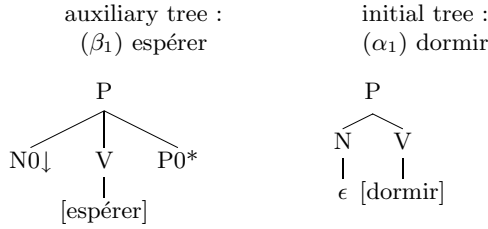
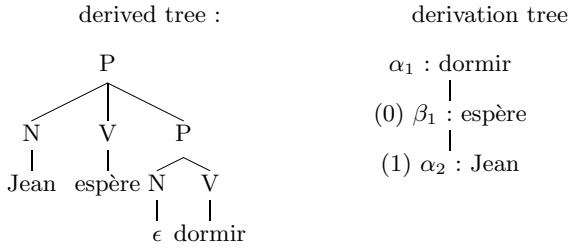
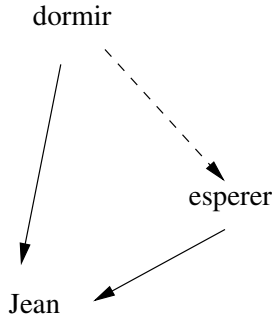
In this section we will briefly expose this problematic in the TAG framework⁵. Control-verbs are often used to illustrate a visible gap between syntax and semantic in TAG. Some deep syntactic argument cannot be represented in the derivation tree although it is supposed to reveal the predicate argument structure of a given sentence. For example, in the sentence 2b a syntactic agreement exists between Marie and the complement of the verb *être*, that is in French the adjective “belle” has a visible feminine marker. This agreement marks a co-indexation between the subject of “esperer” and the subject of “être”. Thus, if we consider “être belle” as a predicate, its argumental position is filled and should be present in the derivation tree following the co-occurrence predicate-argument principle.

- (2) a. Jean espere dormir
 Jean hopes to sleep
 b. Marie espere être belle
 Marie hopes to be nice

But given the toy grammar⁶ in figure 4, the link between these two phrase structures (“Marie” and “être belle”) does not appear in the derivation tree even

⁵ The reader may find a more complete explanation in [11].

⁶ Note the empty node in the initial tree for dormir, we call this node *unrealized substitution node*.

**Fig. 4.** Control verb toy grammar**Fig. 5.** LTAG analysis for “Jean espère dormir”**Fig. 6.** Graph for “Jean espère dormir”

though the derived tree, through its feature structures, shows the existence of such a link (figure 5). We want to make this syntactic link between the nodes denoted by α_1 and α_2 appear as stated by the dependency graph figure 6. For this purpose we need to use the information present in the derived tree and induced by the derivation tree.

We argue that a auxiliary tree anchored by a control verb (i.e Control Tree) has to transfer its controlled argument directly to the tree in which it has been adjoined. Therefore we must consider instead of visualizing tree projection in the derivation tree as nodes, we shall see them as segments of graphs. For the purpose of readability we introduce a new operation: Fusion of argument.

3 A Graph Vision of TAG Derivation

3.1 TAG Derivations and Incomplete Derivations

The only operations visible on a TAG derivation tree are substitution and adjunction because they mark the “jump” from one tree to another. We have to develop a way to represent the missing link and the corresponding derivation it implies. What information do we have ?

- We know the number of arguments (substitution nodes) in a tree ;
- We know from lexical information, which substitution nodes are going to be transfered and to where. This information is called the **control canvas**, defined in section 3.3, and is noted directly in this node.
- We know that a tree receives an argument coming from the adjunction of a Control Tree.

Thus, we can refine the derivation process in order to include the information defined above (cf. fig 7) : instead of working with node, we work with edges and we use a variable box (noted as $\boxed{?}$) to mark the fact that an argument is missing and will be realized through the fusion operation. Once the argument is substituted in tree β_1 , the fusion operation binds that same argument to the variable box of tree α_1 . As the variable box in the elementary tree α_1 is in reality bounded to a node which dominates an empty node, we call this “false” derivation an “incomplete derivation”.

According to [5], TAG derivations are dependencies which reveal deep syntactic structure. So, the dependency graph we were referring to is in fact a derivation graph.

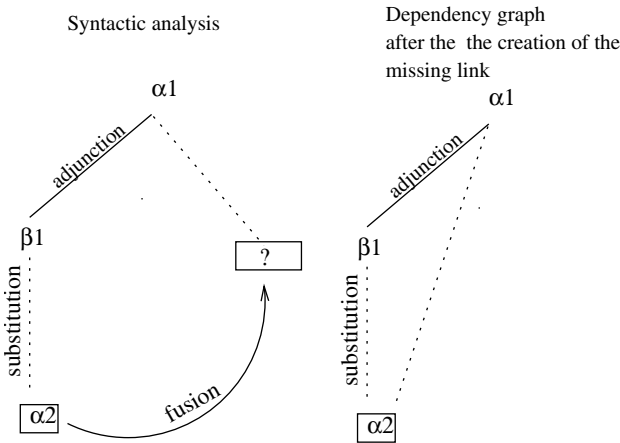


Fig. 7. Step of construction of derivation graph

3.2 The Operation of Fusion

To explain the operation of fusion, we introduce a new abstraction layer which encapsulates TAG derivation within the concept of argument realization.

This layer, composed with two types of transitions : argumental transitions and transfer transitions, is the interface which allows us to make the link between unrealized substitution and effective derivation.

For the sake of simplicity let us imagine we have in our hands a shared forest defined as a context free grammar as defined by [12] plus a stack argument which make this CFG much more like a LIG[13]. This supplementary stack argument allows us to keep the order of the derived tree directly into this forest when we parse this grammar top-down left-right, the derivation tree is also included because the rules which mark an effective derivation are of course in the core of the analysis [14].

The definition of the shared forest we use is given in section 5.1.

Each rule of this grammar is validated : when the rule appears in the proof of an effective derivation⁷ an item is inferred into a chart. The set of these items is the derivation forest minus the incomplete derivations. For example see figure 8.

Argumental Transitions. For the substitution of a tree α onto a tree γ in its node N , the derivation is called D_1 and the item is $\boxed{< N, \gamma, \alpha, subst >}$. It is read as “Substitution of α on the node N of γ ”.

We split this derivation into two more elementary transitions *get* and *put*. The transition *get* means that a node needs an argument and the transition *put* means that a tree is pushing an argument. For a “normal” substitution we get the following decomposition (fig.9) :

We know that the derivation exists because the rule which we created has been validated. We can also decompose the incomplete derivation testified by a node N , from a tree γ , dominating an empty string. As we do not yet have any node to co-index, the *put* transition will not appear (fig. 10). This derivation is called D_2 and the associated item is $< N, \boxed{?}, \gamma, subst >$. This item is interpreted as an argument needs to be linked to this node (fig. 10).

Transfer Transitions. While argumental transitions are managed by substitution and thus demonstrate obligatory completion of argument and thus can be predictable ; the transitions related to adjunction are completely different.

In this perspective, adjunction is almost the opposite of substitution : In case of substitution, the *get* transitions were waiting for completion of *put* transition (indicated by a down-oriented edge); in case of adjunction, we cannot predict precisely where it will occur but is we can tell that an auxiliary tree will have to adjoin somewhere. We indicate this fact by an up-oriented edge. This edge must be linked to an already existing node. Let us define a tree β which adjoins

⁷ Effective derivation occurs when the nodes in the left part of a rule are from a different tree than the right part, except when the rules mark a *foot return* transition.

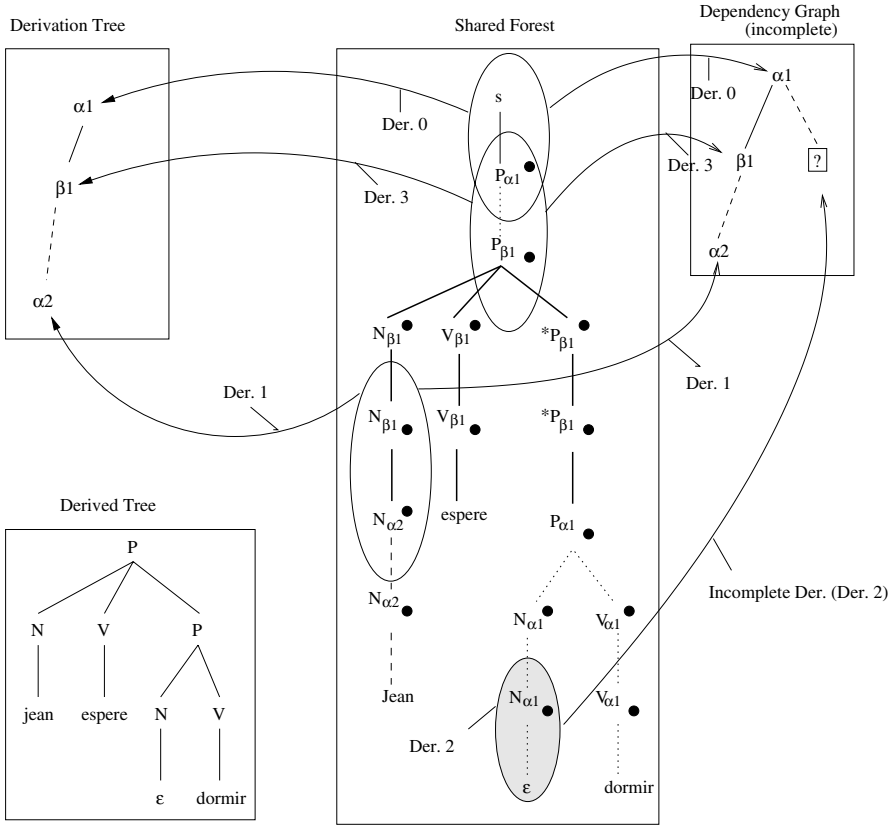


Fig. 8. Shared Forest with incomplete derivation

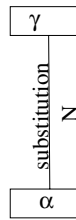


Fig. 9. Derivation $D_1 : \langle N, \alpha, \gamma, subst \rangle$

on a node N of a tree γ . The derivation is called D'_3 and the associated item is $\langle N, \beta, \gamma, adj \rangle$. Figure 11 shows the process.

The *fusion* operation takes place in a particular context :

1. We have a Control Tree β_c
2. This tree transfers its substitution node, as noted by its control-canvas, to a tree γ_c
3. the tree γ_c has an unrealized substitution node

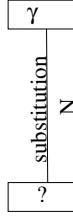


Fig. 10. Derivation $D_2 : < N, \boxed{?}, \gamma, subst >$

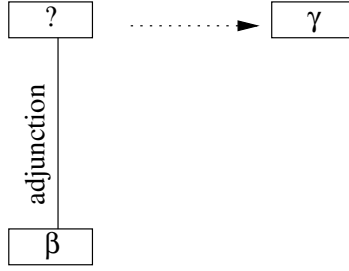


Fig. 11. Derivation $D'_3 : < N, \beta, \gamma, adj >$

This context can be translated formally in terms of derivations :

- The node N_{β_c} has the following control-canvas : $\boxed{i \rightarrow j}$
- The derivation D_1 , marks the effective substitution of α on $N_{\beta_c}^i$:

$$\boxed{D_1 : < N^i, \alpha, \beta_c, subst >}$$

- The derivation D_2 marks an incomplete derivation of the unrealized substitution on the node N of γ_c on the node N^j of γ_c :

$$\boxed{D_2 : < N^j, \boxed{?}, \gamma_c, subst >}$$

- The derivation D_3 marks the adjunction of β_c on the node X of γ_c :

$$\boxed{D_3 : < X, \beta_c, \gamma_c, adj >}$$

Thus, we can decompose the adjunction of β_c on γ_c . The *fusion operation* is the result of the application of the transitions defined above plus what we call a *catch* transition whose only role is to match the argument to be linked (fig. 12).

Then in order to fill in the missing argument, one needs only create a new inference rule which takes as parameters the three items D_1 , D_2 and D_3 to provide a new derivation item, D_4 , which demonstrates the link between “jean” and the unrealized substitution node.

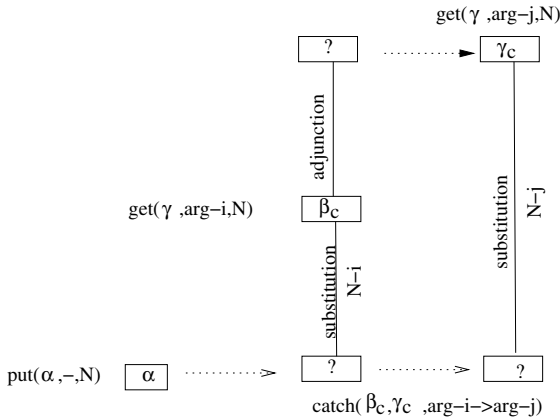


Fig. 12. Derivation $D_3 : \langle X, \beta_c, \gamma_c, adj \rangle$

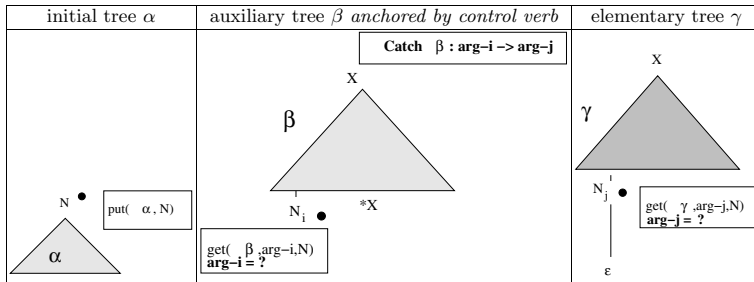


Fig. 13. Transition attachment

Synthetic View of the Process. Let us consider the adjunction of β on γ with α as the initial tree which substitutes onto the node N_i of β and which must be transferred on γ (fig 13).

Fusion Process. The fusion process takes place in three steps while the shared forest is validated. These steps are illustrated in figure 14 and are synthesized by a new inference rule defined in the next section.

3.3 The Missing Link Creation

We claim that a Control Tree pushes its argument over an elementary tree which is missing an argument.

The information indicating which Control Tree argument is to be pushed (and where it is to be pushed) is lexical in nature.

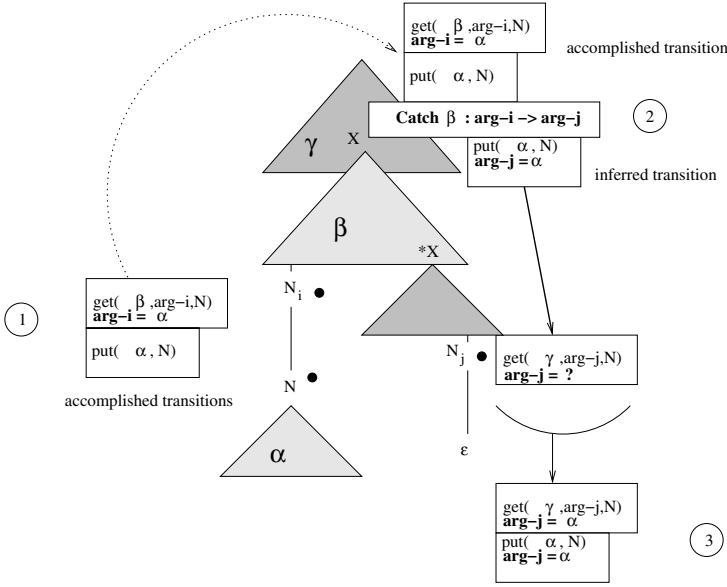


Fig. 14. Synthetic processus of argumental fusion

So, we defined a **Control canvas** associated to a Control Tree :

If an auxiliary tree controls its i^{th} argument and transfers it into the j^{th} argument of an elementary tree (whose j^{th} node dominates an empty string), we mark this fact in the former (i^{th}) node, as $\boxed{N_i \rightarrow j \downarrow}$ ⁸.

Let a Control Tree β_c , with root X , receive a substitution of a tree α on its node N whose control canvas is $\boxed{N_i \rightarrow j \downarrow}$ (item D1). Let a tree γ , root X , whose j^{th} node N dominates an empty string (item D2), receive the adjunction of β_c on its root (item D3). The new inference rule, the *fusion* operation, is:

$$\frac{D3 :< X, \beta_c, \gamma, adj > \quad D1 :< N_i \rightarrow j, \alpha, \beta, subst > \quad D2 :< N_j, \boxed{?}, \gamma, subst >}{D4 :< N_j, \alpha, \gamma, subst >}$$

Once $D4$ (viewed as the new link), is inferred, $D2$ is erased. So the set of these items can be seen as a graph⁹ described with co-referent links.

⁸ -John forbids Mary to sleep : $\boxed{N1 \rightarrow 0 \downarrow}$ - John hopes to sleep : $\boxed{N0 \rightarrow 0 \downarrow}$

⁹ this set also contains another kind of items : the item called *head item* which marks an initial tree spawning the whole string (start item), defined as follow :
if a tree α of root X , covers the string between 0 and n , the length of the string to be parsed, its derivation item, D_0 , is $D_0 : < X, \alpha, -, - >$

4 Conclusion

This proposition has been implemented in [14], its main characteristic is to make an extensive use of the characteristic of a shared forest. For lack of space, we have not presented the algorithms behind the generation of the shared forest. The nodes which are used by the fusion operation to generate the missing links are all nodes from the derivation tree, the node from the derived tree is the one which supports an incomplete derivation.

We based our work on the fact that the shared forest contained both the derived tree and the derivation tree. One of interesting properties is that using item and chart inference rules to generate our derivation items allows us to maintain a shared structure of derivations which may contain all possible derivation graphs in case of syntactic ambiguities. In fact, the structure we proposed could be seen as the semantic counterpart of a classic shared forest[15].

The only addition to the initial TAG formalism is the *control canvas*. We think that we can apply this process in order to deal with a restricted kind of elliptic coordination such as “John loved Mary and slept alone” where the subject of “to sleep” is the one of “to love”, we will just have to maintain a stack of incomplete derivations to maintain the consistency of the fusion operation.

Acknowledgements

We are very grateful to our anonymous reviewers for their valuable comments. We also would like to thank Eric Kow and Jacqueline Lai for their proofreading help.

References

1. Joshi, A.K.: Introduction to tree adjoining grammar. In Manaster-Ramer, A., ed.: *The Mathematics of Language*, J. Benjamins (1987)
2. Abeillé, A.: *Une grammaire lexicalisée d'arbres adjoints pour le français*. PhD thesis, Paris 7 (1991)
3. Candito, M.H., Kahane, S.: Can the tag derivation tree represent a semantic graph ? In: *Proceedings TAG+4*, Philadelphie. (1998) 21–24
4. Schabes, Y., Shieber, S.: An alternative conception of tree-adjoining derivation. *Computational Linguistics* **20** (1994) 91–124
5. Rambow, O., Joshi, A.K.: *A Formal Look at Dependency Grammar and Phrase Structure Grammars, with Special consideration of Word Order Phenomena*. Leo Wanner, Pinter London, 94 (1994)
6. Gardent, C., Kallmeyer, L.: Semantic construction in feature-based tag. In: *Proceedings of EACL 2003*. (2003)
7. Franck, A., van Genabith, J.: Gluetag : Linear logic based semantics for ltag -and what it teaches us about lfg and ltag-. In: *Proceedings of the LFG01 Conference*, University of Hong Kong, Hong Kong. (2001)
8. Weir, D.: *Characterizing Midly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania (1988)

9. Kallmeyer, L., Joshi, A.: Factoring predicate argument and scope semantics: Underspecified semantics with Itag. In: Proceedings of the 12th Amsterdam Colloquium, December. (1999)
10. Kallmeyer, L.: LTAG Semantics with Semantic Unification. In: Proceedings of TAG+7, To appears. (2004)
11. Abeilleé, A.: Verbes "à monté" et auxiliaires dans une grammaire d'arbres adjoints. LINX, Linguistique Institut Nanterre Paris X (1999)
12. Vijay-Shanker, K., Weir, D.: The use of shared forests in tree adjoining grammar parsing. In: EACL '93. (1993) 384–393
13. Aho, A.V.: Indexed grammars-an extension of context-free grammars. J. ACM **15** (1968) 647–671
14. Seddah, D.: Synchronisation des connaissances syntaxiques et sémantiques pour l'analyse d'énoncés en langage naturel à l'aide des grammaires d'arbres adjoints lexicalisées. PhD thesis, Université Henry Poincaré, Nancy (2004)
15. Billot, S., Lang, B.: The structure of shared forests in ambiguous parsing. In: 33rd Conference of the Association for Computational Linguistics (ACL'89). (1989)

5 Annex

5.1 Form of the Shared Forest

The definition of the shared forest we use is the result of the intersection of a given TAG grammar and an linear automata as described by [12]. This definition is extended in order to link the recognition of a subtree dominated by the node which initiates an adjunction to the foot node of the adjoined tree. We represent this forest with a context free grammar augmented by a stack containing the current adjunctions, which is close to a Linear Indexed Grammar,[13].

Each part of the rules corresponds to an item *a la* Cock Kasami Younger whose form is $\langle N, POS, I, J, STACK \rangle$ with N is a node of an elementary tree, POS marks the situation relative to an adjunction (marked \top if an adjunction is still possible, \perp if the adjunction is not possible. This is marked on figure with a bold dot in high position, \top , or a bold dot in low position, \perp). I and J are the start and end indices of the string dominated by the N node. $STACK$ is the stack containing all the call of the subtrees which has started an adjunction et which must be recognized by the foot recognition rules. We used S as the starting axiom of the grammar and n is the length of the initial string.

We show here only the rules relevant to derivation rules :

call transition	rules
Call subst	$\langle \perp, N_\gamma, i, j, -, -, Stack \rangle \rightarrow \langle \top, N_\alpha, i, j, -, -, Stack \rangle$
Call adj	$\langle \top, N_\gamma, i, j, -, -, Stack \rangle \rightarrow \langle \top, N_\beta, i, j, -, -, [N_\gamma Stack] \rangle$
Call axiom	$S \rightarrow \langle \top, N_\alpha, 0, n, -, -, \emptyset \rangle$
Call no subs	$\langle \perp, N_\gamma, i, j, -, -, Stack \rangle \rightarrow \text{true}$
Call foot	$\langle \perp, *N_\beta, i, j, -, -, [N_\gamma Stack] \rangle \rightarrow \langle \top, N_\gamma, i, j, -, -, [Stack] \rangle$

The “Call subst” rule is the rule which starts the recognition of a substitution of the initial tree α on the node N of the tree γ between the indices i and j . “Call adj” starts the recognition of the adjunction of the auxiliary tree β on the node N of an elementary tree γ between i and j . “Call axiom” starts the recognition α of an elementary tree spawning the whole string. “Call no subs” starts the recognition of a node N of a elementary tree γ dominating the empty node between the indices i and j . “Call foot” starts the recognition of a subtree dominated by the node N_γ between the indices i and j , the node $N_{\gamma\text{amma}}$ was the start of the adjunction of the auxiliary tree β and $*N_\beta$ its foot node.

In order to avoid the “call adj” rule to be over generating, we control the size of the stack by the number of possible adjunctions at a given state : if the automata has no cycle and if each state of the automata goes forward (j always superior to i), the number of possible adjunctions on a spine (the path between the root of an auxiliary tree and its foot) is bounded by the length of the string to analyse.

When Categorical Grammars Meet Regular Grammatical Inference

Isabelle Tellier

GRAppA & Inria Futurs, Lille,
MOSTRARE project**
Université Lille 3,
59653 Villeneuve d'Ascq, France
`isabelle.tellier@univ-lille3.fr`

Abstract. In this paper, we first study the connections between subclasses of AB-categorical grammars and finite state automata. Using this, we explain how learnability results for categorical grammars in Gold's model from structured positive examples translate into regular grammatical inference results from strings. A closer analysis of the generalization operator used in categorical grammar inference shows that it is strictly more powerful than the one used in usual regular grammatical inference, as it can lead outside the class of regular languages. Yet, we show that the result can still be represented by a new kind of finite-state generative model called a *recursive automaton*. We prove that every unidirectional categorical grammar, and thus every context-free language, can be represented by such a recursive automaton. We finally identify a new subclass of unidirectional categorical grammars for which learning from strings is not more expensive than learning from structures. A drastic simplification of Kanazawa's learning algorithm from strings for this class follows.

1 Introduction

Grammatical inference deals with the problem of how to infer a grammar from examples of sentences it generates - and from sentences it does not generate, if negative examples are available. In the grammatical inference community (see the ICGI conference), many efforts have concerned the learnability of subclasses of regular grammars, represented by finite state automata [1, 13, 8, 7].

The inference of context-free grammars is more difficult and has received less attention. The most achieved work in this domain is Kanazawa's [12], who proved learnability results of large subclasses of AB-categorical grammars in Gold's model from positive examples [9]. These results concern two kinds of input data: strings and structural examples, i.e. syntactic analysis structures

** This research was partially supported by: "CPER 2000-2006, Contrat de Plan état - région Nord/Pas-de-Calais: axe TACT, projet TIC"; fonds européens FEDER "TIC - Fouille Intelligente de données - Traitement Intelligent des Connaissances" OBJ 2-phasing out - 2001/3 - 4.1 - n 3. And by "ACI masse de données ACIMDD".

where intermediate categories are deleted. But these results are hardly useful, because (except in very restricted cases) the corresponding algorithms have a high complexity [5, 6]. Surprisingly enough, nobody seems to have ever tried to translate these results into the more restricted class of regular grammars. The main characteristic of this class is that it produces only flat trees. We can prove easily that, in this context, structural examples are available for free when strings are available. So, it is worth considering how learnability results for subclasses of AB-categorical grammars from structural examples translate into learnability results for subclasses of regular grammars from strings.

In this paper, we first study how finite state automata translate into categorical grammars, and conversely. We compare learning strategies used in both domains. Doing so, we compare the relative power of the usual generalization operator of “state fusion”, used in traditional regular grammatical inference and the generalization operator of “unifying substitutions on variables” used in categorical grammar learning. We prove that the latter is strictly more powerful than the former, as it sometimes allows to transform a categorical grammar generating a regular language into a categorical grammar generating a context-free grammar. Yet, in this case, it is still possible to represent the result of the operator as a generalized automaton. This leads to the definition of a new class of automata, called *recursive automata*. This class, which is a natural extension of finite state automata, has at least the expressive power of unidirectional categorical grammars, and can thus generate every context-free language.

This article thus proposes to unify grammatical inference results coming from two different traditions: the one focusing on finite state automata, and the one focusing on AB-categorical grammars, showing that they can both benefit from ideas coming from the other one.

2 Finite State Automata and Categorical Grammars

In this section, we first recall basic definitions concerning finite state automata and AB-categorical grammars, and show that there are easy correspondences between them.

2.1 Finite State Automata and Regular Languages

We recall here the classical notations for automata and regular languages.

Definition 1 (Finite State Automaton (FSA) and their Language). *A **finite state automaton** (FSA in the following) A is a 5-tuple $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ such that Q is the finite set of states of A , Σ its finite vocabulary, $q_0 \in Q$ is the initial state of A (we restrict ourselves to automata with a unique initial state) and $F \subseteq Q$ is the set of its final states. Finally, δ is the transition function of A , defined from $Q \times \Sigma$ to 2^Q .*

The language $L(A)$ recognized by A is defined as: $L(A) = \{w \in \Sigma^ \mid \delta^*(q_0, w) \cap F \neq \emptyset\}$, where δ^* is the natural extension of δ on $Q \times \Sigma^*$ such that: for any $a \in \Sigma$, any $u \in \Sigma^*$ and any $q \in Q$, $\delta^*(q, au) = \{\delta^*(q', u) \mid q' \in \delta(q, a)\}$.*

*The set of languages recognized by FSA is called the set of **regular languages**.*

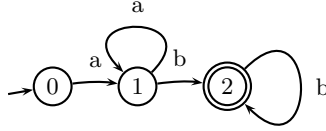


Fig. 1. A Simple Finite State Automaton

Example 1. Figure 1 displays a simple FSA A such that $L(A) = a^+b^+$.

2.2 Grammars and Categorical Grammars

We recall the classical definitions of a generative grammar, and of categorical grammars. Here, we restrict ourselves to AB-(or classical) categorical grammars.

Definition 2 (Generative Grammars and their Language). A **generative grammar** (or simply a grammar in the following) is a 4-tuple $G = \langle \Sigma, N, P, S \rangle$ with Σ the finite terminal vocabulary of G , N its finite non terminal vocabulary, $P \subset (\Sigma \cup N)^+ \times (\Sigma \cup N)^*$ its finite set of rules and $S \in N$ the axiom.

The language generated by a grammar G is $L(G) = \{w \in \Sigma^* \mid S \xrightarrow{*} w\}$ where $\xrightarrow{*}$ is the reflexive and transitive closure of the relation defined by P .

Definition 3 (Categories, AB-Categorical Grammars and their Language). Let \mathcal{B} be a set (at most countably infinite) of basic categories containing a distinguished category $S \in \mathcal{B}$, called the axiom. We note $\text{Cat}(\mathcal{B})$ the smallest set such that $\mathcal{B} \subset \text{Cat}(\mathcal{B})$ and for any $A, B \in \text{Cat}(\mathcal{B})$ we have: $A/B \in \text{Cat}(\mathcal{B})$ and $B \setminus A \in \text{Cat}(\mathcal{B})$.

For every finite vocabulary Σ and for every set of basic categories \mathcal{B} ($S \in \mathcal{B}$), a **categorical grammar** is a finite relation G over $\Sigma \times \text{Cat}(\mathcal{B})$. We note $\langle v, A \rangle \in G$ the assignment of the category $A \in \text{Cat}(\mathcal{B})$ to the element of the vocabulary $v \in \Sigma$. AB-categorical grammars (CGs in the following) are categorical grammars where the syntactic rules take the form of two rewriting schemes: $\forall A, B \in \text{Cat}(\mathcal{B})$

- FA (Forward Application) : $A/B \ B \rightarrow A$
- BA (Backward Application) : $B \ B \setminus A \rightarrow A$

The language generated (or recognized) by a CG G is:

$L(G) = \{w = v_1 \dots v_n \in \Sigma^+ \mid \forall i \in \{1, \dots, n\}, \exists A_i \in \text{Cat}(\mathcal{B}) \text{ such that } \langle v_i, A_i \rangle \in G \text{ and } A_1 \dots A_n \xrightarrow{*} S\}$, where $\xrightarrow{*}$ is the reflexive and transitive closure of \rightarrow .

Example 2. For example, let $\mathcal{B} = \{S, T, CN\}$ (where T stands for “term” and CN for “common noun”), $\Sigma = \{John, runs, loves, a, cat\}$ and G be defined by: $G = \{\langle John, T \rangle, \langle runs, T \setminus S \rangle, \langle loves, (T \setminus S)/T \rangle, \langle loves, T \setminus (S/T) \rangle, \langle cat, CN \rangle, \langle a, (S/(T \setminus S))/CN \rangle, \langle a, ((S/T) \setminus S)/CN \rangle\}$. This CG generates sentences like “John runs”, “John loves a cat”, etc.

Definition 4 (FA-Structures, Structural Examples, Structured Language). A **functor-argument (or FA-) structure** over an alphabet Σ is a binary-branching tree whose leaf nodes are labeled by elements of Σ and whose internal nodes are labeled either by BA or FA. The set of FA-structures over Σ is denoted Σ^F .

For any AB-categorical grammar $G \subset \Sigma \times \text{Cat}(\mathcal{B})$, a structural example for G is an element of Σ^F which can be obtained from the analysis tree resulting from a syntactic parsing of a string $w \in L(G)$ in G by deleting each of its categories. For any CG $G \subset \Sigma \times \text{Cat}(\mathcal{B})$, the structured language $FL(G)$ associated with G is the set of structural examples for G .

\mathcal{G} denotes the class of CGs. For every integer $k \geq 1$, the set of CGs assigning at most k distinct categories to each member of the vocabulary is the class of k -valued CGs denoted by \mathcal{G}_k . When $k = 1$ the grammars are also called rigid.

2.3 From Automata to Categorical Grammars and Back

The expressive power of CGs is the one of ϵ -free (ϵ is the empty string) context-free languages [2]. So, of course, they can also generate ϵ -free regular languages. Correspondences between FSA and CGs are easy to define.

Definition 5 (Regular CGs). We call a **regular CG** a CG $G \subset \Sigma \times \text{Cat}(\mathcal{B})$ that only contains assignments of the form $\langle v, A \rangle$ or $\langle v, A/B \rangle$ with $v \in \Sigma$ and $A, B \in \mathcal{B}$. The set of regular CGs is noted \mathcal{G}_r .

Property 1 (Transformation of an Automaton). Let $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a FSA. Let $\mathcal{B} = (Q \setminus \{q_0\}) \cup \{S\}$ (where $S \notin Q$). It is possible to define a regular CG $G \subset \Sigma \times \text{Cat}(\mathcal{B})$ such that $L(G) = L(A) \setminus \{\epsilon\}$.

Proof (Proof of Property 1). This property is the consequence of sequentially applying the classical transformation of a FSA A into a left-linear regular grammar $G_1 = \langle \Sigma, Q, P_1, S \rangle$, then applying the transformation of G_1 into a CG: $\forall a \in \Sigma$ and $\forall q, q' \in Q$ such that $q' \in \delta(q, a)$ do:

- if $q' \in F$ then
 - $q \longrightarrow a$ is a rule of G_1 (element of P_1) and $\langle a, q \rangle \in G$ (replace q_0 by S);
 - if $\exists u \in \Sigma, \exists q'' \in \delta(q', u)$ then $q \longrightarrow aq'$ is a rule of G_1 and $\langle a, q/q' \rangle \in G$ (replace q_0 by S);
- else $q \longrightarrow aq'$ is a rule of G_1 and $\langle a, q/q' \rangle \in G$ (replace q_0 by S).

$\epsilon \in L(A)$ if and only if $q_0 \in F$. This situation gives rise to a new rule $S \longrightarrow \epsilon$ in P_1 . But in CGs, it is impossible to assign a category to ϵ , so this rule has no counterpart in G . So, we have: $L(A) \setminus \{\epsilon\} = L(G_1) \setminus \{\epsilon\} = L(G)$. \square

Example 3. Let us apply the previous process to the FSA given in Example 1. The rules of the corresponding left-linear regular grammar are the following (where the state of number i is associated with a non terminal symbol noted

Q_i , with $Q_0 = S$): $S \longrightarrow aQ_1$, $Q_1 \longrightarrow aQ_1$, $Q_1 \longrightarrow b$, $Q_1 \longrightarrow bQ_2$, $Q_2 \longrightarrow b$, $Q_2 \longrightarrow bQ_2$. And the CG G is:

$$G = \{\langle a, S/Q_1 \rangle, \langle a, Q_1/Q_1 \rangle, \langle b, Q_1 \rangle, \langle b, Q_1/Q_2 \rangle, \langle b, Q_2 \rangle, \langle b, Q_2/Q_2 \rangle\}.$$

So, FSA can easily be *lexicalized*. Note that the only operator used in categories of a regular CG is / and the only useful rule is FA (it would have been \ and BA if we had first transformed the automaton into a right-linear regular grammar). In fact, *the previous transformation preserves not only the string language, but also the structured language*. A crucial consequence is that structural examples in the sense of Definition 4 are available for free from the corresponding strings: underlying structures produced by automata are always flat, and the only label for internal nodes is FA .

Example 4. Figure 2 displays two analysis trees produced by the CG obtained in Example 3, and (on the right) the corresponding structural examples.

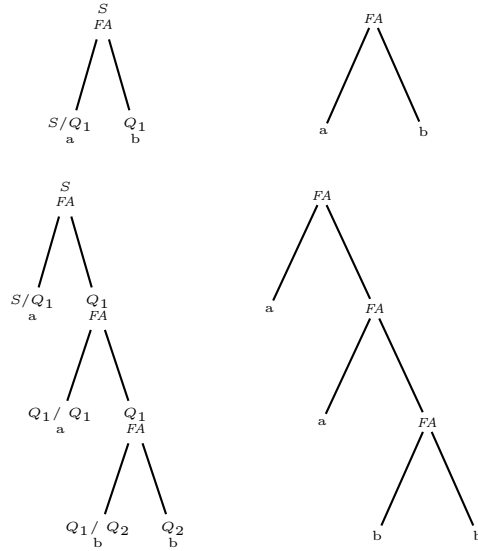


Fig. 2. Syntactic Analyses and the corresponding Structural Examples

Property 2 (Transformation of a Regular AB-Categorical Grammar). Every regular CG $G \subset \Sigma \times \text{Cat}(\mathcal{B})$ can be transformed into a FSA $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ recognizing the same (ϵ -free) language.

Proof (sketch of the proof of Property 2). This transformation is the reverse of the one described in Property 1: the only thing to pay attention to is to add a unique final state F_A in A . Let $Q = \mathcal{B} \cup \{F_A\}$ with $F_A \notin \mathcal{B}$, $q_0 = S$ and $F = \{F_A\}$. Each assignment $\langle a, U/V \rangle \in G$ corresponds to a transition labelled

by a between the states U and V in A (so: $\delta(U, a) = V$) and each assignment $\langle a, U \rangle \in G$ to a transition labeled by a between U and F_A ($\delta(U, a) = F_A$). \square

Example 5. The automaton built by applying this operation to the CG obtained in Example 3 is given in Figure 3. It is not exactly the same as the initial one (in Example 1), because of the final state added to the states coming from basic categories. The result is, usually, nondeterministic.

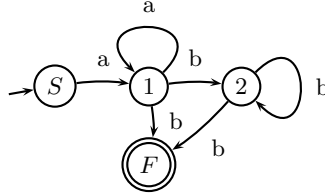


Fig. 3. The Automaton Obtained from the AB-Categorical Grammar

Remark 1. Properties 1 and 2 do not mean that only the CGs that are regular generate a regular language. The CG of Example 2 is not regular in the sense of Definition 5 but it generates a finite (and thus regular) language. But the structures it produces are not flat.

Property 3 (Language Associated with a State). Let G be a regular CG and A be the automaton obtained from it. Then for any basic category Q of G (corresponding to a non-final state Q of A , $S = q_0$), we have two equivalent ways to define the language $L(Q)$ associated with Q :

$L(Q) = \{w = v_1 \dots v_n \in \Sigma^+ \mid \forall i \in \{1, \dots, n\} \exists A_i \in \text{Cat}(\mathcal{B}) \text{ such that } \langle v_i, A_i \rangle \in G \text{ and } A_1 \dots A_n \rightarrow^* Q\}$ and $L(Q) = \{w \in \Sigma^+ \mid F_A \in \delta^+(Q, w)\}$.

Proof (sketch of proof of Property 3). The proof is an easy consequence of Properties 1 and 2, where Q replaces S . \square

The second definition of state language slightly differs from the classical one, because it excludes ϵ (we know that $Q \neq F_A$ so $\epsilon \notin L(Q)$) and uses the fact that there is only one terminal state F_A in A . So, strings in $L(Q)$, which can be associated with the category Q in G , also correspond in A to strings produced by following a path starting from the state Q and reaching the final state F_A . Of course, if $Q = q_0 = S$, we have: $L(S) = L(A) = L(G)$.

Example 6. In the automaton of Figure 3, $L(Q_1) = a^*b^+$ and $L(Q_2) = b^+$.

3 Inference of CGs from Positive Examples

The learnability of CGs in Gold's model from positive examples has received great attention in the last years. Now that we have an easy translation of a subclass of such grammars (the subclass of the regular ones) into automata, it is natural to see how these results translate into regular language learning, to compare them with known results in this domain.

3.1 Gold's Model

To deal with questions of learnability, Kanazawa [12] introduced the notion of grammar system, allowing a reformulation of Gold's model of *identification in the limit from positive examples* [9]. We recall this notion here.

Definition 6 (Grammar System). A *grammar system* is a triple $\langle \Omega, \Lambda, L \rangle$ made of a hypothesis space Ω (here, Ω will be a set of grammars), a sample space Λ , which is a recursive subset of A^* , for some fixed alphabet A (elements of Λ are sentences and subsets of Λ are languages) and $L : \Omega \rightarrow \text{pow}(\Lambda)$ is a naming function. The question of whether $w \in L(G)$ which holds between $w \in \Lambda$ and $G \in \Omega$, is supposed to be computable.

The main grammar systems we deal with in the following of this paper are $\langle \mathcal{G}, \Sigma^*, L \rangle$ and $\langle \mathcal{G}, \Sigma^F, FL \rangle$. The first one concerns the learnability of CGs from strings and the second one the learnability of CGs from structural examples.

Definition 7 (Learnability Criterion). Let $\langle \Omega, \Lambda, L \rangle$ be a grammar system and $\phi : \cup_{k \geq 1} \Lambda^k \rightarrow \Omega$ be a computable function. We say that ϕ **converges** to $G \in \Omega$ on a sequence $\langle s_i \rangle_{i \in \mathbb{N}}$ of elements of Λ if $G_i = \phi(\langle s_0, \dots, s_i \rangle)$ is defined and equal to G for all but finitely many $i \in \mathbb{N}$ - or equivalently if there exists $n_0 \in \mathbb{N}$ such that for all $i \geq n_0$, G_i is defined and equal to G . Such a function ϕ is said to **learn** $\mathcal{G} \subseteq \Omega$ if for every language L in $L(\mathcal{G}) = \{L(G) | G \in \mathcal{G}\}$ and for every infinite sequence $\langle s_i \rangle_{i \in \mathbb{N}}$ that enumerates the elements of L (i.e. such that $\{s_i | i \in \mathbb{N}\} = L$), there exists some G in \mathcal{G} such that $L(G) = L$ and ϕ converges to G on $\langle s_i \rangle_{i \in \mathbb{N}}$.

Theorem 1 (Learnability of \mathcal{G}_k [12]). For any $k \geq 1$, the class \mathcal{G}_k of k -valued CGs is learnable in the grammar systems $\langle \mathcal{G}, \Sigma^*, L \rangle$ (i.e. from strings) and $\langle \mathcal{G}, \Sigma^F, FL \rangle$ (i.e. from structural examples).

3.2 Categorical Grammars and Regular Grammatical Inference

Of course, Theorem 1 holds for k -valued regular CGs. In this section, we first translate Kanazawa's results into regular grammatical inference. We then show the equivalence between two other known results in the case of regular CGs.

Theorem 2 (Learnability of k -valued Regular CGs). For every $k \geq 1$, the class of k -valued regular CGs $\mathcal{G}_k \cap \mathcal{G}_r$ is learnable from structural examples and from strings.

Proof (Sketch of Proof of Theorem 2). This is a direct consequence of Theorem 1, restricted to the class \mathcal{G}_r . To learn regular k -valued CGs from strings, you just have to apply the BP learning strategy from structural examples that are flat trees with FA internal nodes, then to select among the output the CGs that are isomorph with a regular CG (which is decidable) before performing the inclusion tests. \square

It is interesting to notice that, in the case of regular CGs, unlike in the case of general CGs, strings and structures are equivalent - that is, structures are available for free from strings. This is the idea we will try to generalize for larger classes of CGs in the following section.

Unfortunately, this class of grammars is not very interesting. As a matter of fact, if k is given, it is also a bound on the maximal number of transitions labeled by the same element of vocabulary in the corresponding FSA: there exists a finite number of distinct automata satisfying this condition, so the learnability result is in fact trivial.

Nevertheless, the notion of k -valued automaton, i.e. FSA which are the result of applying the process of Property 2 to k -valued regular CGs is original. As a matter of fact, it focuses on the total number of transitions labeled by the same symbol in an automaton, instead of focusing on the total number of states in this automaton. k -valued automata seem well adapted to large alphabets Σ (especially when k is small), which contrasts with usual classes of automata (and usual learning algorithms) considered in the field of regular inference.

Other interesting results worth being compared: the one concerning the class of 0-reversible FSA [1] and the one concerning the class of reversible CGs [3].

Definition 8 (0-Reversibility of a FSA [1]). *A FSA is said to be 0-reversible if and only if it is deterministic and the automaton obtained by reversing the transitions backwards is also deterministic.*

Definition 9 (Reversibility of a CG [3]). *A CG is said to be reversible if it does not contain two assignments of categories for the same element of vocabulary, which are distinct by only one basic category.*

Theorem 3 (Equivalence of these Reversibility Notions). *Let G be a regular CG and A be the FSA obtained from it. A is 0-reversible in the sense of Definition 8 if and only if G is reversible in the sense of Definition 9.*

Proof (Sketch of Proof of Theorem 3). This property is a direct consequence of Theorem 2. As we only considered automata with a unique initial state and no ϵ transition, the conditions for A to be deterministic only concern transitions starting from the same state and labeled by the same symbol. They are equivalent for G with the following ones: $\forall a \in \Sigma$

- $\forall Q_1, Q_2, Q_3 \in \mathcal{B}: \langle a, Q_1/Q_2 \rangle \in G \text{ and } \langle a, Q_1/Q_3 \rangle \in G \Leftrightarrow Q_2 = Q_3.$
- $\forall Q_1, Q_2 \in \mathcal{B}: \langle a, Q_1 \rangle \in G \text{ and } \langle a, Q_1/Q_2 \rangle \in G \Leftrightarrow Q_2 = F_A$ (in fact, $\langle a, Q_1 \rangle$ stands for $\langle a, Q_1/F_A \rangle$);

Similarly, as A has only one final state, the conditions for the reversed automaton to be deterministic are equivalent with the following ones: $\forall a \in \Sigma$

- $\forall Q_1, Q_2 \in \mathcal{B}: \langle a, Q_1 \rangle \in G \text{ and } \langle a, Q_2 \rangle \in G \Leftrightarrow Q_1 = Q_2$
- $\forall Q_1, Q_2, Q_3 \in \mathcal{B}: \langle a, Q_1/Q_2 \rangle \in G \text{ and } \langle a, Q_3/Q_2 \rangle \in G \Leftrightarrow Q_1 = Q_3.$

For regular CGs, these conditions coincide with the one of Definition 9. □

So, the learnability of the class of 0-reversible FSA from strings [1] is equivalent with the one concerning the class of reversible CGs [3] from structures, in the special case of regular CGs. The corresponding learning algorithms should be more carefully compared, but they also seem equivalent.

3.3 Learning Algorithm

The learnability results of Theorem 1 are not only theoretical ones. The original algorithm BP able to identify the set of k -valued CGs without useless category compatible with a set of structural examples is due to Buszkowski & Penn [4]. We briefly recall it here, exemplifying it on a set of flat structural examples.

To identify every k -valued CG compatible with a given sample D of structural examples, the first steps are the following for each element of D :

1. introduce a label S at the root of each structural example;
2. introduce a distinct variable x_i at each argument node (i.e. at the left daughter of each BA node and at the right daughter of each FA node);
3. introduce a fractional category at every other node, respecting the labels of the functional application (FA or BA) to be applied.

The result of collecting all the categories associated with each distinct member of the vocabulary after these steps is a CG called the **general form** of D and noted $GF(D)$.

Example 7. Let D be the couple of structural examples given in Example 4. The previous process gives the result of Figure 4, and $GF(D)$ is defined as follows:

- a: $S/x_1, S/x_4, x_4/x_3$;
- b: $x_1, x_3/x_2, x_2$.

The corresponding FSA is given in Figure 5: it very much looks like what is known in regular grammatical inference as the **maximal canonical automaton** $MCA(D)$: the only difference is that the FSA corresponding to $GF(D)$ has a unique initial state and a unique final state. It is generally not deterministic.

Then, substitutions that unify category assignments are searched for.

Definition 10 (Variable Categories and Substitutions). Let χ be an enumerably infinite set of variables and $\mathcal{B} = \chi \cup \{S\}$. A substitution is a function $\sigma : \chi \longrightarrow \text{Cat}(\mathcal{B})$ that maps variables to categories (it is initialized by the Identity function on χ). A substitution is extended to a function from categories to categories as follows: (i) $\sigma(S) = S$, (ii) $\sigma(A/B) = \sigma(A)/\sigma(B)$ and (iii) $\sigma(A \setminus B) = \sigma(A) \setminus \sigma(B)$ for any $A, B \in \text{Cat}(\mathcal{B})$. Similarly, a substitution is naturally extended to apply to a CG: $\forall G \in \mathcal{G}, \sigma(G) = \{\langle v, \sigma(A) \rangle \mid \langle v, A \rangle \in G\}$. For any CG G , a unifying substitution for G is a substitution that unifies categories assigned to the same element of the vocabulary in G .

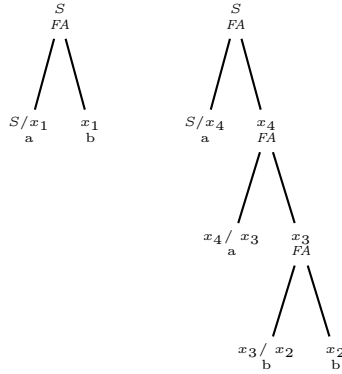


Fig. 4. First Step of the Learning Algorithm

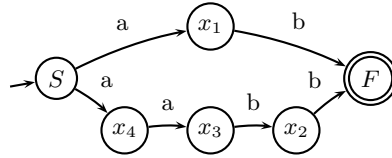


Fig. 5. The Automaton Corresponding with $GF(D)$

Property 4 (Fundamental Property [4]). For any CG G , the following two properties are equivalent: (i) $D \subseteq FL(G)$ and (ii) $\exists \sigma$ such that $\sigma[GF(D)] \subseteq G$.

In other words, CGs whose structured language is compatible with the input D are those which contain a substitution of $GF(D)$. If the target grammar is k -valued, the learning strategy BP thus only consists in finding every possible unifying substitution σ such that the grammar $\sigma[GF(D)]$ is k -valued. If $k = 1$, the learning process is incremental and the target grammar, if it exists, is unique and available in the limit. If $k > 1$, a remaining problem is to select one grammar among the set. This is performed by inclusion tests on the structured languages of the candidates (or by bounded inclusion tests for string languages). We will not go into further details about this final step.

Applying a substitution on a CG is a generalization operation, because we have the following property [4]: $\sigma(G_1) \subseteq G_2 \implies FL(G_1) \subseteq FL(G_2)$. So we always have: $FL(G) \subseteq FL(\sigma(G))$ and, similarly, $L(G) \subseteq L(\sigma(G))$. But in usual regular grammatical inference, the most often used generalization operator is the one of state merging [1, 13, 8]. What is the link between these two operators? In the context of a set D containing only flat structures with internal nodes FA , $GF(D)$ is necessarily a regular CG (see Example 7). So, only two cases can occur to unify two categories:

- conditions of the form $\sigma(x_i) = \sigma(x_j) = x_j$ for any $x_i \in \chi$ and any $x_j \in \chi \cup \{S\}$ specify a state merging: states x_i and x_j are merged.

- conditions of the form $\sigma(x_i) = x_j/x_k$, for any $x_i \in \chi$ and any $x_j, x_k \in \chi \cup \{S\}$ are more difficult to understand. Such a condition means two things:
 - the state x_i must be renamed as a state called x_j/x_k ;
 - every string that could be associated with the category x_i in the grammar $GF(D)$ can now be used as a “transition” between the states x_j and x_k .

Example 8. For example, let us define a substitution σ that unifies some of the categories assigned to a and b in the grammar $GF(D)$ of Example 7 as follows: $\sigma(x_4) = \sigma(x_1) = x_3/x_2$ (and σ is the identity elsewhere). The resulting CG $\sigma(GF(D))$ is the following:

- a: $S/(x_3/x_2), (x_3/x_2)/x_3$;
- b: $x_3/x_2, x_2$.

This CG is no longer regular. Nevertheless, it can be represented in a generalized automaton by adding a “recursive transition”, that is a transition labeled by a state (here, the state x_3/x_2). The corresponding automaton is given in Figure 6.

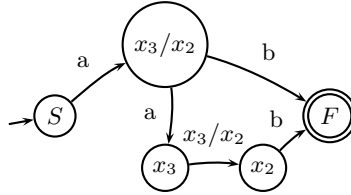


Fig. 6. The Generalized Automaton Corresponding with $\sigma(GF(D))$

In this automaton, previous states x_1 and x_4 were merged and renamed as x_3/x_2 , and a new “recursive” transition labeled by x_3/x_2 replaces the one that was labeled by b between the states x_3 and x_2 . To use this new transition, you need to produce a string of category x_3/x_2 , that is a string that belongs to the state language of x_3/x_2 . According to Property 3, another way to characterize such a string is that it would be obtained by following a path from the state x_3/x_2 to the final state F . To follow such a path, the first possible choice is, of course, the direct transition labeled by b . But another possible choice is to reach x_3 by a where the previous choice is posed again. Of course, a stack is necessary to remember the list of recursive transitions used. The language recognized by this generalized automaton is $a^n b^n$, which is not regular. This generalized automaton can be considered as a special case of Recursive Transition Network.

To understand how such a generalization could occur, let us look at the analysis tree produced by the grammar for the string $aaabbb$, given in Figure 7.

This tree is no longer flat. To understand how it was built, look back at the second tree of Figure 4. The specification of $\sigma(x_4) = x_3/x_2$ provides an equality between labels of this tree: the label of an internal node (x_4) becomes equal to the label of a leaf (x_3/x_2). This equality opens the possibility to insert the subtree rooted by x_4 in the place of the leaf labeled by x_3/x_2 , as is done in Figure 7. This operation is exactly what is called an *adjunction*, in the formalism of Tree Adjoining Grammars [11].

$L(R) = \{w \in \Sigma^+ \mid F \in \gamma^+(q_0, w)\}$, where γ^+ is the natural extension of γ on $Q \times \Sigma^+$, i.e. for any $u \in \Sigma^+$ and $v \in \Sigma^*$ any $q \in Q$, $\gamma^+(q, uv)$ is the smallest subset containing $\{\gamma^*(q', v) \mid q' \in \gamma(q, u)\}$ if $u \in \Sigma$ and $\{\gamma^*(q', v) \mid \exists t \in Q \text{ such that } q' \in \gamma(q, t) \text{ and } u \in L(t)\}$ else, where $L(t)$ is the state language of t .

RA also produce *structures*. These structures are not necessarily flat, because recursive transitions allow a real branching. A real recursivity occurs when there exists a path starting from a state $q \in Q$, using a recursive transition labelled by q and reaching the final state (as it was the case for x_3/x_2 in Figure 6).

Theorem 4 (From Unidirectional GCs to RA). *A unidirectional CG only assigns categories that are either basic or built from the operator $/$. The set of unidirectional CGs is noted \mathcal{G}_U . Every $G \in \mathcal{G}_U$ can be transformed into a strongly equivalent RA, i.e. a RA generating the same structured language.*

Proof (Proof of Theorem 4). It is well known [10] that any CG $G \subset \Sigma \times \text{Cat}(\mathcal{B})$ can be transformed into a strongly equivalent context-free grammar in Chomsky Normal Form $H = \langle \Sigma, N, P, S \rangle$ in the following way: N is the set of every subcategory of a category assigned to a member of the vocabulary in G (a category is a subcategory of itself). Then, for every $\langle v, A \rangle \in G$, let $A \longrightarrow v \in P$ and for all A, B in N , let $A \longrightarrow A/B \ B \in P$ (for unidirectional CGs, this is enough). The set of states of our RA R is the set $N \cup \{F\}$, with $F \notin N$. Rules of the form $A \longrightarrow v$ correspond to a transition labelled by v between the state A and the final state F . Rules of the form $A \longrightarrow A/B \ B$ correspond to a recursive transition labelled by A/B between states A and B . The use of a rule of this form in a derivation in G means that a subtree rooted by A can be decomposed into two subtrees: one rooted by A/B and one rooted by B . This is exactly what is also expressed by the corresponding recursive transition in R . \square

Corollary 1. *Unidirectional CGs can generate every ϵ -free context-free language [2]. So, it immediately follows from Theorem 4 that every ϵ -free context-free language can also be generated by a RA.*

Example 9. The classical unidirectional CG recognizing the language $a^n b^n$, $n \geq 1$, is the following: $G = \{\langle a, S/B \rangle, \langle a, (S/B)/S \rangle, \langle b, B \rangle\}$. The corresponding RA (distinct from the one of Example 8) is given in Figure 8. This RA can be simplified: the recursive transitions that are not *really* recursive can be lexicalized. Here, you can delete the state $(S/B)/S$ and replace the label of the recursive transition between S/B and S by a . But it is not possible for the state S/B .

The main interest of RA is that they produce the same structures as the ones produced by unidirectional CGs, i.e. using only FA rules. So, we hope to infer them from flat trees, like in Example 8. The problem is that the RA obtained from the process exemplified in Example 9 does not belong to the search space of any set of flat trees, because of the states that are not reachable from the initial state. So, we will need a more constraint form for RA (or for unidirectional CGs).

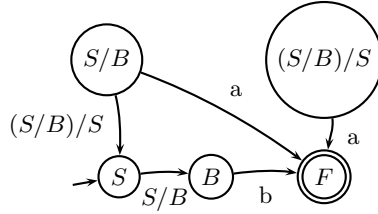


Fig. 8. Another RA recognizing $a^n b^n$

4.2 Learning Unidirectional CGs from Flat Structure

When only strings are available, Kanazawa's learning strategy consists in generating every possible structural example compatible with the input data, then applying the process described in section 3.3. It is a very expensive strategy, not tractable in practice. On the basis of the previous examples, we propose to refine Kanazawa's learning strategy to learn CF-languages from strings.

Definition 13. A CG G is said to have no useless category if every assignment of a category to a member of the vocabulary is used in at least one syntactic analysis of an element of $L(G)$ (G is also said to be in reduced form [12]). Let: $\mathcal{G}_{k,\sigma} = \{\sigma(G) | G \in \mathcal{G}_k \cap \mathcal{G}_r \text{ and } G \text{ has no useless category and } \sigma \text{ is a unifying substitution for } G\}$.

Of course, for every $k \geq 1$, $\mathcal{G}_{k,\sigma} \subseteq \mathcal{G}_k \cap \mathcal{G}_U$ and $\bigcup_{k \geq 1} \{L(G) | G \in \mathcal{G}_{k,\sigma}\}$ has a non null intersection with the set of non-regular CF-languages (see Example 8). But we do not know whether it can generate every CF-language.

The problem is that the notion of accessibility in a RA is different from the notion of accessibility in a FSA, because of recursive transitions. A state which is not accessible in a FSA (corresponding with a useless category in the corresponding CG) can become accessible after a substitution has been applied.

For every k , the main interest of the class $\mathcal{G}_{k,\sigma}$ is that it naturally extends the class $\mathcal{G}_k \cap \mathcal{G}_r$, whose fundamental property is that its members produce structural examples that are flat trees with FA internal nodes only. This property can be used to adapt Kanazawa's standard learning algorithm, as explained below.

Theorem 5. For every $k \geq 1$, for every $G \in \mathcal{G}_{k,\sigma}$, there exists a set D of flat structures with FA internal nodes and there exists τ , a unifying substitution for $GF(D)$ such that $G = \tau(GF(D))$.

Proof (proof of Theorem 5). For every $G \in \mathcal{G}_{k,\sigma}$, by definition there exists $G' \in \mathcal{G}_k \cap \mathcal{G}_r$ without useless category and σ a unifying substitution for G' such that $G = \sigma(G')$. We know by Theorem 2 that $\mathcal{G}_k \cap \mathcal{G}_r$ is learnable from structural examples and from strings. Let D be a characteristic set of structural examples for G' (see [12]). G' is regular, so D is only made of flat structures with FA internal nodes. G' has no useless category, so it belongs to the result of the BP algorithm [12] whose inputs are k and D . This means that there exists a

unifying substitution ρ for $GF(D)$ such that $G' = \rho(GF(D))$. So $G = \sigma(G') = \sigma(\rho(GF(D)))$. Let $\tau = \sigma \circ \rho$, which is a unifying substitution for $GF(D)$. \square

Theorem 5 means that the members of $\mathcal{G}_{k,\sigma}$ admit a characteristic sample made of only flat structures, and that they belong to the result of the BP algorithm whose input is this characteristic sample. This suggests an adaptation of Kanazawa's standard algorithm to learn the class $\mathcal{G}_{k,\sigma}$ from strings (see Algorithm 1). The main difference between this algorithm and Kanazawa's is that not all binary branching structures need to be associated to each string: it is enough to associate flat structures with FA internal nodes. But, to ensure the convergence of this algorithm, flat structures will be associated only to input strings not already recognized (associated with any structure) by the current hypothesis grammars (remember that a^3b^3 is recognized by the CG of Example 8 but not with a flat structure).

Algorithm 1 algorithm to compute elements of $\mathcal{G}_{k,\sigma}$ generating $\langle s_0, \dots, s_i \rangle$

Require: $\langle s_0, \dots, s_i \rangle$ where $\forall i, s_i \in \Sigma^+$ and k

1: $j \leftarrow 0$

2: **repeat**

3: $C_j = \{s_0, \dots, s_j\} \setminus \setminus$ try C_j as a characteristic sample

4: associate a flat structure with FA internal nodes to every element of C_j

5: apply BP algorithm to find the set $R_{j,k} \subset \mathcal{G}_k$ of CGs compatible with this set

6: discard all elements of $R_{j,k}$ whose string language does not include $\{s_{j+1}, \dots, s_i\}$

7: $j \leftarrow j + 1$

8: **until** $j = i + 1$ OR $R_{j,k} \neq \emptyset$

Ensure: $R_{j,k}$: a set of CGs in $\mathcal{G}_{k,\sigma}$ whose string language includes $\langle s_0, \dots, s_i \rangle$

Elements of $\mathcal{G}_{k,\sigma}$ are obtained by applying unifying substitutions to k -valued CG, so they are also at most k -valued. But the k needed by Algorithm 1 may be greater than the one needed by the BP algorithm. Grammars in $\mathcal{G}_{k,\sigma}$ can be considered as having a special normal form: they produce only flat trees or trees that are the result of adjunctions on flat trees (cf Example 8). Theorem 5 ensures that as soon as the input includes a set of strings corresponding to a characteristic sample of flat structures with FA nodes (which will always occur in the limit), then for some j , the set $R_{j,k}$ contains at least one CG generating the target language. Inclusion tests are still necessary to select one grammar.

5 Conclusion

To conclude, this study shows that the domain of regular grammatical inference and the domain of CGs learning can be integrated into a unified framework. The first benefits of this unification is the translation of results from one domain to the other one with very few efforts, and a better understanding of the nature of the generalization operator used in the BP algorithm and Kanazawa's work.

Another less expected result is the introduction of a new class of automata: the class of RA, which naturally extends the class of FSA and allows to represent unidirectional CGs. Unidirectional CGs are better adapted to represent CF-languages for inference from strings than general CGs, because they are more constrained but generate the same class of languages. The class $\mathcal{G}_{k,\sigma}$ is even more interesting because, for each element of this class, there exists a characteristic set of flat structural examples with FA internal nodes only. So, learning this class from strings is not more expansive than learning it from structures. Of course, the expressivity of this class, which is still not known, should be characterized more precisely.

This study shows that the inference of CF-grammars may not be so different from the inference of regular grammars as it first seemed. But a lot remains to be done. A natural perspective is the adaptation of algorithms learning regular languages from positive and negative examples (such as RPNI [13], or Delete [7]) to CF-languages. This requires to define a canonical target. Another possible perspective concerns the adaptation of this work to Lambek grammars.

References

1. D. Angluin. Inference of Reversible Languages *Journal of the ACM* 3: 741–765, 1982.
2. Y. Bar Hillel and C. Gaifman and E. Shamir. On Categorical and Phrase Structure Grammars. *Bulletin of the Research Council of Israel*, 9F, 1960.
3. J. Besombes and J-Y. Marion. newblock Learning Reversible Categorical Grammars from Structures newblock proceedings of *Categorical Grammars* 148–163, 2004.
4. W. Buszkowski and G. Penn. Categorical grammars determined from linguistic data by unification, newblock *Studia Logica*, p. 431–454, 1990.
5. C. Costa-Florencio. Consistent Identification in the Limit of Any of the Classes k -valued Is NP-hard. proceedings of *LACL*, 125–134, LNAI 2099, 2001.
6. C. Costa-Florencio. Consistent Identification in the Limit of Rigid Grammars from Strings Is NP-hard. proceedings of the *IGGI: Algorithms and Applications*, 49–62, LNAI 2484, 2002.
7. F. Denis and A. Lemay and A. Terlutte. Some language classes identifiable in the limit from positive data. proceedings of the *ICGI: Algorithms and Applications*, 63–76, LNAI 2484, 2002.
8. P. Dupont and L. Miclet and E. Vidal. What is the search space of the regular inference. proceedings of *ICGI*, 25–37, LNCS 862, 1994.
9. E.M. Gold. Language identification in the limit. *Information and Control*, 10: 447–474, 1967.
10. G. Huet and C. Retore Survey of a few fundamental representation structures for computational linguistics *ESSLI 2002 lecture*.
11. A. Joshi and Y. Schabes. Tree-Adjoining Grammars. *Handbook of Formal Languages*, 3:69–120. Springer, 1997.
12. M. Kanazawa. *Learnable Classes of Categorical Grammars*, CSLI Publications. 1998.
13. J. Oncina and P. Garcia. *Identifying regular languages in polynomial time*, In *Advances in Structural and Syntactic Pattern Recognition*, vol.5: 99–108, World Scientific, 1992.

The Expressive Power of Restricted Fragments of English

Allan Third

School of Computer Science, University of Manchester, UK

`allan.third@cs.man.ac.uk`

Abstract. Taking the notion of *expressive power* of a language to mean its ability to distinguish different situations, we define four simple fragments of English based on the syntactic constructions they contain, and characterise their expressive power via translation into first-order logic. We also describe the circumstances under which an arbitrary first-order formula can be translated back into an English sentence of each fragment. This work is an extension of the semantic complexity results of Pratt-Hartmann [2], and Pratt-Hartmann and Third [3].

In this paper, we consider some simple syntactic constructions of English, and attempt to describe their expressive power. That is to say, we aim to answer the following question: when can two situations be distinguished by sentences containing, say, relative clauses, which cannot be distinguished without them?

In order to reduce the problem to manageable proportions, since we cannot look at English as a whole, we restrict our attention to small *fragments* – sets of sentences constructed from a fixed, limited grammar and lexicon, where each sentence is associated with an expression of first-order logic representing its truth conditions. Each such fragment of English thus defines a fragment of first-order logic, whose expressive power can be characterised model-theoretically. By varying the syntactic constructions used to define fragments, we can isolate the contributions of those constructions to the range of expressible meanings.

In [2] and [3], the expressive power of a variety of fragments of English was investigated, with the measure of expressivity being the computational complexity of deciding whether a given set of sentences in each fragment is logically consistent. In the present paper, we revisit some of these fragments, and provide alternative, more fine-grained characterisations of expressive power in terms of relations between the structures interpreting each fragment. Specifically, we consider Cop, the simplest fragment given in [2] and [3], and examine how its expressive power varies when extended with transitive verbs and relative clauses. In doing so, we assume some basic familiarity with the syntax of natural language, and the grammar of English, first-order logic and its model theory, and some ideas from the theory of computational complexity.

We use the following terminology throughout. For each fragment F of English, let an F -sentence be an element of F , and let an F -formula be the translation of an F -sentence into first-order logic. Let F also refer ambiguously (but harmlessly

so) to the corresponding fragment of logic. By the *expressive power* of F , we mean the ability of sentences of F to distinguish different situations. In model-theoretic terms, given two structures \mathfrak{A} and \mathfrak{B} , when do \mathfrak{A} and \mathfrak{B} have the same F -theory (make precisely the same F -formulae true), and when do they not? This question can be answered by giving a relation between pairs of structures which preserves truth of F -formulae – a relation which plays much the same role that bisimulation plays for modal logic (see, for example, [1]). We would therefore also like to establish invariance results similar to the van Benthem Characterisation Theorem for modal logic, which states that a first-order formula ϕ is equivalent to a modal formula if and only if the truth of ϕ is preserved by bisimulation. Such a result for a fragment F of English would in effect describe the circumstances under which it is possible to express a first-order formula using sentences of F .

1 The Syllogistic Fragment: Cop

We begin with the fragment named Cop in [3], which to all intents and purposes is the language of the traditional syllogism. The following set of semantically annotated phrase structure rules provides a formal definition of Cop.

Content lexicon

$N/\lambda x[man(x)] \rightarrow \text{man}$	$\text{PropN}/\lambda p[p(socrates)] \rightarrow \text{Socrates}$
$N/\lambda x[mortal(x)] \rightarrow \text{mortal}$	$\text{PropN}/\lambda p[p(diogenes)] \rightarrow \text{Diogenes}$
...	...

Syntax

Formal lexicon

$IP/\phi(\psi) \rightarrow NP/\phi, I'/\psi$	$\text{Det}/\lambda p\lambda q[\exists x(p(x) \wedge q(x))] \rightarrow \text{some}$
$I'/\phi \rightarrow \text{is a } N'/\phi$	$\text{Det}/\lambda p\lambda q[\forall x(p(x) \rightarrow q(x))] \rightarrow \text{every}$
$I'/\neg\phi \rightarrow \text{is not a } N'/\phi$	$\text{Det}/\lambda p\lambda q[\forall x(p(x) \rightarrow \neg q(x))] \rightarrow \text{no}$
$NP/\phi \rightarrow \text{PropN}/\phi$	
$NP/\phi(\psi) \rightarrow \text{Det}/\phi, N'/\psi$	
$N'/\phi \rightarrow N/\phi$	

The expressions to the right of the obliques represent the semantics of the phrases with which they are associated, with $\phi(\psi)$ indicating order of composition. A particular choice of content lexicon corresponds to the selection of a signature for the corresponding first-order language; the above grammar thus defines a family of fragments. Where we refer to “the fragment Cop” in the absence of a contextually salient content lexicon, we mean the union of all such fragments. (The name “Cop” refers to the common grammatical feature of all sentences in this fragment: the copula “is”.)

It is straightforward to verify that the above set of rules generates all and only sentences of the forms

$c \text{ is (not) a } p$	$\text{Some } p \text{ is (not) a } q$
$\text{Every } p \text{ is (not) a } q$	$\text{No } p \text{ is (not) a } q$

and that it generates all and only the formulae

$$\pm p(c) \quad \forall x(p(x) \rightarrow \pm q(x)) \quad \exists x(p(x) \wedge \pm q(x))$$

corresponding to the expected semantics of each sentence.

We take such semantic assignments to be uncontroversial. The occurrence of precisely the above translations as exercises in countless introductory logic courses provides sufficient evidence for their adequacy.

The following result was proved in [2]:

Theorem 1. *The satisfiability of a set E of sentences of Cop can be decided in deterministic polynomial time, where the complexity measure is the number of symbols in E .*

Thus, deciding which sets of Cop-sentences represent logically possible situations – equivalently, which syllogistic arguments are valid – is a tractable problem. This result, however, provides little insight into what can or cannot be expressed in Cop. We now offer an alternative characterisation of the expressive power of this fragment.

Suppose we are dealing with some fixed content lexicon, corresponding to a signature $S = (C, P)$ of constants C and unary predicates P . Let \mathfrak{A} be a structure interpreting S , and let $\text{tp}^{\mathfrak{A}}[a_1, \dots, a_n]$ be the n -type of a_1, \dots, a_n in \mathfrak{A} for all $a_1, \dots, a_n \in A$.

Definition 1. *Let the Cop-configuration of \mathfrak{A} over S be the function $\text{cop}^{\mathfrak{A}} : P \times P \rightarrow \{1, 2, 3, 4, 5\}$ defined as follows for $p, q \in P$.*

$$\begin{aligned} \text{cop}^{\mathfrak{A}}(p, q) &= 1 \text{ if } p^{\mathfrak{A}} = q^{\mathfrak{A}} \\ &2 \text{ if } p^{\mathfrak{A}} \subsetneq q^{\mathfrak{A}} \\ &3 \text{ if } q^{\mathfrak{A}} \subsetneq p^{\mathfrak{A}} \\ &4 \text{ if } p^{\mathfrak{A}} \cap q^{\mathfrak{A}} \neq \emptyset, q^{\mathfrak{A}} \setminus p^{\mathfrak{A}} \neq \emptyset \text{ and } p^{\mathfrak{A}} \setminus q^{\mathfrak{A}} \neq \emptyset \\ &5 \text{ otherwise} \end{aligned}$$

If \mathfrak{A} and \mathfrak{B} are structures interpreting S , we say that \mathfrak{A} and \mathfrak{B} are Cop-similar, written $\mathfrak{A} \sim_{\text{Cop}} \mathfrak{B}$, if the following conditions hold

1. $\text{cop}^{\mathfrak{A}} = \text{cop}^{\mathfrak{B}}$,
2. for every constant $c \in C$, $\text{tp}^{\mathfrak{A}}[c^{\mathfrak{A}}] = \text{tp}^{\mathfrak{B}}[c^{\mathfrak{B}}]$.

Theorem 2. *Let \mathfrak{A} and \mathfrak{B} be structures interpreting S . Then $\mathfrak{A} \sim_{\text{Cop}} \mathfrak{B}$ if and only if \mathfrak{A} and \mathfrak{B} have the same Cop theory over S .*

Proof. Suppose that $\mathfrak{A} \sim_{\text{Cop}} \mathfrak{B}$. By (2), \mathfrak{A} and \mathfrak{B} agree on all ground Cop-formulae. Let ϕ be a non-ground Cop-formula. By considering each possibility for ϕ in turn, the fact that $\text{cop}^{\mathfrak{A}} = \text{cop}^{\mathfrak{B}}$ guarantees that \mathfrak{A} and \mathfrak{B} agree on the truth of ϕ , and thus \mathfrak{A} and \mathfrak{B} have the same Cop theory. The details are straightforward.

Conversely, suppose either that for some pair of unary predicates $p, q \in P$, we have $\text{cop}^{\mathfrak{A}}(p, q) \neq \text{cop}^{\mathfrak{B}}(p, q)$, or that for some constant c , $\text{tp}^{\mathfrak{A}}[c^{\mathfrak{A}}] \neq \text{tp}^{\mathfrak{B}}[c^{\mathfrak{B}}]$.

In the latter case, there exists some $p \in P$ such that $\mathfrak{A} \models p[c^{\mathfrak{A}}]$ and $\mathfrak{B} \not\models p[c^{\mathfrak{B}}]$, or vice versa, in which case the Cop-formula $p(c)$ is true in one of \mathfrak{A} and \mathfrak{B} , and false in the other.

For the former case, we show that for each possible pair of values of $\text{cop}^{\mathfrak{A}}(p, q)$ and $\text{cop}^{\mathfrak{B}}(p, q)$, we can construct a Cop-formula true in \mathfrak{A} and false in \mathfrak{B} . The proof is routine; for brevity, we describe only a single case here.

$\text{cop}^{\mathfrak{A}}(p, q) = 1, \text{cop}^{\mathfrak{B}}(p, q) = 2$: We have that $p^{\mathfrak{A}} = q^{\mathfrak{A}}, p^{\mathfrak{B}} \subseteq q^{\mathfrak{B}}$, and $q^{\mathfrak{B}} \setminus p^{\mathfrak{B}} \neq \emptyset$, and so the Cop-formula $\forall x(q(x) \rightarrow p(x))$ is true in \mathfrak{A} and not \mathfrak{B} . \square

Thus the fragment Cop is capable of distinguishing between two structures if and only if they are Cop-dissimilar. As Cop-similarity is an easy condition to check, we can use it to prove inexpressibility results, such as the following.

Corollary 1. *Let P, Q, R be sets. No Cop-formula (or set of Cop-formulae) is equivalent to $P \cap R = Q \cap R$.*

Proof. Define two structures \mathfrak{A} and \mathfrak{B} over the domain $\{a_1, \dots, a_5\}$, interpreting only p, q and r as follows

$$\begin{array}{ll} p^{\mathfrak{A}} = p^{\mathfrak{B}} = \{a_1, a_2\} & r^{\mathfrak{A}} = \{a_2, a_5\} \\ q^{\mathfrak{A}} = q^{\mathfrak{B}} = \{a_2, a_3, a_4\} & r^{\mathfrak{B}} = \{a_2, a_3, a_5\} \end{array}$$

and consider the Cop-fragment generated by a content lexicon containing common nouns corresponding to each of p, q and r . Neither \mathfrak{A} nor \mathfrak{B} interpret any constants, and it is easy to check that $\text{cop}^{\mathfrak{A}}$ and $\text{cop}^{\mathfrak{B}}$ have the same value for every combination of pairs from $\{p, q, r\}$, and so $\mathfrak{A} \sim_{\text{Cop}} \mathfrak{B}$. But $p^{\mathfrak{A}} \cap r^{\mathfrak{A}} = q^{\mathfrak{A}} \cap r^{\mathfrak{A}}$, and $p^{\mathfrak{B}} \cap r^{\mathfrak{B}} \neq q^{\mathfrak{B}} \cap r^{\mathfrak{B}}$. By Theorem 2, \mathfrak{A} and \mathfrak{B} make the same Cop-formulae true, and the result follows. \square

Theorem 2 thus answers, for Cop, the first of the questions we posed earlier: under what circumstances do two structures make the same sentences true? Does this result allow us then to achieve the second of our goals, and prove an Invariance Theorem for Cop similar to the van Benthem Characterisation Theorem for modal logic? That is, is it true that an arbitrary formula ϕ is equivalent to a Cop-formula if and only if ϕ is invariant for Cop-simulation? (By “invariant for Cop-simulation”, we mean that any pair of Cop-similar structures assign the same truth value to ϕ .)

Unfortunately, as the following example shows, the answer is no.

Example 1. Let c be a constant and p, q unary predicates. The formula $p(c) \vee q(c)$ is easily shown to be invariant for Cop-simulation, and yet is not equivalent to any Cop-formula. To see this second claim, observe that every Cop-formula is Horn, and so has a “least true” Herbrand model; $p(c) \vee q(c)$ is non-Horn, and it is straightforward to construct two equally-minimal Herbrand models of it.

In fact, this example reveals a more general difficulty: it turns out that there is no possible refinement of Cop-simulation which will allow us to prove the desired kind of result. For, suppose there is some relation \sim on structures such that a formula ϕ is invariant for \sim if and only if ϕ is equivalent to a Cop-formula.

Then, taking c to be a constant, and p, q to be unary predicates as above, if $\mathfrak{A}, \mathfrak{B}$ are any pair of structures such that $\mathfrak{A} \sim \mathfrak{B}$, it must certainly be the case that $\mathfrak{A} \models p[c^{\mathfrak{A}}]$ iff $\mathfrak{B} \models p[c^{\mathfrak{B}}]$ and that $\mathfrak{A} \models q[c^{\mathfrak{A}}]$ iff $\mathfrak{B} \models q[c^{\mathfrak{B}}]$. But then it follows that $\mathfrak{A} \models p[c^{\mathfrak{A}}] \vee q[c^{\mathfrak{A}}]$ iff $\mathfrak{B} \models p[c^{\mathfrak{B}}] \vee q[c^{\mathfrak{B}}]$. Hence $p(c) \vee q(c)$ is \sim -invariant – a contradiction, and so no such relation can exist.

A somewhat weaker result is possible, however. Let Cop^* be the fragment of English (and hence of logic) obtained by adding the following rules to the grammar of Cop:

$$\begin{array}{l} \text{IP}/\phi \wedge \psi \rightarrow \text{IP}/\phi, \text{ and, IP}/\psi \\ \text{IP}/\phi \vee \psi \rightarrow \text{IP}/\phi, \text{ or, IP}/\psi \end{array}$$

That is to say, Cop^* is the result of closing Cop under Boolean combinations of sentences. (Sentence negation was already present in Cop.)

Theorem 3. *A first-order formula ϕ is equivalent to a formula in Cop^* if and only if ϕ is invariant for Cop-simulation.*

Proof. (Essentially the same as the proof of the van Benthem Characterisation Theorem – see, e.g., [1],[4].) Theorem 2 guarantees that any formula equivalent to a Cop-formula is invariant for \sim_{Cop} , and so any formula equivalent to a Boolean combination of Cop-formulae (i.e., any Cop^* -formula) must also be invariant for \sim_{Cop} .

To show the converse, suppose that ϕ is invariant for Cop-simulation, and let $\Phi = \{\psi \mid \psi \text{ a Cop}^*\text{-formula, } \phi \models \psi\}$. If we can show that $\Phi \models \phi$, then by compactness, there exists some finite subset $X \subseteq \Phi$ such that $\models \bigwedge X \rightarrow \phi$. By construction of Φ , $\models \phi \rightarrow \bigwedge X$, and so $\models \phi \leftrightarrow \bigwedge X$ – that is, ϕ is equivalent to a conjunction of Cop^{*}-formulae, which must itself be a Cop^{*}-formula.

Now, to show that $\Phi \models \phi$, suppose that Φ is consistent (otherwise we have trivially that $\Phi \models \phi$), and let \mathfrak{A} be a model of Φ . Let $T = \{\psi \mid \psi \text{ a Cop}^*\text{-formula and } \mathfrak{A} \models \psi\}$. We show that $T \cup \{\phi\}$ is consistent. For, suppose it was not. Then, by compactness, for some finite subset $X \subseteq T$, $\models \phi \rightarrow \neg \bigwedge X$. Now, by moving negations inward, we can see that $\neg \bigwedge X$ is logically equivalent to a Cop^{*}-formula χ (since Cop^{*} is closed under Boolean operators) which is a member of Φ . But then $\mathfrak{A} \models \neg \bigwedge X$, contradicting $X \subseteq T$, and $\mathfrak{A} \models T$. So $T \cup \{\phi\}$ is consistent.

Let \mathfrak{B} be any model of $T \cup \{\phi\}$, and let ψ be any Cop^{*}-formula. If $\mathfrak{A} \models \psi$, then $\psi \in T$, and so $\mathfrak{B} \models \psi$. Likewise, if $\mathfrak{A} \models \neg\psi$, then $\neg\psi$ is logically equivalent to an element of T , and so $\mathfrak{B} \models \neg\psi$. Thus \mathfrak{A} and \mathfrak{B} have the same Cop^{*} theory, and so the same Cop theory, and, by Theorem 2, $\mathfrak{A} \sim_{\text{Cop}} \mathfrak{B}$. Since $\mathfrak{B} \models \phi$ and ϕ is invariant for Cop-simulation, $\mathfrak{A} \models \phi$, and so $\Phi \models \phi$ as required. \square

Note that the extra rules added for Cop^{*} are far from logically harmless: in particular, it is easy to show that Cop^{*} has an NP-complete satisfiability problem, compared to the PTIME result for Cop. However, as we saw, without this extra complexity, a result such as Theorem 3 is not possible.

2 Transitive Verbs

We now proceed to extend Cop, beginning by adding new phrase-structure rules in order to allow sentences containing transitive verbs, such as *admire*.

Syntax	Formal Lexicon
$I'/\phi \rightarrow VP/\phi$	$Neg \rightarrow \text{does not}$
$I'/\phi \rightarrow NegP/\phi$	
$NegP/\neg\phi \rightarrow Neg, VP/\phi$	Content Lexicon
$VP/\phi(\psi) \rightarrow TV/\phi, NP/\psi$	
	$TV/\lambda s\lambda x[s(\lambda y[admire(x, y)])] \rightarrow \text{admires}$
	$TV/\lambda s\lambda x[s(\lambda y[despise(x, y)])] \rightarrow \text{despises}$
	...

In order to keep the grammar simple, we have ignored surface syntactic issues of verb-inflection and the replacement of *some* with *any* in negative contexts, neither of which affect the semantics.

Let Cop+TV be the fragment of English generated by the union of the above rules with those of Cop. Via the same process of semantic annotation as before, Cop+TV also defines a fragment of first-order logic over the signature generated by the content lexicon. Thus a sentence such as *Every man admires some philosopher* is assigned the semantics

$$\forall x(man(x) \rightarrow \exists y(philosopher(y) \wedge admire(x, y))).$$

Cop+TV is more expressive than Cop. To see this, let \mathfrak{A}' , \mathfrak{B}' be the structures \mathfrak{A} and \mathfrak{B} , respectively, from the proof of Corollary 1, extended to interpret a relation t by $t^{\mathfrak{A}'} = \emptyset$ and $t^{\mathfrak{B}'} = \{(a_1, a_2)\}$. It is trivial to show that \mathfrak{A}' and \mathfrak{B}' have the same Cop theory, but the Cop+TV-formula $\forall x(p(x) \rightarrow \forall y(p(y) \rightarrow \neg t(x, y)))$ is true in \mathfrak{A}' and false in \mathfrak{B}' . Despite this increase in expressive power, the following result was shown in [3].

Theorem 4. *The satisfiability of a set E of sentences of Cop+TV can be decided in deterministic polynomial time.*

The fact that expressivity can be increased without a corresponding increase in semantic complexity demonstrates that complexity is too coarse a measure of expressive power. As in the case of Cop, we define a relation between structures preserving truth of Cop+TV-formulae.

Definition 2. *Let $S = (C, P, R)$ be a signature consisting of constants C , unary predicates P and binary relations R , and let \mathfrak{A} be a structure interpreting S . Let the TV-configuration of \mathfrak{A} over S be the function $tv^{\mathfrak{A}} : P \times P \times R \rightarrow \{1, 2, 3, 4, 5, 6\}$ defined as follows for $p, q \in P$, $r \in R$.*

$\text{tv}^{\mathfrak{A}}(p, q, r) =$

- 1 if $(p^{\mathfrak{A}} \times q^{\mathfrak{A}}) \cap r^{\mathfrak{A}} = \emptyset$
- 2 if $p^{\mathfrak{A}} \times q^{\mathfrak{A}} \subseteq r^{\mathfrak{A}}$ and $p^{\mathfrak{A}} \times q^{\mathfrak{A}} \neq \emptyset$
- 3 if $p^{\mathfrak{A}} \neq \emptyset$ and for every $a_1 \in p^{\mathfrak{A}}$, there exist $a_2, a_3 \in q^{\mathfrak{A}}$ such that $(a_1, a_2) \in r^{\mathfrak{A}}$, $(a_1, a_3) \notin r^{\mathfrak{A}}$
- 4 if $q^{\mathfrak{A}} \neq \emptyset$, there exists $a_1 \in p^{\mathfrak{A}}$ such that $(\{a_1\} \times q^{\mathfrak{A}}) \cap r^{\mathfrak{A}} = \emptyset$ and there exists $a_2 \in p^{\mathfrak{A}}$ such that $\{a_2\} \times q^{\mathfrak{A}} \subseteq r^{\mathfrak{A}}$
- 5 if $(p^{\mathfrak{A}} \times q^{\mathfrak{A}}) \cap r^{\mathfrak{A}} \neq \emptyset$, there exists $a_1 \in p^{\mathfrak{A}}$ such that $(\{a_1\} \times q^{\mathfrak{A}}) \cap r^{\mathfrak{A}} = \emptyset$ and for every $a_2 \in p^{\mathfrak{A}}$, there exists $a_3 \in q^{\mathfrak{A}}$ such that $(a_2, a_3) \notin r^{\mathfrak{A}}$
- 6 otherwise

If \mathfrak{A} and \mathfrak{B} are structures interpreting S , we say that \mathfrak{A} and \mathfrak{B} are TV-similar, written $\mathfrak{A} \sim_{\text{TV}} \mathfrak{B}$, if

1. $\text{tv}^{\mathfrak{A}} = \text{tv}^{\mathfrak{B}}$,
2. for every pair of constants c, d , $\text{tp}^{\mathfrak{A}}[c^{\mathfrak{A}}, d^{\mathfrak{A}}] = \text{tp}^{\mathfrak{B}}[c^{\mathfrak{B}}, d^{\mathfrak{B}}]$,
3. for every $c \in C$, $p \in P$ and $r \in R$:
 - (a) there exists $a \in A$ such that $\mathfrak{A} \models p[a] \wedge r[a, c^{\mathfrak{A}}]$ iff there exists $b \in B$ such that $\mathfrak{B} \models p[b] \wedge r[b, c^{\mathfrak{B}}]$,
 - (b) there exists $a \in A$ such that $\mathfrak{A} \models p[a] \wedge r[c^{\mathfrak{A}}, a]$ iff there exists $b \in B$ such that $\mathfrak{B} \models p[b] \wedge r[c^{\mathfrak{B}}, b]$, and
4. $\mathfrak{A} \sim_{\text{Cop}} \mathfrak{B}$.

Theorem 5. Let \mathfrak{A} and \mathfrak{B} be structures interpreting $S = (C, P, R)$. Then $\mathfrak{A} \sim_{\text{TV}} \mathfrak{B}$ if and only if \mathfrak{A} and \mathfrak{B} have the same Cop+TV theory over S .

Proof. Essentially the same as that for Theorem 2. □

Since the Cop-formulae in Example 1 are also Cop+TV-formulae, and every Cop+TV-formula is Horn, the same difficulty arises as before, and so in order to prove any kind of Invariance Theorem, we are forced to add sentence (IP) coordination via the same phrase-structure rules as earlier, to produce a fragment Cop+TV* closed under Boolean combinations of sentences. The following result then follows by a near identical argument to that for Theorem 3.

Theorem 6. A first-order formula ϕ is equivalent to a Cop+TV*-formula if and only if ϕ is invariant for TV-simulation.

As before, the addition of sentence coordination leads to an increase in complexity: Cop+TV* has an NP-complete satisfiability problem.

3 Relative Clauses

Let us now consider what happens when we extend the fragment Cop in a different way, by adding restrictive relative clauses. We show again that, as might be expected, such an extension leads to an increase in expressive power.

The following rules, when added to those of Cop, generate sentences containing relative clauses:

Syntax	Formal lexicon
$N'/\phi(\psi) \rightarrow N/\psi, CP/\phi$	$C \rightarrow$
$CP/\phi(\psi) \rightarrow CSpec_t/\phi, C'_t/\psi$	$RelPro/\lambda q\lambda p\lambda x[p(x) \wedge q(x)] \rightarrow \text{who},$
$C'_t/\lambda t[\phi] \rightarrow C, IP/\phi$	which
$NP/\phi \rightarrow RelPro/\phi$	
$CSpec_t \rightarrow$	

To ensure correct English word-order, wh-movement must be applied to sentences (IPs) generated by these rules, so that: (i) every RelPro moves into the nearest CSpec_t which c-commands it; (ii) every CSpec_t is filled by a moved RelPro; and (iii) every NP position vacated by a RelPro moving to CSpec_t is filled by a trace t with semantic value $\lambda p.p(t)$. (In fact, a simple fragment such as this one could be specified just as precisely without any reference to wh-movement. For the sake of extensibility, however, we prefer the more general approach.)

In what follows, we assume that all nouns are animate, and use only *who*, not *which*; the issue of agreement between relative pronouns and their antecedents does not affect the semantics.

Let Cop+Rel be the fragment of English (and hence also the fragment of first-order logic) generated by the combination of the above rules, the rules of Cop and the process of wh-movement. An example of a sentence in Cop+Rel is *Every man who is a stoic is a philosopher who is not a cynic*, which is assigned the semantics

$$\forall x(man(x) \wedge stoic(x) \rightarrow philosopher(x) \wedge \neg cynic(x)).$$

The following complexity result was shown in [2]:

Theorem 7. *The satisfiability problem for a set E of sentences of Cop+Rel is NP-complete.*

This increase in computational complexity is accompanied by a genuine increase over the expressive power of Cop: using the content lexicon from the proof of Corollary 1, it is clear that $P \cap R = Q \cap R$ is expressible in Cop+Rel by the pair of sentences *Every p who is an r is a q* and *Every q who is an r is a p*. Note also that the expressive powers of Cop+Rel and Cop+TV are different: it is straightforward to build pairs of structures which Cop+Rel can distinguish and Cop+TV cannot, and vice versa.

As before, we now define a relation on structures corresponding to preservation of truth in Cop+Rel. Suppose again that $S = (C, P)$ is a signature of constants C and unary predicates P . In order to simplify matters slightly, we assume that P contains a special predicate *thing*, interpreted in every structure \mathfrak{A} so that $thing^{\mathfrak{A}} = A$.

Definition 3. *Let \mathfrak{A} and \mathfrak{B} be structures interpreting S . We say that \mathfrak{A} and \mathfrak{B} are Cop+Rel-similar, written $\mathfrak{A} \sim_{Rel} \mathfrak{B}$, if*

1. for every constant c , $tp^{\mathfrak{A}}[c^{\mathfrak{A}}] = tp^{\mathfrak{B}}[c^{\mathfrak{B}}]$,
2. for every $a \in A$, there exists $b \in B$ such that $tp^{\mathfrak{A}}[a] = tp^{\mathfrak{B}}[b]$,
3. for every $b \in B$, there exists $a \in A$ such that $tp^{\mathfrak{A}}[a] = tp^{\mathfrak{B}}[b]$.

Theorem 8. *Let \mathfrak{A} and \mathfrak{B} be structures interpreting S , as above. Then $\mathfrak{A} \sim_{\text{Rel}} \mathfrak{B}$ if and only if \mathfrak{A} and \mathfrak{B} have the same Cop+Rel theory over S .*

Proof. Suppose that $\mathfrak{A} \sim_{\text{Rel}} \mathfrak{B}$. We show that for every Cop+Rel-formula ϕ , if $\mathfrak{A} \models \phi$, then $\mathfrak{B} \models \phi$. Since Cop+Rel contains sentence negation, it follows that if $\mathfrak{B} \models \phi$, $\mathfrak{A} \models \phi$.

The truth of Cop+Rel sentences containing proper nouns is evidently preserved by clause (1) in Definition 3. Each N' -phrase occurring in a Cop+Rel sentence not containing a proper noun contributes a subformula of the form $p(x)$, or $p(x) \wedge \Pi(x)$ to the semantics of each sentence in which it occurs, where p is some unary predicate, and $\Pi(x)$ is a (possibly empty) Boolean combination of atoms $q(x)$. Call any such formula an N' -formula. Every such Cop+Rel sentence therefore translates to a formula of one of the forms $\forall x(\psi_1(x) \rightarrow \pm\psi_2(x))$ or $\exists x(\psi_1(x) \wedge \pm\psi_2(x))$, where $\psi_1(x)$ and $\psi_2(x)$ are N' -formulae. Since \mathfrak{A} and \mathfrak{B} realise the same 1-types, it is straightforward to show that \mathfrak{A} and \mathfrak{B} agree on the truth value of every Cop+Rel-formula.

Conversely, suppose now that $\mathfrak{A} \not\sim_{\text{Rel}} \mathfrak{B}$. Then one of (1), (2) or (3) in Definition 3 fails. We consider each case in turn.

1. For some constant c , $tp^{\mathfrak{A}}[c^{\mathfrak{A}}] \neq tp^{\mathfrak{B}}[c^{\mathfrak{B}}]$. As in the proof of Theorem 2, there exists a ground Cop-formula (and hence Cop+Rel formula) ϕ such that $\mathfrak{A} \models \phi$ and $\mathfrak{B} \not\models \phi$.
2. There exists $a \in A$ such that for every $b \in B$, $tp^{\mathfrak{A}}[a] \neq tp^{\mathfrak{B}}[b]$. Then for every $b \in B$, there exists a unary predicate p_b such that $\mathfrak{A} \models p_b[a]$ iff $\mathfrak{B} \models \neg p_b[b]$. Let $P_a^+ = \{p^+ \in P \mid \mathfrak{A} \models p^+[a]\}$, and let $P_a^- = P \setminus P_a^+$. We know that $P_a^+ \neq \emptyset$, since $\text{thing} \in P_a^+$, and so the sentence

Some thing $\{\text{who is a } p^+\}_{p^+ \in P_a^+} \{\text{who is not a } p^-\}_{p^- \in P_a^-}$ is a thing

is true in \mathfrak{A} but not in \mathfrak{B} .

3. Similar to the previous case.

Thus if $\mathfrak{A} \not\sim_{\text{Rel}} \mathfrak{B}$, there exist Cop+Rel-formulae true in \mathfrak{A} and false in \mathfrak{B} , and so the theorem holds. \square

Let Cop+Rel* be the fragment generated by the grammar of Cop+Rel (including wh-movement), augmented with sentence coordination in the usual way. Since Cop+Rel-formulae are not in general Horn, the argument of Example 1 does not apply. However, as yet we do not have an invariance result for Cop+Rel without sentence coordination.

It is straightforward to verify that Cop+Rel*-sentences containing embedded IP coordination, such as *Every man who is a stoic or is a cynic is a philosopher*, can always be simulated by coordination of complete Cop+Rel-sentences, and thus

Cop+Rel^* is (semantically) just the result of closing Cop+Rel under Boolean combinations of sentences.

We again obtain an Invariance Theorem by essentially the same argument as for Theorems 3 and 6.

Theorem 9. *A first-order formula ϕ is equivalent to a Cop+Rel^* -formula if and only if it is invariant for Cop+Rel -simulation.*

Note that, unlike in the earlier cases, the formation of Cop+Rel^* from Cop+Rel does not affect the computational complexity: the satisfiability problem for Cop+Rel^* is also NP-complete. In general, the complexity of any fragment with an NP-hard satisfiability problem will be unaffected by the addition of sentence coordination.

4 Transitive Verbs and Relative Clauses

Finally, let us consider what happens when we take the union of all three of the preceding fragments. Let Cop+Rel+TV be the fragment of English (and, as usual, of logic) generated by the union of the phrase structure rules for Cop , Cop+Rel and Cop+TV , along with the rule of *wh*-movement. In addition, for this final fragment, we assume from the beginning that we also allow sentence (IP) coordination – the analogues of the following results are not yet known in its absence.

Clearly, every pair of situations which can be distinguished by any of the fragments we have considered so far can also be distinguished by Cop+Rel+TV , and thus Cop+Rel+TV is strictly more expressive than all of the earlier fragments.

Cop+Rel+TV contains sentences such as *Every man who admires a cynic is a philosopher*, which is given the translation

$$\forall x(\text{man}(x) \wedge \exists y(\text{cynic}(y) \wedge \text{admire}(x, y)) \rightarrow \text{philosopher}(x)).$$

As presented here, Cop+Rel+TV also allows ungrammatical N' -phrases such as **man who* [_{IP} [_{IP} *admires a stoic*] and [_{IP} *a cynic despises*]]. However, it is easy to see that any such N' can be replaced with a grammatical equivalent, such as *man who admires a stoic and whom a cynic despises*. To shorten the presentation, we omit the (routine) details here.

Note that, due to the addition of sentence coordination, this fragment is a slight extension of the fragment named Cop+Rel+TV in [3]. Nonetheless the following result holds.

Theorem 10. *The satisfiability problem for a set E of sentences of Cop+Rel+TV is EXPTIME-complete.*

We characterise the expressive power of Cop+Rel+TV in the by-now familiar manner.

Definition 4. Let \mathfrak{A} and \mathfrak{B} be structures interpreting a signature $S = (C, P, R)$ consisting of constants C , unary predicates P and binary relations R . A Rel+TV-simulation $C \subseteq A \times B$ is a relation satisfying the following conditions:

1. for every constant c , $c^{\mathfrak{A}} C c^{\mathfrak{B}}$.
2. for all $a \in A, b \in B$ such that $a C b$, $\text{tp}^{\mathfrak{A}}[a] = \text{tp}^{\mathfrak{B}}[b]$,
3. for all $a \in A, b \in B$ such that $a C b$, and every constant c , $\text{tp}^{\mathfrak{A}}[a, c^{\mathfrak{A}}] = \text{tp}^{\mathfrak{B}}[b, c^{\mathfrak{B}}]$,
4. for all $a \in A, b \in B$ such that $a C b$, and every $r \in R$,
 - (a) if, for some $a' \in A$, $\mathfrak{A} \models \pm r[a, a']$, then for some $b' \in B$ such that $a' C b'$, $\mathfrak{B} \models \pm r[b, b']$,
 - (b) if, for some $a' \in A$, $\mathfrak{A} \models \pm r[a', a]$, then for some $b' \in B$ such that $a' C b'$, $\mathfrak{B} \models \pm r[b', b]$,
 - (c) if, for some $b' \in B$, $\mathfrak{B} \models \pm r[b, b']$, then for some $a' \in A$ such that $a' C b'$, $\mathfrak{A} \models \pm r[a, a']$,
 - (d) if, for some $b' \in B$, $\mathfrak{B} \models \pm r[b', b]$, then for some $a' \in A$ such that $a' C b'$, $\mathfrak{A} \models \pm r[a', a]$,
5. for all $a \in A$, there exists $b \in B$ such that $a C b$,
6. for all $b \in B$, there exists $a \in A$ such that $a C b$,

Structures \mathfrak{A} and \mathfrak{B} are Rel+TV-similar, written $\mathfrak{A} \sim_{\text{Rel+TV}} \mathfrak{B}$, if some $C \subseteq A \times B$ is a Rel+TV-simulation.

Lemma 1. If $\mathfrak{A} \sim_{\text{Rel+TV}} \mathfrak{B}$, then \mathfrak{A} and \mathfrak{B} agree on the truth values of all Cop+Rel+TV-formulae.

Proof. Let $C \subseteq A \times B$ be a Rel+TV simulation.

By (1) and (3), \mathfrak{A} and \mathfrak{B} agree on all ground Cop+Rel+TV-formulae.

As in the proof of Theorem 8, let an N' -formula be the subformula contributed by an N' -phrase to the semantics of a complete sentence. A simple structural induction shows that for all N' -formulae $\phi(x)$, if $a \in A, b \in B$ such that $a C b$, then $\mathfrak{A} \models \phi[a]$ iff $\mathfrak{B} \models \phi[b]$.

Now let ϕ be any non-ground Cop+Rel+TV-formula, and suppose that $\mathfrak{A} \models \phi$. In each case, $\mathfrak{B} \models \phi$. We give only one example here, for brevity – the rest are similar. Let $\psi_1(x)$ and $\psi_2(x)$ be N' -formulae.

$\phi = \forall x(\psi_1(x) \rightarrow \psi_2(x))$: For all $a \in A$, if $\mathfrak{A} \models \psi_1[a]$, it follows that $\mathfrak{A} \models \psi_2[a]$.

Suppose for some $b \in B$, $\mathfrak{B} \models \psi_1[b] \wedge \neg \psi_2[b]$. By (6), there exists $a \in A$ such that $a C b$ and, by the above induction, $\mathfrak{A} \models \psi_1[a] \wedge \neg \psi_2[a]$ – a contradiction.

So $\mathfrak{B} \models \phi$. □

Lemma 2. Let \mathfrak{A} and \mathfrak{B} be 2-saturated structures with the same Cop+Rel+TV theory. Then $\mathfrak{A} \sim_{\text{Rel+TV}} \mathfrak{B}$.

Proof. Define $C \subseteq A \times B$ as follows:

$$C = \{(a, b) \mid a \in A, b \in B, \text{ and for all } N'\text{-formulae } \phi(x), \\ \mathfrak{A} \models \phi[a] \text{ iff } \mathfrak{B} \models \phi[b]\}$$

We check that C is a Rel+TV-simulation.

1. Immediate, since \mathfrak{A} and \mathfrak{B} have the same Cop+Rel+TV theory.
2. Using the special predicate **thing** if necessary, any 1-type can be specified fully by an N' -formula in the fragment Cop+Rel. Thus for all $a \in A, b \in B$, if aCb , then $\text{tp}^{\mathfrak{A}}[a] = \text{tp}^{\mathfrak{B}}[b]$.
3. Let $a \in A$, and let c be a constant symbol. For every binary predicate r such that $\mathfrak{A} \models \pm r[a, c^{\mathfrak{A}}]$, $\text{thing}(x) \wedge \pm r(x, c)$ is an N' -formula satisfied by a in \mathfrak{A} , and hence for all $b \in B$ such that aCb , we have that $\mathfrak{B} \models \text{thing}[b] \wedge \pm r[b, c^{\mathfrak{B}}]$. A similar argument applies in the case $\mathfrak{A} \models \pm r[c^{\mathfrak{A}}, a]$. It then follows by (1) and (2) that $\text{tp}^{\mathfrak{A}}[a, c^{\mathfrak{A}}] = \text{tp}^{\mathfrak{B}}[b, c^{\mathfrak{B}}]$.
4. (We only consider the first case.) Suppose that for some $a \in A$, there exists a binary r such that $\mathfrak{A} \models \pm r[a, a']$ for some $a' \in A$. We show that for every $b \in B$ such that aCb , there exists $b' \in B$ such that $\mathfrak{B} \models \pm r[b, b']$ and $a'Cb'$. Let Φ be the set of N' -formulae satisfied by a' . For every finite sequence $\phi_1(x), \dots, \phi_n(x)$ of elements of Φ , a satisfies $\psi(x) = \exists y(\phi_1(y) \wedge \dots \wedge \phi_n(y) \wedge \pm r(x, y))$ in \mathfrak{A} . As $\psi(x)$ is an N' -formula, we have that for all $b \in B$ such that aCb , $\mathfrak{B} \models \psi[b]$. By 2-saturation of \mathfrak{B} , there exists $b' \in B$ such that $\mathfrak{B} \models \bigwedge_{\phi(y) \in \Phi} \phi[b'] \wedge \pm r[b, b']$. That is, since Φ is closed under negation, b' satisfies precisely the N' -formulae satisfied by a' , and hence $a'Cb'$.
5. Let $a \in A$, and let Φ be the set of N' -formulae satisfied by a in \mathfrak{A} . Then for every finite sequence of elements $\phi_1(x), \dots, \phi_n(x)$ of Φ , $\mathfrak{A} \models \phi_1[a] \wedge \dots \wedge \phi_n[a]$, and hence $\mathfrak{A} \models \exists x(\phi_1(x) \wedge \dots \wedge \phi_n(x))$ – a Cop+Rel+TV-formula, which must therefore be true in \mathfrak{B} also. So for each finite sequence of elements $\phi_1(x), \dots, \phi_n(x)$ of Φ , there exists $b \in B$ such that $\mathfrak{B} \models \phi_1[b] \wedge \dots \wedge \phi_n[b]$, and so by 2-saturation of \mathfrak{B} , there exists $b \in B$ such that $\mathfrak{B} \models \bigwedge_{\phi(x) \in \Phi} \phi[b]$. Since Φ is closed under negation, it follows that aCb .
6. Similar to (5).

Thus C is a Rel+TV-simulation, and so $\mathfrak{A} \sim_{\text{Rel+TV}} \mathfrak{B}$. □

In light of Lemmas 1 and 2, we can now prove

Theorem 11. *A first-order formula ϕ is equivalent to a formula in Cop+Rel+TV if and only if ϕ is invariant for Rel+TV-simulation.*

Proof. Essentially the same as the proof of Theorem 3, save that, having built structures \mathfrak{A} and \mathfrak{B} with the same Cop+Rel+TV theory, we cannot immediately conclude that $\mathfrak{A} \sim_{\text{Rel+TV}} \mathfrak{B}$. However, as every structure has a 2-saturated elementary extension (see, e.g., [4]), we are able to construct elementary extensions \mathfrak{A}^* and \mathfrak{B}^* of \mathfrak{A} and \mathfrak{B} respectively such that $\mathfrak{A}^* \sim_{\text{Rel}} \mathfrak{B}^*$, whence the rest of the proof proceeds as earlier. □

5 Conclusion

The results presented here extend the complexity results of [2] and [3] by providing semantic characterisations of the expressive power of four simple fragments

of English – the syllogistic fragment Cop, and extensions of Cop with transitive verbs and relative clauses. For each fragment, we defined a notion of simulation between pairs of structures such that structures are similar if and only if they are indistinguishable by that fragment. Using these simulations, we were also able to show when arbitrary first-order formulae can be translated back into (extensions of) each fragment.

The techniques used to obtain these results are fully general, and can be applied to arbitrary further extensions of Cop, or other fragments of English (or, of course, naturally-delineated fragments of any natural language.) We are currently working on filling in the missing results mentioned above – for example, Cop+Rel+TV *without* sentence coordination – as well as characterising the expressive power of extensions of Cop with anaphora, ditransitive verbs and various forms of sub-sentence coordination. In doing so, we aim to map out the contributions made by these syntactic constructions to the range of expressible meanings, and thus at least partially describe the relationship between the syntax and semantics of English.

References

1. Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal logic*. Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, 2001.
2. Ian Pratt-Hartmann. Fragments of language. *Journal of Logic, Language and Information*, 13:207–223, 2004.
3. Ian Pratt-Hartmann and Allan Third. More fragments of language. *Notre-Dame Journal of Formal Logic*, 2004. submitted.
4. Johan van Benthem. *Exploring logical dynamics*. CSLI Publications, Stanford, 1996.

The Complexity and Generative Capacity of Lexicalized Abstract Categorical Grammars

Ryo Yoshinaka^{1,2} and Makoto Kanazawa¹

¹ National Institute of Informatics,

2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

² Graduate School of Interdisciplinary Information Studies, University of Tokyo,

7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan

ry@nii.ac.jp

Abstract. Previous studies have shown that some well-known classes of grammars can be simulated by *Abstract Categorical Grammars* (de Groote 2001) in straightforward ways. These classes of grammars all generate subclasses of the PTIME languages. While the exact generative capacity of the class of ACGs and the complexity of its universal membership problem are both unknown, we show that the universal membership problem for the class of *lexicalized* ACGs is NP-complete and the languages generated by lexicalized ACGs form a subclass of NP which includes some NP-complete languages.

1 Introduction

De Groote [1] has introduced *Abstract Categorical Grammars* (ACGs), in which both basic building blocks of the grammar as well as grammatical combinations of them are represented by *linear* typed λ -terms.¹ The ACG has a number of attractive features as a grammar formalism for natural language. The expressive power of the typed λ -calculus allows straightforward encoding of diverse types of data, including strings, trees, and semantic representations, as well as operations on those data, within a single framework, while the linearity constraint captures resource sensitivity of natural language.

In this paper, we focus on the power of the ACG formalism as a device for generating string languages. Previous studies have shown that several well-established grammar formalisms can be simulated by ACGs in straightforward ways. Thus, the string languages generated by context-free grammars [1, 5], tree-adjoining grammars [2], linear context-free tree grammars [5], and m -linear context-free rewriting systems [4, 5] can all be generated by ACGs. These languages all belong to the class of PTIME languages.

The exact generative capacity of the class of ACGs and the complexity of its universal membership problem are both unknown, although it is easy to see that

¹ An almost identical formalism has been proposed independently by Muskens [10], who calls it *Lambda Grammar*.

the universal membership problem is EXPSPACE-hard. In this paper, we investigate the complexity and generative capacity of the class of *lexicalized* ACGs (LACGs), which, like categorial grammars and lexicalized tree-adjoining grammars, express all grammar-specific information as properties of lexical items. We show in Section 3 that the languages generated by LACGs form a subclass of NP which includes some NP-complete languages as well as languages that are not *semilinear*. This suggests that, while lexicalization severely restricts the power of ACGs in general, it is not overly restrictive for the purpose of modeling natural language grammars. In Section 4, we turn our attention to the class of *second-order* ACGs, a small subclass of the ACGs generating only semilinear languages. The lexicalized second-order ACGs happen to be equivalent to *de Saussure grammars* [8] with the restriction of linearity added. We show that every second-order ACG has an equivalent lexicalized second-order ACG (if the empty string is disregarded).

2 Abstract Categorial Grammars

2.1 Preliminaries

Definition 1. Let \mathcal{A} be a finite non-empty set of *atomic types*. The set $\mathcal{T}(\mathcal{A})$ of *types* built on \mathcal{A} is defined as follows:

- An atomic type $p \in \mathcal{A}$ is a type in $\mathcal{T}(\mathcal{A})$.
- If $\gamma, \delta \in \mathcal{T}(\mathcal{A})$, then $\gamma \multimap \delta \in \mathcal{T}(\mathcal{A})$.

The *order* Od of a type is defined as follows:

- The order $\text{Od}(p)$ of an atomic type $p \in \mathcal{A}$ is 1.
- The order of $\gamma \multimap \delta$ is defined by $\text{Od}(\gamma \multimap \delta) = \max\{\text{Od}(\gamma) + 1, \text{Od}(\delta)\}$.

A *higher-order signature* Σ is a triple $\langle \mathcal{A}, \mathcal{C}, \tau \rangle$ where \mathcal{A} is a finite non-empty set of atomic types, \mathcal{C} is a finite set of constants, and τ is a function from \mathcal{C} to $\mathcal{T}(\mathcal{A})$.

Let \mathcal{X} be a countably infinite set of *variables*. The set $\Lambda(\Sigma)$ of *linear λ -terms* built upon Σ , the set $\text{Fv}(M)$ of *free variables* in $M \in \Lambda(\Sigma)$, and the type $\hat{\tau}(M)$ of M are defined inductively as follows:

- A constant $c \in \mathcal{C}$ is a linear λ -term of type $\hat{\tau}(c) = \tau(c)$ and $\text{Fv}(c) = \emptyset$.
- For a variable $x \in \mathcal{X}$ and a type $\gamma \in \mathcal{T}(\mathcal{A})$, x^γ is a linear λ -term of type $\hat{\tau}(x^\gamma) = \gamma$ and $\text{Fv}(x^\gamma) = \{x^\gamma\}$.
- For two linear λ -terms M and N , if $\hat{\tau}(M) = \gamma \multimap \delta$, $\hat{\tau}(N) = \gamma$ and $\text{Fv}(M) \cap \text{Fv}(N) = \emptyset$, then MN is a linear λ -term of type $\hat{\tau}(MN) = \delta$ and $\text{Fv}(MN) = \text{Fv}(M) \cup \text{Fv}(N)$.
- For a variable $x \in \mathcal{X}$ and a linear λ -term M , if $x^\gamma \in \text{Fv}(M)$, then $\lambda x^\gamma.M$ is a linear λ -term of type $\hat{\tau}(\lambda x^\gamma.M) = \gamma \multimap \hat{\tau}(M)$ and $\text{Fv}(\lambda x^\gamma.M) = \text{Fv}(M) - \{x^\gamma\}$.

We will simply say λ -term instead of linear λ -term, since this paper does not treat any non-linear λ -terms. A λ -term M is *closed* iff $\text{FV}(M) = \emptyset$. A λ -term M is a *combinator* iff M is closed and M contains no constants. As usual, let \rightarrow_β , \rightarrow_β , $=_\beta$, $=_{\beta\eta}$ denote one-step β -reduction, multi-step β -reduction, β -equality, and $\beta\eta$ -equality, respectively.

For convenience, we simply write τ instead of $\hat{\tau}$, often omit the superscript on a variable (in particular on a bound variable) if its type is clear from the context, and adopt the following notations:

$$\begin{aligned} M_1 \dots M_{n-1} M_n &= (M_1 \dots M_{n-1}) M_n \\ \lambda x_1 x_2 \dots x_n. M &= \lambda x_1. (\lambda x_2 \dots x_n. M) \\ \gamma_1 \multimap \gamma_2 \multimap \dots \multimap \gamma_n &= \gamma_1 \multimap (\gamma_2 \multimap \dots \multimap \gamma_n) \\ \gamma^n \multimap \delta &= \begin{cases} \delta & \text{if } n = 0 \\ \gamma \multimap (\gamma^{n-1} \multimap \delta) & \text{if } n \geq 1 \end{cases} \end{aligned}$$

We use upper case italic letters M, N, \dots for λ -terms, lower case italic letters x, y, z, \dots for variables, sanserif $\mathbf{a}, \mathbf{A}, \dots$ for constants.

Definition 2 (de Groote [1]). An *Abstract Categorical Grammar (ACG)* \mathcal{G} is a quadruple $\langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$, where

- $\Sigma_0 = \langle \mathcal{A}_0, \mathcal{C}_0, \tau_0 \rangle$ is a higher-order signature, called the *abstract vocabulary*,
- $\Sigma_1 = \langle \mathcal{A}_1, \mathcal{C}_1, \tau_1 \rangle$ is a higher-order signature, called the *object vocabulary*,
- \mathcal{L} is a “compatible” homomorphism from $\Lambda(\Sigma_0)$ to $\Lambda(\Sigma_1)$, called the *lexicon*,
- $s \in \mathcal{A}_0$ is called the *distinguished type*.

More precisely, a lexicon \mathcal{L} is a pair of finite functions $\phi : \mathcal{A}_0 \rightarrow \mathcal{T}(\mathcal{A}_1)$ and $\psi : \mathcal{C}_0 \rightarrow \Lambda(\Sigma_1)$ such that $\psi(\mathbf{c})$ is closed and $\tau_1(\psi(\mathbf{c})) = \hat{\phi}(\tau_0(\mathbf{c}))$ for the unique homomorphic extension $\hat{\phi}$ of ϕ for all $\mathbf{c} \in \mathcal{C}_0$. In this case, it is easy to see that $\tau_1(\hat{\psi}(M)) = \hat{\phi}(\tau_0(M))$ for all $M \in \Lambda(\Sigma_0)$ for the unique homomorphic extension $\hat{\psi}$ of ψ such that $\hat{\psi}(x^\gamma) = x^{\hat{\phi}(\gamma)}$ for all $x \in \mathcal{X}$ and $\gamma \in \mathcal{T}(\mathcal{A}_0)$. We will simply write \mathcal{L} instead of $\hat{\phi}$ or $\hat{\psi}$, if no confusion occurs.

We sometimes use the modifier *abstract* or *object* to specify the vocabulary that a given type or term belongs to. Thus, we speak of *abstract atomic types*, *object atomic types*, *abstract constants*, *object constants*, etc.

An ACG $\mathcal{G} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$ generates two languages, the *abstract language* $\mathcal{A}(\mathcal{G})$ and the *object language* $\mathcal{O}(\mathcal{G})$, defined as follows:

$$\begin{aligned} \mathcal{A}(\mathcal{G}) &= \{M \in \Lambda(\Sigma_0) \mid M \text{ is a closed } \beta\text{-normal } \lambda\text{-term and } \tau_0(M) = s.\} \\ \mathcal{O}(\mathcal{G}) &= \{P \in \Lambda(\Sigma_1) \mid P \text{ is the } \beta\text{-normal form of } \mathcal{L}(M) \text{ for some } M \in \mathcal{A}(\mathcal{G}).\} \end{aligned}$$

The abstract language can be thought as a set of abstract grammatical structures, and the object language is regarded as the set of concrete forms obtained from these abstract structures and the lexicon. So, the term *Abstract Categorical Languages (ACLs)* means the object languages of ACGs.

Now, suppose that an alphabet \mathcal{C} is given. To represent strings on \mathcal{C} by λ -terms, we define a higher-order signature $\Sigma = \langle \mathcal{A}, \mathcal{C}, \tau \rangle$ as $\mathcal{A} = \{o\}$, $\tau(w) = str = o \multimap o$ for all $w \in \mathcal{C}$. So, we can represent a string $w_1 \dots w_n$ on \mathcal{C} by a λ -term $\lambda z^o.(w_1(w_2(\dots(w_n z) \dots)))$, and the empty string ε by $\lambda z^o.z$. Then, the concatenation of two strings is represented by the combinator $+ = \lambda y_1^{str} y_2^{str} z^o.y_1(y_2 z)$. Using the infix notation for $+$, we see that $L + (M + N) =_{\beta} (L + M) + N =_{\beta} \lambda z.L(M(Nz))$ for any λ -terms L , M and N of type str . For convenience, we simply write $M_1 + \dots + M_n$ for the β -normal form of the λ -term $(\dots(M_1 + M_2) + \dots + M_n)$, omitting parentheses. That is, a string $w_1 \dots w_n$ is represented by $w_1 + \dots + w_n$. We say that $M_1 + \dots + M_n$ is the *concatenation* of M_1, \dots, M_n . We say that $w_m + \dots + w_{m+k}$ is a *substring* of $w_1 + \dots + w_n$ if $1 \leq m \leq m+k \leq n$, though the former is strictly speaking not a *subterm* of the latter.

Lemma 1. *If $M(w_1 + \dots + w_n) \rightarrow_{\beta} x_1 + \dots + x_m$ for a λ -term M , then $w_1 + \dots + w_n$ is a substring of $x_1 + \dots + x_m$.*

Lemma 2. *Suppose that a type γ is constructed from the type str only. Then, there exist a combinator Z^{γ} of type γ and a λ -term Z_c^{γ} of type γ which contains exactly one occurrence of the constant c of type str .*

Proof. By induction on $\gamma = \gamma_1 \multimap \dots \multimap \gamma_m \multimap str$, we define combinators Z^{γ} of type γ and $Y^{\gamma \multimap str}$ of type $\gamma \multimap str$. Let $Z^{\gamma} = \lambda z_1^{\gamma_1} \dots z_m^{\gamma_m}.(Y^{\gamma_1 \multimap str} z_1^{\gamma_1}) + \dots + (Y^{\gamma_m \multimap str} z_m^{\gamma_m})$ and $Y^{\gamma \multimap str} = \lambda z^{\gamma}.z Z^{\gamma_1} \dots Z^{\gamma_m}$.

Let $Z_c^{\gamma} = \lambda z_1^{\gamma_1} \dots z_m^{\gamma_m}.c + (Z^{\gamma} z_1 \dots z_m)$ for $\gamma = \gamma_1 \multimap \dots \multimap \gamma_m \multimap str$. \square

2.2 Mathematical Properties of Abstract Categorical Grammars

There are many results on the mathematical properties of ACGs which are themselves, or are corollaries to, already known facts. These properties demonstrate the rich expressive power of ACGs.

Theorem 1 (de Groote and Pogodalla [4]). *For every language L defined by an m -linear context-free rewriting system, there is an ACG \mathcal{G} such that $\mathcal{O}(\mathcal{G})$ coincides with L .*

Definition 3. The *universal membership problem* is the problem of determining whether $T \in \mathcal{O}(\mathcal{G})$. The *emptiness problem* for ACGs is the problem of determining whether $\mathcal{O}(\mathcal{G}) = \emptyset$.

Proposition 1. *The universal membership problem for ACGs is at least as hard as the emptiness problem for ACGs.*

Proof. From an arbitrary ACG $\mathcal{G} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$ with $\Sigma_0 = \langle \mathcal{A}_0, \mathcal{C}_0, \tau_0 \rangle$, define $\mathcal{G}' = \langle \Sigma_0, \Sigma'_1, \mathcal{L}', s \rangle$ where $\Sigma'_1 = \langle \{o\}, \emptyset, \emptyset \rangle$ and $\mathcal{L}'(p) = str$ for every atomic type $p \in \mathcal{A}_0$ and $\mathcal{L}'(c) = Z^{\mathcal{L}'(\tau_0(c))}$ for every constant $c \in \mathcal{C}_0$, where Z^{γ} is as in Lemma 2. Then, $\mathcal{L}'(M) \rightarrow_{\beta} \lambda z^o.z$ for every $M \in \mathcal{A}(\mathcal{G}') = \mathcal{A}(\mathcal{G})$. Therefore, $\mathcal{O}(\mathcal{G}) \neq \emptyset$ iff $\lambda z^o.z \in \mathcal{O}(\mathcal{G}')$. \square

The following proposition is an easy corollary to a result obtained by de Groote et al. [3]

Proposition 2. *The emptiness problem for ACGs is decidable iff the multiplicative exponential linear logic (**MELL**) is decidable.*

Proof. Let $\mathcal{G} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$ be an ACG, where $\Sigma_0 = \langle \mathcal{A}_0, \mathcal{C}_0, \tau_0 \rangle$, $\mathcal{C}_0 = \{c_1, \dots, c_n\}$, and $\tau_0(c_i) = A_i$ for $1 \leq i \leq n$. Then $\mathcal{O}(\mathcal{G}) \neq \emptyset$ iff $\mathcal{A}(\mathcal{G}) \neq \emptyset$ iff $!A_1, \dots, !A_n \Rightarrow s$ is provable in **MELL**. This proves the “if” direction.

The “only if” direction can be proved as follows. De Groote et al. [3] show that the decidability of **MELL** is equivalent to the decidability of a fragment of it called **IMELL** $^\circ_0$. Formulas of **IMELL** $^\circ_0$ are of the form $!A$ or A , where A is a pure implicative formula, and the right-hand side of a sequent of **IMELL** $^\circ_0$ must be a pure implicative formula. Given a sequent $\mathcal{S} = !A_1, \dots, !A_n, B_1, \dots, B_m \Rightarrow C$ of **IMELL** $^\circ_0$, we can always form an ACG $\mathcal{G}^\mathcal{S} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$ such that $\Sigma_0 = \langle \mathcal{A}_\mathcal{S} \cup \{s\}, \mathcal{C}_0, \tau_0 \rangle$, $\mathcal{A}_\mathcal{S}$ is the set of atomic formulas in the sequent \mathcal{S} , $s \notin \mathcal{A}_\mathcal{S}$, $\mathcal{C}_0 = \{c_i \mid 1 \leq i \leq n\} \cup \{b\}$, $\tau_0(c_i) = A_i$ and $\tau_0(b) = (B_1 \multimap \dots \multimap B_m \multimap C) \multimap s$. (For instance, let $\Sigma_1 = \langle \{o\}, \emptyset, \emptyset \rangle$, $\mathcal{L}(p) = str$ for all $p \in \mathcal{A}_\mathcal{S} \cup \{s\}$ and let $\mathcal{L}(c) = \mathcal{L}^{\mathcal{L}(\tau_0(c))}$ for all $c \in \mathcal{C}_0$.) Then \mathcal{S} is provable in **IMELL** $^\circ_0$ iff \mathcal{S} is provable in **MELL** iff $\mathcal{A}(\mathcal{G}^\mathcal{S}) \neq \emptyset$ iff $\mathcal{O}(\mathcal{G}^\mathcal{S}) \neq \emptyset$. \square

The decidability of **MELL** is still open but it is known to be at least as hard as Petri-net reachability, which is at least EXPSPACE-hard [9]. Indeed, we can represent Petri-net reachability sets by ACGs directly using a reduction from *vector addition systems* (VASs), which are equivalent to Petri-nets.

Definition 4. An m -dimensional vector addition system (m -VAS) \mathcal{V} is a pair $\langle \Delta, \vec{s} \rangle$, where Δ is a finite subset of \mathbb{Z}^m and $\vec{s} \in \mathbb{N}^m$. We call \mathbb{N}^m the set of *configurations*. $\Rightarrow_{\vec{d}}$ for $\vec{d} \in \mathbb{Z}^m$ is a binary relation on \mathbb{N}^m such that for $\vec{a}, \vec{b} \in \mathbb{N}^m$, $\vec{a} \Rightarrow_{\vec{d}} \vec{b}$ iff $\vec{b} = \vec{a} + \vec{d}$. The union of $\Rightarrow_{\vec{d}}$ for $\vec{d} \in \Delta$ is denoted by \Rightarrow_Δ . A configuration \vec{b} is *reachable* iff $\vec{s} \Rightarrow_\Delta^* \vec{b}$. The *reachability set* $\mathcal{R}(\mathcal{V}) \subseteq \mathbb{N}^m$ of an m -VAS \mathcal{V} is the set of reachable configurations.

Definition 5. If w is a string over an alphabet $\mathcal{C} = \{c_1, \dots, c_m\}$ (or a λ -term in $\Lambda(\langle \mathcal{A}, \mathcal{C}, \tau \rangle)$ for some \mathcal{A} and τ), we write $\#_i(w)$ for the number of occurrences of c_i in w . We call $\#(w) = \langle \#_1(w), \dots, \#_m(w) \rangle$ the *Parikh vector* of w . If L is a language over \mathcal{C} , the *Parikh image* of L , denoted $\#(L)$, is the subset of \mathbb{N}^m defined by $\#(L) = \{\#(w) \mid w \in L\}$.

A set of vectors of natural numbers $\Gamma \subseteq \mathbb{N}^m$ is *linear* iff there are $\vec{a}_1, \dots, \vec{a}_n, \vec{b} \in \mathbb{N}^m$ such that $\Gamma = \{\vec{b} + \sum_{j=1}^n k_j \vec{a}_j \mid k_j \in \mathbb{N}\}$. $\Gamma \subseteq \mathbb{N}^m$ is *semilinear* iff there are linear sets $\Gamma_1, \dots, \Gamma_n \subseteq \mathbb{N}^m$ such that $\Gamma = \bigcup_{i=1}^n \Gamma_i$. Let L be a string language over an alphabet $\mathcal{C} = \{c_1, \dots, c_m\}$ (or $L \subseteq \Lambda(\langle \mathcal{A}, \mathcal{C}, \tau \rangle)$ for some \mathcal{A} and τ). L is *linear* (*semilinear*) iff the Parikh image of L is linear (semilinear).

Proposition 3. *For every reachability set $\mathcal{R}(\mathcal{V})$, there is an ACG $\mathcal{G}^\mathcal{V}$ such that $\mathcal{R}(\mathcal{V}) = \#(\mathcal{O}(\mathcal{G}^\mathcal{V}))$.*

Proof. Suppose that an m -VAS $\mathcal{V} = \langle \Delta, \vec{s} \rangle$ is given. Let us define a set \mathcal{A}_0 of atomic types and a function $\rho : \mathbb{Z}^m \rightarrow T(\mathcal{A}_0)$ as follows:

$$\begin{aligned} \mathcal{A}_0 &= \{p_i \mid 1 \leq i \leq m\} \cup \{q\} \\ \rho(\vec{d}) &= (p_1^{\nu(d_1)} \multimap \dots \multimap p_m^{\nu(d_m)} \multimap q) \multimap p_1^{\pi(d_1)} \multimap \dots \multimap p_m^{\pi(d_m)} \multimap q \\ \text{where } \nu(d_i) &= \begin{cases} 0 & \text{if } d_i \geq 0 \\ -d_i & \text{otherwise} \end{cases}, \quad \pi(d_i) = \begin{cases} 0 & \text{if } d_i \leq 0 \\ d_i & \text{otherwise} \end{cases} \end{aligned}$$

Let $\mathcal{G}^\mathcal{V} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, q \rangle$ with $\Sigma_0 = \langle \mathcal{A}_0, \mathcal{C}_0, \tau_0 \rangle$ and $\Sigma_1 = \langle \{o\}, \mathcal{C}_1, \tau_1 \rangle$ where $\mathcal{C}_0 = \{\mathbf{a}_i \mid 1 \leq i \leq m\} \cup \{\mathbf{b}_{\vec{d}} \mid \vec{d} \in \Delta\} \cup \{\mathbf{c}\}$, $\tau_0(\mathbf{a}_i) = p_i$, $\tau_0(\mathbf{b}_{\vec{d}}) = \rho(\vec{d})$, $\tau_0(\mathbf{c}) = p_1^{s_1} \multimap \dots \multimap p_m^{s_m} \multimap q$, $\mathcal{C}_1 = \{\mathbf{e}_i \mid 1 \leq i \leq m\}$, $\tau_1(\mathbf{e}_i) = str$, $\mathcal{L}(p_i) = \mathcal{L}(q) = str$, $\mathcal{L}(\mathbf{a}_i) = \mathbf{e}_i$, $\mathcal{L}(\mathbf{b}_{\vec{d}}) = Z^{\mathcal{L}(\rho(\vec{d}))}$, and $\mathcal{L}(\mathbf{c}) = Z^{\mathcal{L}(\tau_0(\mathbf{c}))}$. Then, we see that $\vec{b} \in \mathcal{R}(\mathcal{V})$ iff there is $P \in \mathcal{O}(\mathcal{G}^\mathcal{V})$ such that $\#(P) = \vec{b}$. The “if” part can be shown by induction on the total number of occurrences of $\mathbf{b}_{\vec{d}}$ for $\vec{d} \in \Delta$ in $M \in \mathcal{A}(\mathcal{G}^\mathcal{V})$ such that $\mathcal{L}(M) \rightarrow_\beta P \in \mathcal{O}(\mathcal{G}^\mathcal{V})$. The “only if” part can be shown by induction on n where n is such that $\vec{s} \Rightarrow_\Delta^n \vec{b}$. \square

Corollary 1. *There is an ACL which is not semilinear.*

Proof. It is known that there is a VAS whose reachability set is not semilinear [6]. This implies that there is an ACG which generates a non-semilinear language by Proposition 3. \square

3 The Class of Lexicalized Abstract Categorical Languages

In this section, we investigate the class of *lexicalized* ACGs (LACGs).

Definition 6. Let $\mathcal{G} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$ be an ACG. An abstract constant $\mathbf{c} \in \mathcal{C}_0$ is *lexical* iff $\mathcal{L}(\mathbf{c})$ contains an object constant. \mathcal{G} is *lexicalized* iff every abstract constant $\mathbf{c} \in \mathcal{C}_0$ is lexical.

Corollary 2. *There is a language defined by an LACG (LACL) which is not semilinear.*

Proof. Let \mathcal{V} be an m -VAS whose reachability set is non-semilinear. We obtain an LACG \mathcal{G}' from the ACG $\mathcal{G}^\mathcal{V}$ in the proof of Proposition 3 by adding a new constant \mathbf{f} of type str to \mathcal{C}_1 and redefining \mathcal{L} by $\mathcal{L}(\mathbf{a}_i) = \mathbf{e}_i$, $\mathcal{L}(\mathbf{b}_{\vec{d}}) = Z_{\mathbf{f}}^{\mathcal{L}(\rho(\vec{d}))}$, and $\mathcal{L}(\mathbf{c}) = Z_{\mathbf{f}}^{\mathcal{L}(\tau_0(\mathbf{c}))}$, where $Z_{\mathbf{f}}$ is as in Lemma 2. Since $\#(\mathcal{O}(\mathcal{G}^\mathcal{V}))$ can be obtained from $\#(\mathcal{O}(\mathcal{G}'))$ by a projection, and semilinearity is preserved under projections, the non-semilinearity of the former implies the non-semilinearity of the latter. \square

Corollary 3. *The emptiness problem for LACGs is decidable iff the multiplicative exponential linear logic (**MELL**) is decidable.*

Proof. We can lexicalize the ACG \mathcal{G}^S in the proof of Proposition 2 by Lemma 2. \square

Proposition 4. *The universal membership problem for LACGs is in NP.*

Proof. For an ACG $\mathcal{G} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$, if $\mathcal{L}(S) \rightarrow_\beta T \in \Lambda(\Sigma_1)$, the number of occurrences of constants in S does not exceed the number of occurrences of constants in T . Let T have m occurrences of constants. $T \in \mathcal{O}(\mathcal{G})$ iff there are abstract constants c_1, \dots, c_n for some $n \leq m$ and a combinator X of type $\tau_0(c_1) \multimap \dots \multimap \tau_0(c_n) \multimap s$ such that $\mathcal{L}(Xc_1 \dots c_n) \rightarrow_\beta T$. The size of a linear combinator X is bounded by a polynomial function of the size of its type and the number of β -reduction steps needed to eliminate redexes of a linear λ -term M is bounded by the size of M . This shows that the question “ $T \in \mathcal{O}(\mathcal{G})$?” is in NP. \square

Moreover, by the NP-hardness of the implicational fragment of intuitionistic linear logic (**IMLL**[−]) [7], the NP-hardness of the universal membership problem for LACGs holds.

Proposition 5. *The universal membership problem for LACGs is NP-complete.*

Proof. For a given sequent $\mathcal{S} = A_1, \dots, A_n \Rightarrow B$ of **IMLL**[−], we define an ACG $\mathcal{G}^S = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$ where $\Sigma_0 = \langle \mathcal{A}_S \cup \{s\}, \{a\}, \tau_0 \rangle$, \mathcal{A}_S is the set of atomic formulas in the sequent \mathcal{S} , $s \notin \mathcal{A}_S$, $\tau_0(a) = (A_1 \multimap \dots \multimap A_n \multimap B) \multimap s$, $\Sigma_1 = \langle \{o\}, \{c\}, \{c^\dagger \multimap str\} \rangle$, $\mathcal{L}(p) = str$ for all $p \in \mathcal{A}_S \cup \{s\}$, and $\mathcal{L}(a) = Z_c^{\mathcal{L}(\tau_0(a))}$. Then, \mathcal{S} is provable in **IMLL**[−] iff $\lambda z^o.cz \in \mathcal{O}(\mathcal{G}^S)$. \square

A question that naturally suggests itself at this point is whether or not there is an LACL which is NP-complete. We answer this question in the affirmative in the remainder of this section.

3.1 An NP-Complete Variation of the Satisfiability Problem

Definition 7. Let \mathcal{V} be a finite set of *Boolean variables*. Members of \mathcal{V} are called *positive literals* and members of $\neg\mathcal{V} = \{\neg v \mid v \in \mathcal{V}\}$ are called *negative literals*. $\mathcal{V} \cup \neg\mathcal{V}$ is the set of *literals*. A *conjunctive normal form formula (CNF)* \mathcal{F} on \mathcal{V} is a collection of *clauses*, which are non-empty subsets of $\mathcal{V} \cup \neg\mathcal{V}$. A valuation ψ on \mathcal{V} is a mapping from $\mathcal{V} \cup \neg\mathcal{V}$ to $\{0, 1\}$ such that $\psi(v) + \psi(\neg v) = 1$ for all $v \in \mathcal{V}$. A clause $\mathcal{C} \in \mathcal{F}$ is *satisfied by a valuation ψ via a literal $x \in \mathcal{V} \cup \neg\mathcal{V}$* iff $x \in \mathcal{C}$ and $\psi(x) = 1$. A CNF \mathcal{F} is *satisfied by a valuation ψ* iff every $\mathcal{C} \in \mathcal{F}$ is satisfied by ψ . A CNF \mathcal{F} is *satisfiable* iff there is ψ that satisfies \mathcal{F} . A *2PIN-CNF* is a special form of a CNF such that every Boolean variable $v \in \mathcal{V}$ positively occurs at most twice in \mathcal{F} and negatively occurs at most once in \mathcal{F} .

It is well known that the problem of determining whether or not a given CNF \mathcal{F} is satisfiable is NP-complete.

Theorem 2. *The question of whether or not a given 2P1N-CNF \mathcal{F} is satisfiable is NP-complete.*

Proof. For a given CNF \mathcal{F} on \mathcal{V} , we construct a 2P1N-CNF \mathcal{F}' on \mathcal{V}' such that \mathcal{F} is satisfiable iff \mathcal{F}' is satisfiable as follows.

- (i) For each $v_i \in \mathcal{V}$, let m_i be the number of occurrences of the positive literal v_i in \mathcal{F} and n_i be the number of occurrences of the negative literal $\neg v_i$ in \mathcal{F} .
- (ii) Introduce new Boolean variables $v_{i,j}$ for $1 \leq j \leq m_i$ and $u_{i,k}$ for $1 \leq k \leq n_i$.
- (iii) Replace the j -th occurrence of the positive literal v_i with $v_{i,j}$ for all j .
- (iv) Replace the k -th occurrence of the negative literal $\neg v_i$ with $u_{i,k}$ for all k .
- (v) Add clauses $\{v_{i,j}, \neg v_{i,j+1}\}$ for $1 \leq j < m_i$, $\{v_{i,m_i}, u_{i,1}\}$, $\{\neg u_{i,k}, u_{i,k+1}\}$ for $1 \leq k < n_i$, and $\{\neg u_{i,n_i}, \neg v_{i,1}\}$. (If $m_i = 0$, add the clause $\{\neg u_{i,n_i}, u_{i,1}\}$. If $n_i = 0$, add the clause $\{v_{i,m_i}, \neg v_{i,1}\}$.) \square

3.2 Universal Membership Problem

Although the complexity of the universal membership problem for LACGs has already been clarified, we present an alternative proof of its NP-completeness, since our technique for presenting an NP-complete LACL is an elaboration of this alternative proof.

Definition 8. Suppose that a 2P1N-CNF $\mathcal{F} = \{C_1, \dots, C_m\}$ on $\mathcal{V} = \{v_1, \dots, v_l\}$ is given. Let $\mathcal{G}^{m,l} = \langle \Sigma_0^{m,l}, \Sigma_1^{m,l}, \mathcal{L}^{m,l}, s \rangle$ where $\Sigma_0^{m,l} = \langle \{o, s\}, \mathcal{C}_0^{m,l}, \tau_0^{m,l} \rangle$ and $\Sigma_1^{m,l} = \langle \{o\}, \mathcal{C}_1^{m,l}, \tau_1^{m,l} \rangle$. $\mathcal{L}^{m,l}(o) = o$, $\mathcal{L}^{m,l}(s) = str$, and $\tau_1^{m,l}(x) = str$ for every constant $x \in \mathcal{C}_1^{m,l}$. The other sets and functions are defined as follows, where $1 \leq i \leq l$ and $1 \leq j \leq m$:

$x \in \mathcal{C}_0^{m,l}$	$\tau_0^{m,l}(x)$	$\mathcal{L}^{m,l}(x)$
A	$(o \multimap o) \multimap s$	$\lambda y^{str}.(a + y)$
$Q_{j,i}$	$((o \multimap o)^3 \multimap o \multimap o) \multimap o \multimap o$	$\lambda x^{str^3 \multimap str}.x c_j v_i p_{j,i}$
$M_{j,i}$	$((o \multimap o)^3 \multimap o \multimap o) \multimap o \multimap o$	$\lambda x^{str^3 \multimap str}.x c_j (v_i + v_i) n_{j,i}$
V_i	$o \multimap o$	v_i
$P_{j,i}$	$o \multimap o$	$p_{j,i}$
$N_{j,i}$	$o \multimap o$	$n_{j,i}$

Let

$$T_{\mathcal{F}} = a + c_1 + c_2 + \dots + c_m + v_1 + v_1 + v_2 + v_2 + \dots + v_l + v_l + \vec{p}_{\mathcal{F}} + \vec{n}_{\mathcal{F}}$$

where $\vec{p}_{\mathcal{F}}$ is the concatenation of $p_{j,i}$ such that $v_i \in \mathcal{C}_j$ and $\vec{n}_{\mathcal{F}}$ is the concatenation of $n_{j,i}$ such that $\neg v_i \in \mathcal{C}_j$.

Note that $\mathcal{G}^{m,l}$ depends only on the number of Boolean variables and the number of clauses in \mathcal{F} . The essence of \mathcal{F} is encoded in $\vec{p}_{\mathcal{F}}$ and $\vec{n}_{\mathcal{F}}$ in $T_{\mathcal{F}}$. Therefore, we state that for every 2P1N-CNF \mathcal{F} such that \mathcal{F} contains at most m clauses and at most l Boolean variables, $T_{\mathcal{F}} \in \mathcal{O}(\mathcal{G}^{m,l})$ iff \mathcal{F} is satisfiable.

It remains to show $\mathcal{L}^{m,l}(S_\phi) \rightarrow_\beta T_{\mathcal{F}}$. Fix j and suppose that $\phi(\mathcal{C}_j) = v_i$. Since $\mathcal{L}^{m,l}(S_j) = \mathcal{L}^{m,l}(\mathbf{Q}_{j,i}) = \lambda x.xc_j v_i p_{j,i}$, in the β -reduction from $\mathcal{L}^{m,l}(S_\phi)$ to its β -normal form, c_j is substituted for w_j in S'_ϕ , v_i for u_j , and $p_{j,i}$ for y_j . Suppose that $\phi(\mathcal{C}_j) = \neg v_i$. Since $\mathcal{L}^{m,l}(S_j) = \mathcal{L}^{m,l}(\mathbf{M}_{j,i}) = \lambda x.xc_j (v_i + v_i) n_{j,i}$, c_j is substituted for w_j in S'_ϕ , $v_i + v_i$ for u_j , and $n_{j,i}$ for y_j in the β -reduction from $\mathcal{L}^{m,l}(S_\phi)$ to its β -normal form. Thus, by these substitutions, w_j in S'_ϕ is replaced with c_j for all j , V_i with $v_i + v_i$ for all i , $P_{j,i}$ with $p_{j,i}$ for all $P_{j,i}$ in \vec{P} , and $N_{j,i}$ with $n_{j,i}$ for all $N_{j,i}$ in \vec{N} . Therefore, $\mathcal{L}^{m,l}(S_\phi) \rightarrow_\beta T_{\mathcal{F}} \in \mathcal{O}(\mathcal{G}^{m,l})$. \square

Lemma 4. \mathcal{F} is satisfiable whenever $T_{\mathcal{F}} \in \mathcal{O}(\mathcal{G}^{m,l})$.

Proof. Suppose that there is $S \in \mathcal{A}(\mathcal{G}^{m,l})$ such that $\mathcal{L}^{m,l}(S) \rightarrow_\beta T_{\mathcal{F}}$. We define a valuation ψ as follows:

$$\psi(v_i) = \begin{cases} 1 & \text{if } \mathbf{Q}_{j,i} \text{ appears in } S \text{ for some } j \\ 0 & \text{if } \mathbf{M}_{j,i} \text{ appears in } S \text{ for some } j \\ \text{any value} & \text{otherwise} \end{cases}$$

It is easy to verify that ψ is well defined, because if there are i, j and k such that both $\mathbf{Q}_{j,i}$ and $\mathbf{M}_{k,i}$ are in S , then v_i must occur at least three times in $\mathcal{L}^{m,l}(S)$. We confirm that every \mathcal{C}_j is satisfied by ψ . It is clear that for every c_j in $T_{\mathcal{F}}$, there is exactly one i such that either $\mathbf{Q}_{j,i}$ or $\mathbf{M}_{j,i}$ appears in S . If $\mathbf{Q}_{j,i}$ is in S , since $\mathcal{L}^{m,l}(\mathbf{Q}_{j,i})$ contains $p_{j,i}$, then $\mathcal{L}^{m,l}(S)$ and $T_{\mathcal{F}}$ also contain $p_{j,i}$. That is, $v_i \in \mathcal{C}_j$ by the definition of $T_{\mathcal{F}}$. So, \mathcal{C}_j is satisfied by ψ via v_i . When $\mathbf{M}_{j,i}$ is in S , similarly ψ satisfies \mathcal{C}_j via $\neg v_i$. \square

Proposition 6. \mathcal{F} is satisfiable iff $T_{\mathcal{F}} \in \mathcal{O}(\mathcal{G}^{m,l})$.

3.3 An NP-Complete Lexicalized Abstract Categorical Language

In this subsection, we present an LACG which generates an NP-complete language. We construct an LACG $\mathcal{G}^* = \langle \Sigma_0^*, \Sigma_1^*, \mathcal{L}^*, s \rangle$ such that for every 2P1N-CNF \mathcal{F} , one can find $T_{\mathcal{F}}^*$ such that $T_{\mathcal{F}}^* \in \mathcal{O}(\mathcal{G}^*)$ iff \mathcal{F} is satisfiable. Let $\mathcal{C}_1^* = \{a, \llbracket, \rrbracket, c, d, v, p, n\}$. We use the following λ -terms in $\Lambda(\Sigma_1^*)$, which play the role played by $c_j, v_i, p_{j,i}, n_{j,i} \in \mathcal{C}_1^{m,l}$ in the preceding construction:

$$\begin{aligned} C_j &= \llbracket + \overbrace{c + \cdots + c}^{j\text{-times}} + \rrbracket \\ V_i &= \llbracket + \overbrace{v + \cdots + v}^{i\text{-times}} + \rrbracket \\ P_{j,i} &= \llbracket + \overbrace{d + \cdots + d}^{j\text{-times}} + \overbrace{p + \cdots + p}^{i\text{-times}} + \rrbracket \\ N_{j,i} &= \llbracket + \overbrace{d + \cdots + d}^{j\text{-times}} + \overbrace{n + \cdots + n}^{i\text{-times}} + \rrbracket \end{aligned}$$

Let

$$T_{\mathcal{F}}^* = \mathbf{a} + C_1 + \cdots + C_m + V_1 + V_1 + \cdots + V_l + V_l + \vec{P}_{\mathcal{F}} + \vec{N}_{\mathcal{F}}$$

where $P_{j,i}$ appears in $\vec{P}_{\mathcal{F}}$ iff $v_i \in \mathcal{C}_j$, and $N_{j,i}$ appears in $\vec{N}_{\mathcal{F}}$ iff $\neg v_i \in \mathcal{C}_j$. By the brackets \llbracket and \rrbracket , no other interpretation of $T_{\mathcal{F}}^*$ as a concatenation of \mathbf{a} , C_j , V_i , $P_{j,i}$ and $N_{j,i}$ is allowed, e.g., $V_3 + V_3$ cannot be interpreted as $V_2 + V_2 + V_2$ etc. Thus, the remaining task is to make sure that, whenever C_j appears as a substring of some $T \in \mathcal{O}(\mathcal{G}^*)$, either V_i and $P_{j,i}$ or $V_i + V_i$ and $N_{j,i}$ appear as substrings of T for some i . (Recall that $\mathcal{L}^{m,l}(\mathbf{Q}_{j,i}) = \lambda x.x\mathbf{c}_j\mathbf{v}_i\mathbf{p}_{j,i}$ and $\mathcal{L}^{m,l}(\mathbf{M}_{j,i}) = \lambda x.x\mathbf{c}_j(\mathbf{v}_i + \mathbf{v}_i)\mathbf{n}_{j,i}$.)

Definition 9. Let $\mathcal{G}^* = \langle \Sigma_0^*, \Sigma_1^*, \mathcal{L}^*, s \rangle$ with $\Sigma_0^* = \langle \mathcal{A}_0^*, \mathcal{C}_0^*, \tau_0^* \rangle$ and $\Sigma_1^* = \langle \mathcal{A}_1^*, \mathcal{C}_1^*, \tau_1^* \rangle$, where $\mathcal{A}_0^* = \{o, c, p, n, s\}$, $\mathcal{A}_1^* = \{o\}$, $\tau_1^*(x) = str$ for every constant $x \in \mathcal{C}_1^*$, $\mathcal{L}^*(o) = o$, $\mathcal{L}^*(c) = str^2 \multimap str$, $\mathcal{L}^*(p) = str^3 \multimap str$, $\mathcal{L}^*(n) = str^4 \multimap str$, $\mathcal{L}^*(s) = str$, and the other set and functions are defined as follows:

$x \in \mathcal{C}_0^*$	$\tau_0^*(x)$	$\mathcal{L}^*(x)$
A	$(o \multimap o) \multimap s$	$\lambda y_0.(\mathbf{a} + y_0)$
L	$c \multimap o \multimap o$	$\lambda x_2.x_2\llbracket\llbracket$
C	$c \multimap c$	$\lambda x_2y_1y_2.x_2(y_1 + \mathbf{c})(y_2 + \mathbf{d})$
O_p	$p \multimap c$	$\lambda x_3y_1y_2.x_3(y_1 + \llbracket)y_2$
U_p	$p \multimap p$	$\lambda x_3y_1y_2y_3.x_3y_1(y_2 + \mathbf{v})(y_3 + \mathbf{p})$
R_p	$((o \multimap o)^3 \multimap o \multimap o) \multimap p$	$\lambda x_3y_1y_2y_3.x_3y_1(y_2 + \llbracket)(y_3 + \llbracket)$
O_n	$n \multimap c$	$\lambda x_4y_1y_2.x_4(y_1 + \llbracket)\llbracket y_2$
U_n	$n \multimap n$	$\lambda x_4y_1y_2y_3y_4.x_4y_1(y_2 + \mathbf{v})(y_3 + \mathbf{v})(y_4 + \mathbf{n})$
R_n	$((o \multimap o)^3 \multimap o \multimap o) \multimap n$	$\lambda x_3y_1y_2y_3y_4.x_3y_1(y_2 + \llbracket + y_3 + \llbracket)(y_4 + \llbracket)$
$\llbracket, \rrbracket, \mathbf{D}, \mathbf{V}, \mathbf{P}, \mathbf{N}$	$o \multimap o$	$\llbracket, \rrbracket, \mathbf{d}, \mathbf{v}, \mathbf{p}, \mathbf{n}$, respectively

where the type of each x_i is $str^i \multimap str$ and the type of each y_i is str .

Lemma 5. $T_{\mathcal{F}}^* \in \mathcal{O}(\mathcal{G}^*)$ whenever \mathcal{F} is satisfiable.

Proof. Let us define the following λ -terms in $\Lambda(\Sigma_0^*)$ as follows, which play the role played by $V_i, P_{j,i}, N_{j,i}, Q_{j,i}, M_{j,i} \in \mathcal{C}_0^{m,l}$ in the preceding construction:

$$\begin{aligned}
 \bar{V}_i &= \lambda z^o. \overbrace{\llbracket (\mathbf{V}(\dots (\mathbf{V}(\llbracket z))) \dots))}^{i\text{-times}} \\
 \bar{P}_{j,i} &= \lambda z^o. \overbrace{\llbracket (\mathbf{D}(\dots (\mathbf{D}(\overbrace{\mathbf{P}(\dots (\mathbf{P}(\llbracket z)))}^{i\text{-times}})) \dots))}^{j\text{-times}} \\
 \bar{N}_{j,i} &= \lambda z^o. \overbrace{\llbracket (\mathbf{D}(\dots (\mathbf{D}(\overbrace{\mathbf{N}(\dots (\mathbf{N}(\llbracket z)))}^{i\text{-times}})) \dots))}^{j\text{-times}} \\
 \bar{Q}_{j,i} &= \lambda x^{(o \multimap o)^3 \multimap o \multimap o}. \overbrace{\mathbf{L}(\overbrace{\mathbf{C}(\dots (\mathbf{C}(\overbrace{\mathbf{O}_p(\overbrace{\mathbf{U}_p(\dots (\mathbf{U}_p(\mathbf{R}_p x)))}^{i\text{-times}})) \dots))}^{j\text{-times}})) \dots)} \\
 \bar{M}_{j,i} &= \lambda x^{(o \multimap o)^3 \multimap o \multimap o}. \mathbf{L}(\overbrace{\mathbf{C}(\dots (\mathbf{C}(\overbrace{\mathbf{O}_n(\overbrace{\mathbf{U}_n(\dots (\mathbf{U}_n(\mathbf{R}_n x)))}^{i\text{-times}})) \dots))}^{j\text{-times}})) \dots)
 \end{aligned}$$

where $\tau_0^*(\bar{V}_i) = \tau_0^*(\bar{P}_{j,i}) = \tau_0^*(\bar{N}_{j,i}) = o \multimap o$ and $\tau_0^*(\bar{Q}_{j,i}) = \tau_0^*(\bar{M}_{j,i}) = ((o \multimap o)^3 \multimap o \multimap o) \multimap o \multimap o$. Then, $\mathcal{L}^*(\bar{V}_i) = V_i$, $\mathcal{L}^*(\bar{P}_{j,i}) = P_{j,i}$, and $\mathcal{L}^*(\bar{N}_{j,i}) = N_{j,i}$. It is easy to see that $\mathcal{L}^*(\bar{Q}_{j,i}) \rightarrow_\beta \lambda x.xC_j V_i P_{j,i}$ and $\mathcal{L}^*(\bar{M}_{j,i}) \rightarrow_\beta \lambda x.xC_j (V_i + V_i) N_{j,i}$.

Suppose that \mathcal{F} is satisfiable. Let S^* be the β -normal form of S_ϕ^* obtained by replacing V_i in S_ϕ in the proof of Lemma 3 with \bar{V}_i , $P_{j,i}$ with $\bar{P}_{j,i}$, $N_{j,i}$ with $\bar{N}_{j,i}$, $Q_{j,i}$ with $\bar{Q}_{j,i}$, and $M_{j,i}$ with $\bar{M}_{j,i}$. Then, $S^* \in \mathcal{A}(\mathcal{G}^*)$ and $\mathcal{L}^*(S^*) \rightarrow_\beta T_{\mathcal{F}}^*$. \square

Lemma 6. \mathcal{F} is satisfiable whenever $T_{\mathcal{F}}^* \in \mathcal{O}(\mathcal{G}^*)$.

Proof. If $S[\vec{y}] \in \Lambda(\Sigma_0^*)$ of type $o \multimap o$ has free variables \vec{y} of type $o \multimap o$, then S is in one of the following forms:

$$S[\vec{y}] =_{\beta\eta} \lambda z.S'[\vec{y}_1](S''[\vec{y}_2]z) \text{ where } \vec{y} = \vec{y}_1\vec{y}_2 \quad (1)$$

$$\text{or } \lambda z.z \text{ only if } \vec{y} = \emptyset \quad (2)$$

$$\text{or } y \text{ only if } \vec{y} = y \quad (3)$$

$$\text{or } w \in \{\llbracket, \rrbracket, D, V, P, N\} \text{ only if } \vec{y} = \emptyset \quad (4)$$

$$\text{or } L(C(\dots(C(O_p(U_p(\dots(U_p(R_p(\lambda y_1 y_2 y_3.S'[\vec{y}y_1 y_2 y_3]))\dots))))\dots))\dots) \quad (5)$$

$$\text{or } L(C(\dots(C(O_n(U_n(\dots(U_n(R_n(\lambda y_1 y_2 y_3.S'[\vec{y}y_1 y_2 y_3]))\dots))))\dots))\dots) \quad (6)$$

where S' and S'' have the type $o \multimap o$ and are also in one of these forms. If a λ -term S is in the form of (5), then we see $\bar{Q}_{j,i}(\lambda y_1 y_2 y_3.S'[\vec{y}y_1 y_2 y_3]) \rightarrow_\beta S[\vec{y}]$, and if S is in the form of (6), then $\bar{M}_{j,i}(\lambda y_1 y_2 y_3.S'[\vec{y}y_1 y_2 y_3]) \rightarrow_\beta S[\vec{y}]$, where $\bar{Q}_{j,i}$ and $\bar{M}_{j,i}$ are as in Lemma 5. So, we can rewrite (5) and (6) as follows:

$$\bar{Q}_{j,i}(\lambda y_1 y_2 y_3.S'[\vec{y}y_1 y_2 y_3]) \quad (5')$$

$$\bar{M}_{j,i}(\lambda y_1 y_2 y_3.S'[\vec{y}y_1 y_2 y_3]) \quad (6')$$

Clearly every λ -term in $\mathcal{A}(\mathcal{G}^*)$ is of the form AS for some S of type $o \multimap o$. Now, suppose that $\mathcal{L}^*(AS) \rightarrow_\beta T_{\mathcal{F}}^*$. We can assume that S is recursively constructed by (1)–(4), (5') and (6'). Thus, the only ways to construct C_j as a substring of the β -normal form of $\mathcal{L}^*(AS)$ are to use $\bar{Q}_{j,i}$ or $\bar{M}_{j,i}$ in S for some i as in (5') or (6') by Lemma 1. Therefore, the following valuation ψ is well-defined and satisfies \mathcal{F} as in Lemma 4.

$$\psi(v_i) = \begin{cases} 1 & \text{if } \bar{Q}_{j,i} \text{ is used for some } j \\ 0 & \text{if } \bar{M}_{j,i} \text{ is used for some } j \\ \text{any value} & \text{otherwise} \end{cases} \quad \square$$

Proposition 7. *The problem of determining whether $T \in \mathcal{O}(\mathcal{G}^*)$ is NP-complete.*²

² S. Salvati (personal communication) has independently obtained an alternative proof of this proposition using a reduction from 3-PARTITION.

4 Second-Order Abstract Categorical Grammars

Although we lack a good characterization of the class of general ACLs, we expect it to be much broader than the class of LACLs. In this section, we focus on a small subclass of the class of ACGs, in which every grammar has an equivalent lexicalized grammar if combinators in the object language are disregarded. We say that an ACG \mathcal{G} is *second-order* iff the order of the type of each abstract constant of \mathcal{G} is at most two. ACGs used by de Groote and Pogodalla to simulate m -linear context-free rewriting systems [4,5] and tree-adjoining grammars [2] are second-order ACGs (2ACGs) in this sense. Kracht [8] defines *de Saussure grammars*, which generate sets of pairs of λ -terms whose first component represents strings. It is easy to see that the lexicalized 2ACGs (L2ACGs) are equivalent to de Saussure grammars with the restriction of linearity added, if we view the latter as generating string languages.³

If \mathcal{G} is a 2ACG, every $M \in \mathcal{A}(\mathcal{G})$ is a pure application term. This means that $\mathcal{A}(\mathcal{G})$ can be identified with the set of derivation trees of a context-free grammar.

Proposition 8. *Every 2ACL is semilinear.*

Proof. The semilinearity of context-free languages implies that $\#(\mathcal{A}(\mathcal{G}))$ is semilinear for every 2ACG \mathcal{G} . Since the lexicon \mathcal{L} is a linear translation which maps $\#(M)$ to $\#(\mathcal{L}(M))$ for $M \in \Lambda(\Sigma_0)$ and semilinearity is preserved under linear translations, $\#(\mathcal{O}(\mathcal{G}))$ is also semilinear. \square

The above result is optimal in the sense that the types of the abstract constants of an ACG \mathcal{G}^\vee in the proof of Proposition 3, which may generate a non-semilinear language, are at most third-order.

We construct an L2ACG equivalent to a given 2ACG in the remainder of this section.

Lemma 7. *For a λ -term M , let $C(M)$ be the multiset of constants such that for each $c \in \mathcal{C}$, c is an element of $C(M)$ with multiplicity n iff c occurs in M n times. Fix a type γ and a multiset C of constants. There are finitely many closed λ -terms M of type γ modulo $=_\beta$ such that $C(M)$ is a sub-multiset of C .*

Definition 10. Let a 2ACG $\mathcal{G} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$ and an abstract atomic type $p \in \mathcal{A}_0$ be given. An n -loop on p is a sequence $\langle M_1, \dots, M_n \rangle$ of λ -terms $M_i \in \Lambda(\Sigma_0)$ of type p each of which contains a lexical constant and a free variable y_i of type p . A λ -term $M \in \Lambda(\Sigma_0)$ contains an n -loop $\langle M_1, \dots, M_n \rangle$ iff there are M_0 and M_{n+1} such that $M = M_0[M_1[\dots[M_n[M_{n+1}/y_n]/y_{n-1}]\dots]/y_0]$ with $y_0 \in \text{FV}(M_0)$.

³ S. Salvati (personal communication) has shown that every 2ACG generates a PTIME language.

Definition 11. Fix $\mathcal{G} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$ with $\Sigma_0 = \langle \mathcal{A}_0, \mathcal{C}_0, \tau_0 \rangle$ and $\mathcal{G}' = \langle \Sigma'_0, \Sigma_1, \mathcal{L}', s \rangle$ with $\Sigma'_0 = \langle \mathcal{A}_0, \mathcal{C}'_0, \tau'_0 \rangle$. For $M \in \Lambda(\Sigma_0)$ and $M' \in \Lambda(\Sigma'_0)$, $M \sim M'$ iff $\tau_0(M) = \tau'_0(M')$ and $\mathcal{L}(M) =_\beta \mathcal{L}'(M')$.

Definition 12. For a 2ACG $\mathcal{G} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$, define the set $\Theta_{\mathcal{G}}$ of *basic* λ -terms of \mathcal{G} as follows:

$$\begin{aligned} \Theta_{\mathcal{G}}^0 &= \{ M \in \Lambda(\Sigma_0) \mid M \text{ is closed, of an atomic type, and contains} \\ &\quad \text{at least one lexical constant but no 3-loop.} \} \\ \Theta_{\mathcal{G}} &= \{ \lambda \vec{x}. M_0 \in \Lambda(\Sigma_0) \mid M_0[\vec{M}/\vec{x}] \in \Theta_{\mathcal{G}}^0, \\ &\quad \text{each } \lambda\text{-term in } \vec{M}, M_0 \text{ contains a lexical constant, and} \\ &\quad \vec{M} \text{ and } \vec{x} \text{ have atomic types } (\vec{x} \text{ and } \vec{M} \text{ can be empty}). \} \end{aligned}$$

Note that every $M \in \Theta_{\mathcal{G}}^0$ contains no variables or λ -abstractions.

Lemma 8. For every 2ACG \mathcal{G} , $\Theta_{\mathcal{G}}$ is finite up to \sim .

Proof. By Lemma 7 and the definition of $\Theta_{\mathcal{G}}$, it is enough to show an upper bound $K_{\mathcal{G}}$ on the number of occurrences of lexical constants in $M \in \Theta_{\mathcal{G}}^0$. This is because the size of the type of $\lambda \vec{x}. M_0 \in \Theta_{\mathcal{G}}$ and the number of occurrences of lexical constants in $\lambda \vec{x}. M_0$ are both bounded by $K_{\mathcal{G}}$, and hence the type of $\mathcal{L}(\lambda \vec{x}. M_0)$ and the number of constants in $\mathcal{L}(\lambda \vec{x}. M_0)$ are both bounded, since \mathcal{L} is a homomorphism.

Let $n = \max\{n \mid \tau_0(c) = p_1 \multimap \dots \multimap p_n \multimap q \text{ for } p_i, q \in \mathcal{A}_0, c \in \mathcal{C}_0\}$ and $|M|$ denote the number of occurrences of lexical constants in $M \in \Lambda(\Sigma_0)$. Since $M \in \Theta_{\mathcal{G}}^0$ can be identified with a derivation tree of a context-free grammar, we can give an upper bound $K_{\mathcal{G}}$ in a way similar to the proof of Ogden's Lemma for context-free grammars.

We define a sequence $\langle M = M_0, \dots, M_m \rangle$ of λ -terms of atomic types such that M_{i+1} is a subterm of M_i and $|M_i| > |M_{i+1}|$. Let $M_0 = M$. Suppose that M_i is defined. If $|M_i| = 1$, then let $m = i$ and halt. Otherwise, take the smallest subterm M'_i of M_i such that $|M_i| = |M'_i|$ and $\tau_0(M'_i) \in \mathcal{A}_0$. For $M'_i = cN_1 \dots N_{n'}$ for $n' \leq n$, let j be such that $|N_j| \geq |N_{j'}|$ for any $1 \leq j' \leq n'$. Define $M_{i+1} = N_j$.

Because M has no 3-loop, we have $m \leq 3|\mathcal{A}_0|$. By the definition, $|M_m| = 1$ and $|M_i| \leq n|M_{i+1}| + 1$ for all $0 \leq i < m$. Therefore, $|M_0| \leq (n+1)^{3|\mathcal{A}_0|}$. \square

Proposition 9. For every 2ACG \mathcal{G} , there is an L2ACG \mathcal{G}' such that $\mathcal{O}(\mathcal{G}) = \mathcal{O}(\mathcal{G}') \cup \Omega$ for $\Omega = \{P \in \mathcal{O}(\mathcal{G}) \mid P \text{ is a combinator}\}$.

Proof. For a given 2ACG $\mathcal{G} = \langle \Sigma_0, \Sigma_1, \mathcal{L}, s \rangle$ with $\Sigma_0 = \langle \mathcal{A}_0, \mathcal{C}_0, \tau_0 \rangle$, we define an L2ACG $\mathcal{G}' = \langle \Sigma'_0, \Sigma_1, \mathcal{L}', s \rangle$ with $\Sigma'_0 = \langle \mathcal{A}_0, \mathcal{C}'_0, \tau'_0 \rangle$ where

$$\begin{aligned} \mathcal{C}'_0 &= \{c_\Gamma \mid \Gamma \in \Theta_{\mathcal{G}}/\sim\}, \\ \tau'_0 &= \{c_\Gamma \mapsto \neg\tau_0(M) \mid M \in \Gamma \in \Theta_{\mathcal{G}}/\sim\}, \\ \mathcal{L}'(p) &= \mathcal{L}(p) \text{ for all } p \in \mathcal{A}_0, \text{ and} \\ \mathcal{L}'(c_\Gamma) &\text{ is the } \beta\text{-normal form of } \mathcal{L}(M) \text{ for } M \in \Gamma. \end{aligned}$$

By Lemma 8, \mathcal{C}'_0 is finite.

Clearly $\mathcal{O}(\mathcal{G}') \cup \Omega \subseteq \mathcal{O}(\mathcal{G})$. To prove that $\mathcal{O}(\mathcal{G}) \subseteq \mathcal{O}(\mathcal{G}') \cup \Omega$, we show the following claim by induction on the size of M :

- Let $M \in \Lambda(\Sigma_0)$ have an atomic type and contain a lexical constant but no variables. If $M = M_0[\vec{M}/\vec{x}]$ where each λ -term in \vec{M}, M_0 contains a lexical constant and each λ -term in \vec{M} has an atomic type, then there is $N_0 \in \Lambda(\Sigma'_0)$ such that $N_0 \sim M_0$.

Considering the case $M = M_0 \in \mathcal{A}(\mathcal{G})$, the proposition follows from the claim.

(Base) If $M = M_0[\vec{M}/\vec{x}] \in \Theta_{\mathcal{G}}^0$, $\lambda\vec{x}.M_0 \in \Theta_{\mathcal{G}}$ for a sequence \vec{x} made up of the variables in $\text{Fv}(M_0)$. Then, $c_{\Gamma}\vec{x} \sim M_0$ for $c_{\Gamma} \in \mathcal{C}'_0$ such that $\lambda\vec{x}.M_0 \in \Gamma \in \Theta_{\mathcal{G}}/\sim$.

(Step) Suppose that $M = M_0[\vec{M}/\vec{x}]$ contains a 3-loop.

(Case 1) M_0 contains a 3-loop $\langle L_1, L_2, L_3 \rangle$. Then, there are L_0 and L_4 such that $M_0 = L_0[L_1[L_2[L_3[L_4/y_3]/y_2]/y_1]/y_0$. Let us divide \vec{x} into three parts \vec{x}_1 , \vec{x}_2 , and \vec{x}_3 , where \vec{x}_1 , \vec{x}_2 , and \vec{x}_3 are sequences made up of the variables in $\text{Fv}(L_0) \cup \text{Fv}(L_1) - \{y_0, y_1\}$, $\text{Fv}(L_2) - \{y_2\}$, and $\text{Fv}(L_3) \cup \text{Fv}(L_4) - \{y_3\}$, respectively. Let \vec{M}_j for $1 \leq j \leq 3$ be the λ -terms such that $M_0[\vec{M}_j/\vec{x}_j]_{1 \leq j \leq 3} = M$. By applying the induction hypothesis to $L_0[L_1/y_0][L_3[L_4/y_3][\vec{M}_3/\vec{x}_3]/y_1, \vec{M}_1/\vec{x}_1]$, we obtain $N'_0 \in \Lambda(\Sigma'_0)$ with $N'_0 \sim L_0[L_1/y_0]$. By applying the induction hypothesis to $L_2[L_3[L_4/y_3]/y_2][\vec{M}_2/\vec{x}_2, \vec{M}_3/\vec{x}_3]$, we obtain $N''_0 \in \Lambda(\Sigma'_0)$ with $N''_0 \sim L_2[L_3[L_4/y_3]/y_2]$. Let $N_0 = N'_0[N''_0/y_1]$. Then, $N_0 \sim M_0$.

(Case 2) There are a λ -term M_k in \vec{M} and λ -terms $L_0, L_1, L_2, L'_3, L''_3, L_4$ such that

$$\begin{cases} M_0 = L_0[L_1[L_2[L'_3/y_2]/y_1]/y_0] \\ M_k = L''_3[L_4/y_3] \\ M_0[M_k/x_k] = L_0[L_1[L_2[L'_3[L''_3[L_4/y_3]/x_k]/y_2]/y_1]/y_0] \end{cases}$$

and $\langle L_1, L_2, L'_3[L''_3/x_k] \rangle$ is a 3-loop. Let us divide \vec{x} into three parts \vec{x}_1 , \vec{x}_2 , and \vec{x}_3 , where \vec{x}_1 , \vec{x}_2 , and \vec{x}_3 are sequences made up of the variables in $\text{Fv}(L_0) \cup \text{Fv}(L_1) - \{y_0, y_1\}$, $\text{Fv}(L_2) - \{y_2\}$, and $\text{Fv}(L'_3)$, respectively. Let \vec{M}_j for $1 \leq j \leq 3$ be the λ -terms such that $M_0[\vec{M}_j/\vec{x}_j]_{1 \leq j \leq 3} = M$. Note that x_k is in \vec{x}_3 and thus $L'_3[\vec{M}_3/\vec{x}_3]$ contains a lexical constant. By applying the induction hypothesis to $L_0[L_1/y_0][L'_3[\vec{M}_3/\vec{x}_3]/y_1, \vec{M}_1/\vec{x}_1]$, we obtain $N'_0 \sim L_0[L_1/y_0]$. By applying the induction hypothesis to $L_2[L'_3/y_2][\vec{M}_2/\vec{x}_2, \vec{M}_3/\vec{x}_3]$, we obtain $N''_0 \sim L_2[L'_3/y_2]$. Let $N_0 = N'_0[N''_0/y_1]$. Then $N_0 \sim M_0$.

(Case 3) There are M_k in \vec{M} and $L_0, L_1, L'_2, L''_2, L_3, L_4$ such that

$$\begin{cases} M_0 = L_0[L_1[L'_2/y_1]/y_0] \\ M_k = L''_2[L_3[L_4/y_3]/y_2] \\ M_0[M_k/x_k] = L_0[L_1[L'_2[L''_2[L_3[L_4/y_3]/y_2]/x_k]/y_1]/y_0] \end{cases}$$

and $\langle L_1, L'_2[L''_2/x_k], L_3 \rangle$ is a 3-loop. There are two subcases.

(Case 3.1) L'_2 contains a lexical constant. Let us divide \vec{x} into two parts \vec{x}_1 and \vec{x}_2 , where \vec{x}_1 and \vec{x}_2 are sequences made up of the variables in $\text{Fv}(L_0) \cup \text{Fv}(L_1) -$

$\{y_0, y_1\}$ and $\text{Fv}(L'_2)$, respectively. We obtain $N'_0 \sim L_0[L_1/y_0]$ by applying the induction hypothesis to $L_0[L_1/y_0][L_3[L_4/y_3]/y_1, \vec{M}_1/\vec{x}_1]$. We obtain $N''_0 \sim L'_2$ by applying the induction hypothesis to $L'_2[\vec{M}_2/\vec{x}_2]$. Then, $N'_0[N''_0/y_1] \sim M_0$.

(Case 3.2) L''_2 contains a lexical constant. We obtain $N_0 \sim L_0[L_1[L'_2/y_1]/y_0]$ by applying the induction hypothesis to $L_0[L_1[L'_2/y_1]/y_0][\vec{M}'/\vec{x}]$ where \vec{M}' is obtained from \vec{M} by replacing M_k with $L''_2[L_4/y_2]$.

(Case 4) If there are a λ -term M_k in \vec{M} and λ -terms L_1, L_2, L_3, L_4 such that $M_k = L_1[L_2[L_3[L_4/y_3]/y_2]/y_1]$ and $\langle L_2, L_3 \rangle$ is a 2-loop, then it is easy to verify the claim by applying the induction hypothesis to $M_0[\vec{M}'/\vec{x}]$ where \vec{M}' is obtained from \vec{M} by replacing M_k with $L_1[L_2[L_4/y_2]/y_1]$. \square

Proposition 9 implies that every 2ACG generating a string language not containing the empty string has an equivalent L2ACG. Although the construction of \mathcal{G}' from \mathcal{G} as given in the proof of Proposition 9 is not effective, it can be made so. It does not follow, however, that the universal membership for the class of 2ACGs is in NP, since the size of \mathcal{G}' may be exponential in the size of \mathcal{G} .

5 Conclusion and Open Problems

The results in this paper imply that the class of LACLs properly includes the the class of languages generated by linear context-free rewriting systems even though it is a (probably) small subclass of the class of ACLs. There are many related open problems on the formal properties of ACGs:

- the exact generative capacity of the class of ACGs and the complexity of its universal membership problem;
- the relation between the class of LACLs and other subclasses of NP which contain some NP-complete languages, e.g., the class of *indexed languages* [11];

References

- [1] Philippe de Groote. Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155, 2001.
- [2] Philippe de Groote. Tree-adjoining grammars as abstract categorial grammars. In *TAG+6, Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 145–150. Università di Venezia, 2002.
- [3] Philippe de Groote, Bruno Guillaume, and Sylvain Salvati. Vector addition tree automata. In *Proceedings of the 19th Annual IEEE symposium on Logic in Computer Science*, pages 64–73, July 2004.
- [4] Philippe de Groote and Sylvain Pogodalla. m -linear context-free rewriting systems as abstract categorial grammars. In R. T. Oehrle et J. Rogers, editor, *Proceedings of Mathematics of Language - MOL-8, Bloomington, Indiana, U. S.*, pages 71–80, June 2003.

- [5] Philippe de Groote and Sylvain Pogodalla. On the expressive power of abstract categorial grammars: Representing context-free formalisms. *Journal of Logic, Language and Information*, 13(4):421–438, 2004.
- [6] John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979.
- [7] Max I. Kanovich. The complexity of Horn fragments of linear logic. *Annals of Pure and Applied Logic*, 69:195–241, 1994.
- [8] Marcus Kracht. *The Mathematics of Language*, volume 63 of *Studies in Generative Grammar*, pages 447–459. Mouton de Gruyter, 2003.
- [9] Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.
- [10] Reinhard Muskens. Language, lambdas, and logic. In Geert-Jan Kruijff and Richard Oehrle, editors, *Resource Sensitivity in Binding and Anaphora*, Studies in Linguistics and Philosophy, pages 23–54. Kluwer, 2003.
- [11] William C. Rounds. Complexity of recognition in intermediate-level languages. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 145–158, October 1973.

More Algebras for Determiners

R. Zuber

CNRS, Paris

`Richard.Zuber@linguist.jussieu.fr`

Abstract. Some new algebras, which are possible denotations for various determiners, are studied. One of them is the algebra of generalised cardinal quantifiers which is a sub-algebra of conservative quantifiers and which contains cardinal, co-cardinal and proportional quantifiers. In addition some non-conservative quantifiers are studied (symmetric, contrapositional and fixed points with respect to the post-complement). It is shown that co-intersective quantifiers are contrapositional. The analysis is extended to quantifiers of higher types.

1 Introduction

The study of quantification in natural language (NL) within the framework of generalised quantifier theory (GQT) is more than a quarter of century old. Inspired by the results and methods of Mostowski ([13]) it has given rise to many unforeseen secure formal and empirical results against a background of the classical quantificational logic. The range of problems treated in GQT is very large. It concerns not only traditional logical questions such as expressive power, first order definability or variable binding but also the documentation, classification and study of rich variety of quantifiers in different NLs ([1] [5], [6], [7], [8],[10], [11],[12], [15], [16],[18]). This classificatory and explanatory work shows that NL quantifiers exhibit many specific properties and it has been conjectured that some of these properties may characterise quantifiers in all NLs. For instance the property of *conservativity* (see below) has been considered a semantic universal in the sense that all quantifiers in all NLs should satisfy it. More recent empirical research ([7], [18]) shows that conservativity should not be considered as an "absolute" universal but a kind of relative universal. This roughly means that violations of conservativity are not arbitrary but are due to very specific contexts which still impose constraints on quantifiers which are possibly weaker or of a different nature. Unfortunately I will not say much more about this problem.

Another general result in GQT is that many of the constraints are Boolean in nature: this means that the sets of objects satisfying a particular constraint form Boolean algebras. This fact is of course compatible with a general observation that NL grammatical categories have Boolean structure ([9]). What is interesting, however, in this case is the fact that various constraints on quantifiers are often logically related, thus giving rise to an array of sub-algebras

of a fixed Boolean algebra. This means in particular that there are expressions denoting quantifiers belonging to many algebras at the same time. Moreover, as we will see, there are expressions which denote quantifiers which are atoms in one algebra and which are not atoms in the corresponding super-algebra. A look at the possible status of such expressions and their relevance in linguistics does not seem pointless.

The purpose of this paper is to prepare formal tools which can help to cope with these two problems: (1) how the constraints of conservativity, or similar "universalistic" constraints can be relativised or superseded, and (2) what is the linguistic status of expressions denoting in different but related denotational algebras. Although I will use many linguistic examples I am not concerned here with applicational issues but basically by presentation of various algebraic properties of quantifiers considered as possible denotations of NL determiners.

Most results presented in this paper are not technically complex and will be given without proofs as obvious facts. Although many results obtained are also valid for infinite universes I will implicitly assume their finiteness. There are various reasons for this. First, although I am not directly concerned with issues of computational complexity, obviously one can naturally approach such issues only in finite contexts. Second, the finite-infinite distinction surely involves a specific competence which very likely is not a linguistic competence. Finally, there is a large and important class quantifiers, *proportional* quantifiers which can be defined only in finite domains. I discuss this class at some length.

2 Formal Preliminaries

The version of formal semantics adopted here is the one developed by Keenan (cf. [9]). Expressions of category C have their denotational algebra D_C , the set of possible denotations. D_C are atomic (and complete) Boolean algebras. Functional categories denote in $D_{B/A}$. The set of functions from A into B will be noted as $[A \rightarrow B]$. Atomicity of $D_{B/A}$ is inherited from the atomicity of D_B . When there are no constraints on functions from D_A on D_B , atoms of $D_{B/A}$ are determined by atoms of D_B in the following way:

Proposition: For any $a \in D_A$, and for any $\alpha \in AT(D_B)$, the function $f_{a,\alpha} \in D_{B/A}$ defined as $f_{a,\alpha}(x) = \alpha$ if $x = a$, and O_{D_B} otherwise, is an atom of $D_{B/A}$. Furthermore, every element of $D_{B/A}$ contains an atom of this form.

Noun phrases (NPs) denote functions from properties onto truth values. They are called quantifiers of type $\langle 1 \rangle$. They are elements of D_{NP} which are sets of sets. According to the above result, for any property P , the function f_P defined as $f_P(X) = 1$ if $X = P$ and $f_P(X) = O$ if $X \neq P$ is an atom of D_{NP} . Thus atoms of D_{NP} are singletons which contain a set as their unique element.

In GQT quantifiers are, roughly speaking, denotations of NPs or of their syntactic parts. Formally they are relations between sets. They can be expressed as functions into truth-values. For instance quantifiers of type $\langle 1 \rangle$ are functions from $P(E)$, the power set of E , the universe of objects, into $\{0, 1\}$.

In what follows we will be basically interested in the logic of denotations of (unary) nominal determiners. These are expressions (like *every*, *no*) which combine with common nouns to form noun phrases. Thus, semantically, they are functions from $P(E)$ onto type $\langle 1 \rangle$ quantifiers. They are called type $\langle 1, 1 \rangle$ quantifiers. These quantifiers can be viewed as binary relations on sets. Indeed a type $\langle 1, 1 \rangle$ quantifier F , which is a function in $[P(E) \rightarrow [P(E) \rightarrow \{0, 1\}]]$ corresponds to the binary relation Q between sets defined by $QXY \Leftrightarrow F(X)(Y) = 1$. Let us denote the set of all type $\langle 1, 1 \rangle$ quantifiers, or functions from $[P(E) \rightarrow [P(E) \rightarrow \{0, 1\}]]$ by $PDET$. This set forms, given Proposition, an atomic Boolean algebra with Boolean operations defined pointwise.

Notice that Boolean character of quantifiers and their arguments allows us to distinguish two types of negations or complements. The first is the usual Boolean complement. The second, called *postcomplement* is defined by the complement of one of its arguments. More precisely, let F be a type $\langle 1, 1 \rangle$ quantifier. Then its postcomplement, noted $F - not$, is the type $\langle 1, 1 \rangle$ quantifier defined as $F - not(X)(Y) = F(X)(Y')$. Similarly, if Q is a type $\langle 1 \rangle$ quantifier, then its postcomplement $Q - not$ is defined as $Q - not(Y) = Q(Y')$, or, equivalently: $Y \in Q - not$ iff $Y' \in Q$. Roughly speaking, syntactically, the postcomplement corresponds to the negation of verb phrases. In what follows we will use in particular the fact that postcomplements of meets or joins of two functions are respectively meets and joins of postcomplements of these functions.

One of the best known constraints on possible denotations of determiners is the constraint of conservativity. Since I am going to use two notions of conservativity, which depend on which specific intersection of arguments is taken into consideration, I will refer to this "classical" notion of conservativity by $CONS1$. By definition:

D1: $F \in CONS1$ iff for any property X, Y and Z if $X \cap Y = X \cap Z$ then $F(X)(Y) = F(X)(Z)$

There are two other equivalent definitions of conservativity indicated by the following facts:

Fact 1: (cf. [9]) $F \in CONS1$ iff for any property X, Y , $F(X)(Y) = F(X)(X \cap Y)$

Fact 2: $F \in CONS1$ iff for any property X, Y one has $F(X)(Y) = F(X)(X' \cup Y)$

The algebra $CONS1$ has two sub-algebras, the algebra INT of intersective functions, and the algebra $CO - INT$ of co-intersective functions ([5]):

D2: $F \in INT$, iff for all properties X, Y, Z and W , if $X \cap Y = Z \cap W$ then $F(X)(Y)$ is true iff $F(Z)(W)$ is true.

D3: $F \in CO - INT$ iff for all properties X, Y, Z and W , if $X - Y = Z - W$ then $F(X)(Y) = F(Z)(W)$.

Both sets INT and $CO-INT$ form atomic (and complete) Boolean algebras. Their atoms are determined by a property: for any property P the function F_P such that $F_P(X)(Y) = 1$ iff $X \cap Y = P$ is an atom of INT and the function F_P such that $F_P(X)(Y) = 1$ iff $X - Y = P$ is an atom of $CO-INT$. Exclusion determiners (cf.[16]) denote such atomic functions: *no...except Leo and Lea* denotes an atomic intersepective function determined by the set composed of two elements, Leo and Lea.

Algebras INT and $CO-INT$ are important for two reasons. First, as shown in [5] the algebra $CONS1$ is a Boolean closure of INT and $CO-INT$. Second, Keenan also shows that quantifiers belonging to INT or to $CO-INT$ are exactly those which are sortally reducible. This means that it is possible to replace *salva veritate* the first argument of these quantifiers by the universal property E and replace the second argument by a Boolean combination of the first argument with the second: if $F \in INT$ then $F(X)(Y) = F(E)(X \cap Y)$ and if $F \in CO-CARD$ then $F(X)(Y) = F(E)(X' \cup Y)$.

Sortally reducible quantifiers have a more general property $FIDQ$ or the property of freely increasable domain of quantification. It is defined in D4 and proposition 0 shows that members of INT and of $CO-INT$ have this property:

D4: $F \in FIDQ$ iff there is a binary Boolean function h such that $F(X)(Y) = F(X_1)(h(X_1, Y))$, for any X_1 such that $X \subseteq X_1$.

Proposition 0: $INT \cup CO-INT \subseteq FIDQ$

Since $SOME$ is an intersepective quantifier it follows from the above propositions that $SOME$ is a member of $FIDQ$ and thus that *Some wild cats are dangerous* is equivalent to *Some cats are wild and dangerous*.

The algebra INT contains a sub-algebra $CARD$ of cardinal functions: they are denotations of, roughly speaking, various numerals. By definition:

D4: $F \in CARD$ iff for all properties X, Y, W and Z , if $|X \cap Y| = |Z \cap W|$ then $F(X)(Y)$ is true iff $F(W)(Z)$ is true.

Atoms of $CARD$ are functions f_α , such that $f_\alpha(X)(Y) = 1$ iff $|X \cap Y| = \alpha$, for α a cardinal,. Thus the determiner *exactly n* denotes an atomic cardinal function.

As might be expected the algebra $CO-INT$ has an analogous sub-algebra. This is the algebra $CO-CARD$ of co-cardinal functions:

D5: $F \in CO-CARD$ iff for all properties X, Y, W and Z , if $|X - Y| = |W - Z|$ then $F(X)(Y) = F(W)(Z)$

Determiners like *every...except five* denote co-cardinal functions.

All the algebras presented above form the basic stock of algebras for unary NL determiners (cf.[12]). We will study now some other semantic properties of determiners and denotational algebras determined by them.

3 Other Algebras

Let me start the presentation of other algebras by a somewhat less known algebra *GCARD* of generalised cardinals. It was introduced in [19] in order to account for some properties of the definite article *the*. By definition:

D6: $F \in GCARD$ iff for all properties X, Y, Z if $|X \cap Y| = |X \cap Z|$ then $F(X)(Y) = F(X)(Z)$.

Obviously the algebra *GCARD1* is a proper sub-algebra of *CONS1* and contains as proper sub-algebras *CARD* and *CO - CARD*. Furthermore we have:

Proposition 1: The algebra *GCARD* is atomic and its atoms are determined as follows: for any $n \leq |E|$ and any $A \subseteq E$, the function $at_{A,n}$ such that $at_{A,n}(X)(Y) = 1$ iff $X = A$ and $|X \cap Y| = n$ is an atom of *GCARD*.

Other properties of *GCARD1* relate it to other algebras I will introduce now. If *PDET* is the algebra of possible denotations of determiners, that is all functions in $[P(E) \rightarrow [P(E) \rightarrow \{0, 1\}]]$ then conservative functions form a sub-algebra of *PDET*. There are obviously many other sub-algebras of *PDET*. Let us have a look at some of them. First we have a rather natural property of symmetry determining symmetric determiners:

D7: $F \in SYM$ iff for all properties X, Y one has $F(X)(Y) = F(Y)(X)$

We notice that all intersective functions are symmetric though there are symmetric ones which are not intersective. An analogous property which all co-intersective functions have is the property *CONTR* of being contrapositional:

D8: $F \in CONTR$ iff for all properties X, Y one has $F(X)(Y) = F(Y')(X')$.

The quantifier *ALL* is obviously contrapositional. Moreover we have:

Proposition 2: $CONS1 \cap SYM = INT$

Proposition 3: $CONS1 \cap CONTR = CO - INT$

We prove proposition 3, more precisely its "from left to right" part. Suppose that for some arbitrary X, Y, W and Z we have (1) $X - Y = W - Z$. This is equivalent to (2) $E \cap (X' \cup Y) = E \cap (W' \cup Z)$. Then:

$$\begin{aligned}
 F(X)(Y) &= F(X)(Y \cup X') = && - CONS1 \text{ (fact 2)} \\
 &= F(X - Y)(X') = && - CONTR \\
 &= F(X - Y)(\emptyset) = && - CONS1 \text{ (fact 1)} \\
 &= F(E)(X' \cup Y) = && - CONTR \\
 &= F(E)(W' \cup Z) = && - CONS1 \text{ (D 1 and (2))} \\
 &= F(W - Z)(\emptyset) = && - CONTR \\
 &= F(W - Z)(W - Z \cap W') = && - \text{set theoretical equivalence}
 \end{aligned}$$

$$\begin{aligned}
&= F(W - Z)(W') = && \text{-CONS1 (fact 1)} \\
&= F(W)(W' \cup Z) = && \text{-CONTR} \\
&= F(W)(Z)
\end{aligned}$$

Thus all co-intersective functions are contrapositional.

There are various properties relating the algebra *GCARD* to *CARD* and *CO - CARD* in a very similar way in which the algebra *CONS1* is related to *INT* and *CO - INT*. In particular we have:

Proposition 4: $GCARD1 \cap SYM = CARD$

Proposition 5: $GCARD1 \cap CONTR = CO - CARD$

The proofs of propositions 4 and 5 are similar to the proof of proposition 3.

Propositions 4 and 5 can be used to give a new characterisation of *logical* quantifiers: these are quantifiers which are classically conservative, satisfy the condition of extension and are permutation invariant. Keenan and Westerstahl ([12]) indicate that logical quantifiers are Boolean combinations of cardinal and co-cardinal quantifiers. Given propositions 4 and 5 we have the following characterisation of logical quantifiers (cf. [20]):

Proposition 6: A logical type $\langle 1, 1 \rangle$ quantifier is a Boolean combination of symmetric generalised cardinals and contrapositional generalised cardinals.

Recent research on natural language type $\langle 1, 1 \rangle$ quantifiers shows that in fact there are natural classes of such quantifiers which need not to be conservative in the classical sense. In particular such quantifiers can naturally occur in existential contexts (cf. [7]). Similarly Zuber ([17]) strongly suggests that in some languages there are non-conservative determiners which are, however, systematically related to specific conservative ones: they are related by the relation of argument inversion. Let Q be a type $\langle 1, 1 \rangle$ quantifier. Then:

D 10: Q^i is the inverse of Q iff $Q^i(X)(Y) = Q(Y)(X)$

For instance the determiner *apart from Leo only...* denotes a quantifier which is the inverse of the quantifier denoted by *every...except Leo*. One observes also that symmetric quantifiers are their own inverses. Similarly, inversion preserves contraposition. On the other hand inversion does not preserve (classical) conservativity. It is interesting that inversion of classically conservative quantifiers gives rise to a special class called *CONS2* ([7]):

D 11: $F \in CONS2$ iff for all properties X, Y, Z if $X \cap Z = Y \cap Z$ then $F(X)(Z) = F(Y)(Z)$

The following fact is obvious now:

Fact 3: $F \in CONS1$ iff $F^i \in CONS2$

Consider the expression *mostly* as a nominal determiner. In this case it denotes the quantifier *MOSTLY* such that $MOSTLY(X)(Y) = 1$ iff $|Y \cap X| > |Y - X|$.

This means that *MOSTLY* is the inverse of *MOST* and thus *Mostly Germans are bear drinkers* is equivalent to *Most bear drinkers are German*.

As with classical conservativity quantifiers satisfying *CONS2* can be characterised in two other equivalent ways. thus we have:

Fact 4: $F \in \text{CONS2}$ iff for any X, Y one has $F(X)(Y) = F(X \cap Y)(Y)$

Fact 5: $F \in \text{CONS2}$ iff for any X, Y one has $F(X)(Y) = F(X \cup Y')(Y)$

It is easy to prove proposition 6 from which, in conjunction with proposition 2, follows proposition 7 (stated in Keenan [7]):

Proposition 7: $\text{CONS1} \cap \text{CONS2} \subseteq \text{SYM}$

Proposition 8: $\text{CONS1} \cap \text{CONS2} = \text{INT}$

In order to obtain a similar result for co-intersective quantifiers we need:

D 12: $F \in \text{CO} - \text{CONS2}$ iff for all properties X, Y, Z if $X - Z = Y - Z$ then $F(X)(Z) = F(Y)(Z)$

We have the following facts and propositions concerning $\text{CO} - \text{CONS2}$. We prove Proposition 9, the proof of Proposition 10 being similar:

Fact 6: $F \in \text{CO} - \text{CONS2}$ iff for any property X, Y one has $F(X)(Y) = F(X - Y)(Y)$

Fact 7: $F \in \text{CO} - \text{CONS2}$ iff for any property X, Y one has $F(X)(Y) = F(X \cup Y)(Y)$

Proposition 9: $\text{CONS1} \cap \text{CO} - \text{CONS2} \subseteq \text{CONTR}$

Proposition 10: $\text{CONS1} \cap \text{CO} - \text{CONS2} = \text{CO} - \text{INT}$

Proof of Proposition 9: Suppose $F \in \text{CONS1} \cap \text{CO} - \text{CONS2}$. Then

$$\begin{aligned}
 F(X)(Y) &= F(X - Y)(Y) && \text{- fact 5} \\
 &= F(X - Y)(X' \cup Y) && \text{- classical conservativity (fact 2)} \\
 &= F((X' \cup Y)' \cap Y')(X' \cup Y) && \text{- set theoretical equivalence} \\
 &= F(Y')(X' \cup Y) && \text{- fact 5} \\
 &= F(Y')(X') && \text{- fact 2}
 \end{aligned}$$

Given that co-intersective functions are classically conservative and contrapositive we have to prove the "from left to right" part. So suppose that for some arbitrary sets X, Y, W and Z we have $X - Y = W - Z$ and $F \in \text{CONS1} \cap \text{CO} - \text{CONS2}$. Then:

$$\begin{aligned}
 F(X)(Y) &= F(X)(X' \cup Y) && \text{- fact 2} \\
 &= F(X \cap Y')(X') && \text{- proposition 9} \\
 &= F(X \cap Y')(\emptyset) && \text{- CONS1 (fact 1)} \\
 &= F(W - Z)(W \cap Z' \cap W') && \text{- supposition and a set theoretical equivalence}
 \end{aligned}$$

$=F(W - Z)(W') =$	- classical conservativity
$=F(W)(W' \cup Z) =$	- <i>CONTR</i>
$=F(W)(Z)$	-classical conservativity (fact 2)

There are two other algebras I would like to discuss briefly. The first is the algebra *PROPORT* of *proportional* determiners. By definition ([6]):

D13: $F \in \text{PROPORT}$ iff for all properties X, Y, W, Z if $|W| \times |X \cap Y| = |X| \times |W \cap Z|$ then $F(X)(Y) = F(W)(Z)$

Proportional quantifiers have the following properties:

Fact 8: Proportional quantifiers form a sub-algebra of *GCARD*.

Fact 9: $F \in \text{PROPORT}$ iff $F - \text{not} \in \text{PROPORT}$

Proposition 11: For $1 \leq m < n$ the functions $F_{m,n}$ such that $F_{m,n}(X)(Y) = 1$ iff $|X \cap Y|/|X| = n/m$ are atoms of *PROPORT*

There are various examples of proportional determiners one of the best known being *most*, in the sense *more than half*. The quantifiers denoted by the determiners like *exactly n percent* are atoms of *PROPORT*. Other examples of proportional quantifiers include *six out of twelve*, *exactly 20 %*, *all but a tenth*, etc. According to fact 9, postcomplements of proportional quantifiers are also proportional quantifiers.

It follows from the fact 8 that the class *PROPORT* includes various *improperly* proportional quantifiers. For instance *less than zero percent*, *at least zero percent*, *zero percent*, *more than zero percent*, *hundred percent*, *more than fifty and less than hundred percent*, etc. are (denote), according to D13, proportional quantifiers. These quantifiers correspond to the zero, the unit element of the algebra *PROPORT*, and the quantifiers *SOME*, *NO*, *ALL* and *MOST BUT NOT ALL*, respectively. We know that these quantifiers belong also to other algebras. For instance *SOME* and *NO* are at the same time cardinal, intersective, symmetric, generalised cardinal and conservative.

In spite of the above examples it is not true in general that cardinal or co-cardinal functions are proportional quantifiers. The exact relationship between *CARD*, *GCARD* and *PROPORT* remains to be established. Such a relationship might be helpful to our better understanding of conditions for the first order definissability since proportional quantifiers are precisely known as being in most cases not definable in the first order logic (see [12] for a discussion).

The last algebra I want to mention is the algebra *FPPCPL* of quantifiers which are fixed points with respect to the operation of postcomplementation:

D 14: $F \in \text{FPPCPL}$ iff for all properties X, Y one has $F(X)(Y) = F(X)(Y')$

A natural example of a *FPPCPL* is given by the denotation of *half of*. The following proposition is a consequence of the fact that postcomplements preserve meets and joins, allows us to obtain other natural examples of such quantifiers:

Proposition 12: For any F type $\langle 1, 1 \rangle$ we have $F \vee F - not \in FPPCPL$ and $F \wedge F - not \in FPPCPL$

Thus, given that $SOME - not$ equals to $NOT - ALL$ the complex quantifier $SOME BUT NOT - ALL$ is a fixed point with respect to postcomplement. Similarly a conjunction or a disjunction of an intersective and the corresponding co-intersective forms such a quantifier. This is for instance the case with $FIVE AND FIVE - not$ or $TEN OR TEN - not$.

Although the above examples of quantifiers which are fixed points with respect to postcomplements are also examples of (classically) conservative quantifiers, the definition D 14 and Proposition 12 are not restricted to such quantifiers. The following proposition shows that it is possible to define the class $FPPCPL$ in the general unrestricted case using the definitional format adopted here:

Proposition 13: Let $F \in PDET$. Then $F \in FPPCPL$ iff there exists a binary function \otimes on sets for which $X \otimes Y = X \otimes Y'$, for any X, Y , and if $X \otimes Y = X \otimes Z$ then $F(X)(Y) = F(X)(Z)$

Proof \Rightarrow . Let $F \in FPPCPL$. Define a binary function " \otimes " as follows: $X \otimes Y = F(X)(Y)$. Obviously $X \otimes Y = X \otimes Y'$. It is easy to check that this function is the function we need in order for the conclusion to be satisfied.

\Leftarrow If \otimes is such that $X \otimes Y = X \otimes Y'$ and F satisfies the necessary condition, then $F(X)(Y) = F(X)(Y')$ and thus $F \in FPPCPL$.

Proposition 13 shows how we can define, for most classes of quantifiers we have distinguished, their fixed point sub-classes. What is interesting is the fact that this can be done using the general definitional format adopted here. Thus:

Proposition 14: For any type $\langle 1, 1 \rangle$ quantifier F , $F \in CONS1 \cap FPPCPL$ iff for any X, Y, Z , $F(X)(Y) = F(X)(Z)$ whenever $X \cap Y = X - Z$.

Similarly we can define, using the following propositions, quantifiers which are proportional, cardinal, co-cardinal, generalised cardinal, etc and at the same time are fixed points with respect to postcomplements. Here are just some such definitional properties illustrating this claim:

Proposition 15: $F \in PROPORT \cap FPPCPL$ iff $F(X)(Y) = F(X)(Z)$ whenever for any X, Y, W, Z , $|W| \times |X \cap Y| = |X| \times |W - Z|$.

Proposition 16: $F \in INT \cap FPPCPL$ iff $F(X)(Y) = F(W)(Z)$ whenever $X \cap Y = W - Z$.

Proposition 17: $F \in GCARD \cap FPPCPL$ iff $F(X)(Y) = F(X)(Z)$ whenever $|X \cap Y| = |X - Z|$.

Proposition 18: $F \in CARD \cap FPPCPL$ iff $F(X)(Y) = F(X)(Z)$ whenever $|X \cap Y| = |W - Z|$.

The following fact limits the usefulness of some of these propositions:

Fact 10: $INT \cap FPPCPL = CO - INT$

This fact indicates that intersective or co-intersective functions which are at the same time fixed points for postcomplement are just the trivial elements of the algebra, the unit and the zero elements. In other words quantifiers which satisfy the condition $F(X)(Y) = F(W)(Z)$ if $X \cap Y = W - Z$ are just the trivial elements of the algebra $PDET$.

Let me give some examples of non-trivial members of $FPPCPL$. They are necessarily neither intersective nor co-intersective. Thus for any k, l , such that $k + l = |E|$ define a type $\langle 1, 1 \rangle$ quantifier $F_{k,l}$ as $F_{k,l}(X)(Y) = 1$ iff $|X| = k + l$ and $|X \cap Y| = k$ or $|X| = k + l$ and $|X \cap Y| = l$. This function, which is denoted by the somewhat complex determiner like *Among the (k+l)...exactly k... or exactly l...* is a generalised cardinal function member of $FPPCPL$.

Another example is a determiner, also not very natural, like *either only Leo or else any other...*, (with *either...or* understood as exclusive disjunction) as in the following noun phrase: *either only Leo or else every other student*. This determiner denotes a classically conservative function belonging to $FPPCPL$.

The two preceding examples show the way to obtain proportional quantifiers members of $FPPCPL$: we have just to make disjunctions which in some sense exhaust all possibilities. Thus the determiners like *40 or 60 percent of* or *one third or two thirds of* are determiners denoting such quantifiers.

One of the interests of quantifiers which are fixed points for postcomplements is that they may have linguistic applications showing the usefulness of various generalisations concerning type $\langle 1, 1 \rangle$ quantifiers. These quantifiers are considered here as functions taking two sets into truth values. What is essential in most cases is not the fact that such functions have truth-values as their values but the fact that the results of applications of these functions are equal or not to results of application of other functions. So obviously we can generalize most of type $\langle 1, 1 \rangle$ quantifiers to functions from pairs of sets onto type of objects other than truth-values. Such a move is useful if we want to analyse "interrogative determiners" in a way similar to the analysis of traditional "declarative" determiners (cf. [4]). Notice now that in questions fixed points may be involved. For instance there is a level of analysis in which the two interrogatives *Which numbers are greater than n ?* and *Which numbers are not greater than n ?* should be considered as being equivalent. Since this equivalence is not the logical equivalence, i.e. not equality of truth-values, a generalisation of involved notions is necessary. This means in particular that the interrogative determiner *which* denotes a generalised function which is fixed with respect to (generalised) postcomplements. The fact that such equivalence need not hold for all interrogatives (cf. [17]) just indicates that there are various types of interrogative determiners.

4 Higher Type Quantifiers

Up to now we were basically interested in the denotations of determiners taking one common noun to form an NP. Binary and, more generally n -ary determiners, have been introduced and extensively studied in [10] and, in somewhat different perspective, in [3]. Their denotations form atomic Boolean algebras. It is interesting that there is a systematic way to form n -ary determiners from Boolean conjunctions and unary determiners ([10]). For instance the complex determiners like *all...and...* or *most...and...* should be considered as binary determiners because the NP *most students and teachers* does not mean *most individuals who are students and teachers* but rather *most students and most teachers*.

What is important, however, that there exist intrinsically n -ary determiners, that is determiners which are not Boolean compositions of unary ones. In other words we should distinguish between *reducible* and *non-reducible* denotations of n -ary determiners. More precisely, restricting ourselves to binary determiners and quantifiers of type $\langle\langle 1, 1 \rangle, 1\rangle$ we have (cf. [3]):

D15: F of type $\langle\langle 1, 1 \rangle, 1\rangle$ is $\langle 1, 1 \rangle$ reducible iff there are two type $\langle 1, 1 \rangle$ quantifiers F_1 and F_2 and a binary Boolean function h such that for all sets X, Y, Z , $F(X, Y)(Z) = h(F_1(X)(Z), F_2(Y)(Z))$

Clearly the quantifiers *MOST...AND...* and *ALL...AND...* are reducible. For reducible quantifiers the following proposition holds:

Proposition 19: Reducible type $\langle\langle 1, 1 \rangle, 1\rangle$ quantifiers form a Boolean algebra.

Beghelli ([3]) shows that comparative determiners like *more...than...* do not denote reducible quantifiers. It follows from proposition 19 that *not more...than...* also denote unreducible quantifiers.

There are thus n -ary determiners denoting higher type quantifiers. We will study only some higher type quantifiers: the ones which correspond to binary relations with relations as arguments. So in general they are of the type $\langle 1^k, 1^l \rangle$ which corresponds to binary relations between k -ary relations and l -ary relations. Even more specifically we will consider quantifiers of this type assuming that either $k = 1$ or $l = 1$. Determiners denoting such quantifiers are sometimes called *structured* (n -ary) determiners.

Most of the properties of type $\langle 1, 1 \rangle$ quantifiers we studied generalise to higher type quantifiers, if we keep in mind the distinction between "subject" arguments and the "predicate" arguments of a quantifier. The idea, roughly, is that instead of looking at intersections, or cardinalities of the intersection, of the only argument of an unary determiner with the predicative argument we have to look at intersections, or the cardinalities of intersections, of every "subject" argument of the n -ary determiner, with the predicative argument (or predicative arguments). Let us first define the post-complement of a higher type quantifier:

D 16: If F is a type $\langle 1^k, 1^l \rangle$ quantifier then $F - not$ is that type $\langle 1^k, 1^l \rangle$ quantifier for which $F - n(X_1, \dots, X_k)(Y_1, \dots, Y_l) = F(X_1, \dots, X_k)(Y'_1, \dots, Y'_l)$

For other definitions we restrict one of the arguments of the relation to a set. Reducible higher type quantifiers give us a hint as to exactly which intersections should be taken into consideration in such definitions. This is because, as we observe, reducible higher type quantifiers inherit some of their properties from lower order one to which they are reducible. Thus, roughly, if a higher type quantifier is reducible to two conservative quantifiers of lower type then it is reasonable to suppose that the higher type quantifier is conservative. The same with other properties. Having this in mind we get various definitions of higher type quantifiers. In fact some of such definitions have already been applied (see [10],[3],[7]). Thus let D be a type $\langle 1^n, 1 \rangle$ quantifier, that is a function from n -tuples of subsets of E to type $\langle 1 \rangle$ functions (i.e. denotations of NPs) over E and let $CONS1_{\langle 1^n, 1 \rangle}$ be the set of (classically) conservative denotations of n -ary structured determiners. Then

D17: $D \in CONS1_{\langle 1^n, 1 \rangle}$ iff $\forall X_i, Y_1, Y_2, D(X_1, \dots, X_n)(Y_1) = D(X_1, \dots, X_n)(Y_2)$ if $X_i \cap Y_1 = X_i \cap Y_2$, for every $1 \leq i \leq n$.

As in the case of type $\langle 1, 1 \rangle$ quantifiers we have equivalent definitions of conservativity for higher type quantifiers. Thus, given that $A_i \cap B = A_i \cap B \cap \bigcup_n A_i$ for all $A_i, B \in E$, we have the following propositions concerning conservativity for higher types:

Proposition 20: $D \in CONS1_{\langle 1^n, 1 \rangle}$ iff $D(X_1, \dots, X_n)(Y) = D(X_1, \dots, X_n)(Y \cap \bigcup_n A_i)$

Proposition 21: $D \in CONS1_{\langle 1^n, 1 \rangle}$ iff $D(X_1, \dots, X_n)(Y) = D(X_1, \dots, X_n)(Y \cap \bigcap_n A'_i)$

In a similar way we define other higher type quantifiers. For instance intersection, co-intersective, cardinal, co-cardinal, and generalised cardinal functions are defined as follows;

D 18: $D \in INT_{\langle 1^n, 1 \rangle}$ iff $\forall X_i, Y_i, Z_1, Z_2, D(X_1, \dots, X_n)(Z_1) = D(Y_1, \dots, Y_n)(Z_2)$ if $X_i \cap Z_1 = Y_i \cap Z_2$, for every $1 \leq i \leq n$.

D 19: $D \in CO - INT_{\langle 1^n, 1 \rangle}$ iff whenever $X_i - Z_1 = Y_i - Z_2, \forall X_i, Y_i, Z_1, Z_2$, we have $D(X_1, \dots, X_n)(Z_1) = D(Y_1, \dots, Y_n)(Z_2)$

D20: $D \in CARD_{\langle 1^n, 1 \rangle}$ iff $\forall X_i, Y_i, Z_1, Z_2, D(X_1, \dots, X_n)(Z_1) = D(Y_1, \dots, Y_n)(Z_2)$ if $|X_i \cap Z_1| = |Y_i \cap Z_2|$, for every $1 \leq i \leq n$.

D21: $D \in CO - CARD_{\langle 1^n, 1 \rangle}$ iff $\forall X_i, Y_i, Z_1, Z_2$, if $|X_i - Z_1| = |Y_i - Z_2|$, for every $1 \leq i \leq n$, then $D(X_1, \dots, X_n)(Z_1) = D(Y_1, \dots, Y_n)(Z_2)$

D22: $D \in GCARD_{\langle 1^n, 1 \rangle}$ iff $\forall X_i, Y_1, Y_2, D(X_1, \dots, X_n)(Y_1) = D(X_1, \dots, X_n)(Y_2)$ if $|X_i \cap Y_1| = |Y_i \cap Y_2|$, for every $1 \leq i \leq n$.

For these quantifiers the following facts hold:

Fact 11: $CONS1_{\langle 1^n, 1 \rangle}, GCARD_{\langle 1^n, 1 \rangle}, INT_{\langle 1^n, 1 \rangle}, CO-INT_{\langle 1^n, 1 \rangle}, CARD_{\langle 1^n, 1 \rangle}, CO-CARD_{\langle 1^n, 1 \rangle}$ form Boolean algebras.

Fact 12: $CARD_{\langle 1^n, 1 \rangle} \subseteq INT_{\langle 1^n, 1 \rangle} \subseteq CONS1_{\langle 1^n, 1 \rangle}$

Fact 13: $CARD_{\langle 1^n, 1 \rangle} \cup CO-CARD_{\langle 1^n, 1 \rangle} \subseteq GCARD_{\langle 1^n, 1 \rangle} \subseteq CONS1_{\langle 1^n, 1 \rangle}$

In fact more can be shown. Using Proposition 20 and some results from [10] we can indicate atoms of various higher type algebras showing that they are atomic. Thus we have:

Fact 14: Let $1 \leq i \leq n$, $P_i \subseteq E$ and $P \subseteq \bigcup_i P_i$. Then the function $D_{P_1, \dots, P_n, P}$ such that $D_{P_1, \dots, P_n, P}(X_1, \dots, X_n)(Y) = 1$ iff $X_i = P_i$ and $P = Y \cap \bigcup_i X_i$ is an atom of $CONS1_{\langle 1^n, 1 \rangle}$. All atoms of $CONS1_{\langle 1^n, 1 \rangle}$ are of this form.

Fact 15: The function F_{P_1, \dots, P_n} such that $F_{P_1, \dots, P_n}(X_1, \dots, X_n)(Y) = 1$ iff $X_i \cap Y = P_i$ are atoms of $INT_{\langle 1^n, 1 \rangle}$.

Fact 16: The function F_{P_1, \dots, P_n} such that $F_{P_1, \dots, P_n}(X_1, \dots, X_n)(Y) = 1$ iff $X_i Y' = P_i$ are atoms of $CO-INT_{\langle 1^n, 1 \rangle}$.

Thus atoms of algebras of higher type intersective and co-intersective functions are determined by sequences of sets. For $n = 1$ we obtain from these facts atoms of the algebras of the corresponding type $\langle 1, 1 \rangle$ quantifiers. In particular for any $A, B \subseteq E$ such that $B \subseteq A$, the function $F_{A, B}$ such that $F_{A, B}(X)(Y) = 1$ iff $X = A$ and $Y \cap X = B$ is an atom of $CONS1$ (cf. [5]). Furthermore, atoms INT and $CO-INT$ are related to atoms of $CONS1$ in the way indicated in

Proposition 22: Let $F_{P_1, \dots, P_n, P}$ be an atom of $CONS1_{\langle 1^n, 1 \rangle}$, $G_{P_1 \cap P, \dots, P_n \cap P}$ be an atom of $INT_{\langle 1^n, 1 \rangle}$ and $H_{P_1 - P, \dots, P_n - P}$ be an atom of $CO-INT_{\langle 1^n, 1 \rangle}$. Then $F_{P_1, \dots, P_n, P} = G_{P_1 \cap P, \dots, P_n \cap P} \wedge H_{P_1 - P, \dots, P_n - P}$.

^s

We observe also that the application of the operation of postcomplementation to higher type quantifiers holds results similar to the ones we get in the case of simple quantifiers. Thus:

Fact 17: $D \in INT_{\langle 1^n, 1 \rangle}$ iff $D - not \in CO-INT_{\langle 1^n, 1 \rangle}$.

It is also possible to generalise some other properties studied above to non unary determiners, in particular properties of being symmetric or contrapositional. Recall that quantifiers are relations between relations. Since some of these relations are binary they can be symmetric. Furthermore, since such relations relate Boolean objects, to check whether they relate permuted complements of their arguments is possible as well. This leads to the generalised notion of contrapositional of a quantifier as well, especially when the objects standing in the binary relation are of the same type. We know that quantifiers corresponding to binary relations may relate objects of different types. For instance, type $\langle 1, 1 \rangle, 1$ quantifiers are relations between binary relations and sets (as in *more students than teachers are vegetarians* and type $\langle 1, \langle 1, 1 \rangle \rangle$ quantifiers are relations between

sets and binary relations (as in *more vegetarians are students than teachers*). To extend the notion of symmetry and contrapositionality to such cases we need general definitions to be given in the framework we used in this paper:

D 23: A type $\langle 1^n, 1 \rangle$ quantifier D is symmetric iff there exists a binary commutative function \otimes such that $\forall X_i, Y_i, Z_1, Z_2, D(X_1, \dots, X_n)(Z_1) = D(Y_1, \dots, Y_n)(Z_2)$ if $X_i \otimes Z_1 = Y_i \otimes Z_2$, for every $1 \leq i \leq n$.

D 24: A type $\langle 1^n, 1 \rangle$ quantifier D is contrapositional iff there exists a binary commutative function \otimes such that if $X_i \otimes Z'_1 = Y_i \otimes Z'_2, \forall X_i, Y_i, Z_1, Z_2$, all $1 \leq i \leq n$ then $D(X_1, \dots, X_n)(Z_1) = D(Y_1, \dots, Y_n)(Z_2)$

It is easy to check that if $n = 1$ we get from D 23 the "ordinary" symmetry for simple quantifiers and from D 24 an "ordinary" contraposition for simple quantifiers. For instance one observes that post-complements of symmetric quantifiers are contrapositional and *vice versa*. Furthermore the following propositions are true:

Proposition 23: Let $F \in PDET_{\langle 1^n, 1 \rangle}, G \in PDET_{\langle 1, 1^n \rangle}$ and $F(X_1, \dots, X_n)(Y) = G(Y)(X_1, \dots, X_n)$. Then F is symmetric iff G is symmetric.

Proposition 24: Let $F \in PDET_{\langle 1^n, 1 \rangle}, G \in PDET_{\langle 1, 1^n \rangle}$ and $F(X_1, \dots, X_n)(Y) = G(Y')(X'_1, \dots, X'_n)$. Then F is contrapositional iff G is contrapositional.

Proportional n-ary determiners represent a more difficult case since proportionality in this case may depend on whether we consider reducible or unreducible quantifiers. We have seen that determiners like *most* or *20 percent of* denote proportional type $\langle 1, 1 \rangle$ quantifiers. The question we can ask now is the following: are the reducible type $\langle \langle 1, 1 \rangle, 1 \rangle$ quantifiers based on them and the corresponding type $\langle 1 \rangle$ quantifiers also proportional? Should we consider for instance that the NP *most students and teachers* (considered as containing a binary determiner) involves proportional quantification. I think the answer should be negative. The reason is that in this case there is no proportional relation between the two arguments: *students* and *teachers*. The truth value of the sentence *Most students and teachers danced* does not depend on any proportional relation between the number of students and the number of teachers. This is different from the case of unreducible determiners like *twice as many students as teachers* since in the latter example there is a (non trivial) proportional relation between the number of students and the number of teachers. Any general definition of proportional quantifiers should account for this difference. This leads to the following definition of type $\langle \langle 1, 1 \rangle, 1 \rangle$ proportional quantifiers:

D 25: $D \in PROPORT_{\langle 1^2, 1 \rangle}$ iff for all $X_1, X_2, Y_1, Y_2, Z_1, Z_2, D(X_1, X_2)(Z_1) = D(Y_1, Y_2)(Z_2)$ whenever $|Y_1| \times |Y_2| \times |X_1 \cap Z_1| = |X_1| \times |X_2| \times |Y_1 \cap Z_2|$ and $|Y_1| \times |Y_2| \times |X_2 \cap Z_1| = |X_1| \times |X_2| \times |Y_2 \cap Z_2|$

One checks by calculation that according to D 22 determiners like *twice as many... as...* or *20 percent more...than...* denote proportional quantifiers whereas

the reducible quantifier *MOST...AND...* is not proportional. Furthermore the following proposition is true:

Proposition 25: $PROPORT_{\langle 1^2, 1 \rangle}$ is a sub-algebra of $GCARD_{\langle 1^2, 1 \rangle}$

Incidentally the analogue of fact 9 (that proportionality is closed with respect to postcomplements) does not hold for the higher type quantifiers. We observe also that even if definition D 25 can be formally extended to n-ary cases it is not clear what would be empirical content of such extensions. In fact it seems that proportional quantifiers still need to be studied more deeply.

5 Conclusive Remarks

Most of this paper is occupied by the presentation of some new or more general algebras representing possible denotations for nominal determiners, not only unary ones. I left aside the problem of how many quantifiers there are in various algebras when different constraints defining them are taken into account. The exact number of quantifiers, members of a given fixed algebra, depends of course on the number of individuals in the model and a specific definitional restriction. Some of these numbers have been established for various specific cases ([15], [10], [2], [6], [12], [8]).

The algebras which were studied here are basically *GCARD*, *SYM*, *CONTR*, *FPPCPL* and, to some extent, *PROPORT*. One of the reasons for introducing them is that they give rise to natural inferential patterns. Another reason is that all of them can be considered, it seems to me, as representing "borderline cases" between "linguistically useful" and "logically natural" quantifiers. For instance, concerning *SYM*, *CONTR* and *FPPCPL* we observe that they are defined by very natural "logical" properties. The naturalness of these properties and relations between *SYM* and *INT* on the one hand and between *CONTR* and *CO - INT* on the other hand established above are sufficient reasons, I believe, to study them. Proportional quantifiers are on the border-line because they involve two different cognitive competences: surely a linguistic competence and, in addition, an elementarily arithmetic competence which is probably not a linguistic one.

References

1. Barwise, J. and Cooper, R. (1981) Generalized quantifiers and natural language *Linguistics and Philosophy* 4, 159-219
2. Beghelli, F. (1992) Comparative Quantifiers, in Dekker, P. and Stokhof, M. (eds.) *Proc. of the VIII Amsterdam Colloquium*
3. Beghelli, F. (1994) Structured Quantifiers, in Kanazawa, M. and Piñon, Ch. (eds.) *Dynamics, Polarity, and Quantification*, CSLI Publications, 119-145
4. Gutiérrez-Rexach, J. (1997) Questions and Generalized Quantifiers, in A. Szabolcsi (ed.) *Ways of Scope Taking*, Kluwer, pp. 409-452

5. Keenan, E. L. (1993) Natural Language, Sortal Reducibility and Generalised Quantifiers, *Journal of Symbolic logic* 58:1, pp. 314-325.
6. Keenan, E.L. (2002) Some Properties of Natural Language Quantifiers: Generalized Quantifier Theory, *Linguistics and Philosophy* 25:5-6, pp.627-654
7. Keenan, E. L. (2003) The Definiteness Effect: Semantics or Pragmatics, *Natural Language Semantics*, 3:2, pp.187-216
8. Keenan, E. L. (in press) Excursions in Natural Logic, forthcoming in Casadio, C. et al. (eds.) *Language and Grammar: Studies in Mathematical Linguistics and Natural Language*, CSLI, pp.3-24
9. Keenan, E. L. and Faltz, L. M. (1985) *Boolean Semantics for Natural Language*, D. Reidel Publishing Company, Dordrecht.
10. Keenan, E. L. and Moss, L. (1985) Generalized quantifiers and the expressive power of natural language, in J. van Benthem and A. ter Meulen (eds.) *Generalized Quantifiers*, Foris, Dordrecht, pp.73-124
11. Keenan, E. L. and Stavi, J. (1986) A semantic characterisation of natural language determiners, *Linguistics and Philosophy* 9, pp.253-326
12. Keenan, E. L. and Westerstahl, D. (1997) Generalized Quantifiers in Linguistics and Logic, in van Benthem, J. and ter Meulen, A. (eds.) *Handbook of Logic and Language*, Elsevier, pp. 837-893.
13. Mostowski, A. (1957) On a generalisation of quantifiers, *Fundamenta Mathematicae* 44, pp.12-36
14. Thijsse, E. (1984) Counting Quantifiers, in J. van Benthem and A. ter Meulen (eds.) *Generalized Quantifiers*, Foris, Dordrecht, pp.127-146
15. Väänänen, J. and Westerstahl, D. (2002) On the Expressive Power of Monotone Natural Language Quantifiers over Finite Models, *Journal of Philos. Logic* 31:4, pp. 327-358
16. Zuber, R. (1998) On the Semantics of Exclusion and Inclusion Phrases, in Lawson, A. (ed.) *SALT8*, Cornell University Press, pp. 267-283
17. Zuber, R. (2000) On Inclusive Questions, in Billerey et al. (eds.) *WCCFL 19 Proceedings*, Cascadilla Press, pp.617-630
18. Zuber, R. (2004a) A class of non-conservative determiners in Polish, *Linguisticae Investigationes* 27:1, pp. 147-165
19. Zuber, R. (2004b) Some remarks on syncategorematicity, in L. Hunyadi et al.: *The Eight Symposium on Logic and Language: Preliminary Papers*, Debrecen, pp. 165-174
20. Zuber, R. (unpublished) Une caractérisation algébrique des quantificateurs logiques

Author Index

- Béchet, Denis 1, 18
Bonato, Roberto 35
Burke, David A. 51

Crabbé, Benoît 84

da S. Corrêa, Marcelo 67
Dikovsky, Alexander 18

Foret, Annie 1, 18
Francez, Nissim 101

Gaiffe, Bertrand 287
Gardent, Claire 131
Gärtner, Hans-Martin 114

Haddad, Bassam 147
Haeusler, E. Hermann 67
Hale, John T. 162

Johannisson, Kristofer 51

Kanazawa, Makoto 330
Kühnberger, Kai-Uwe 255

Lecomte, Alain 205
Ljunglöf, Peter 177

Michaelis, Jens 114
Moreau, Erwan 189

Nait Abdallah, Areski 205
Niehren, Joachim 221

Parmentier, Yannick 131
Preller, Anne 238

Reuer, Veit 255

Sagot, Benoît 271
Seddah, Djamé 287
Stabler, Edward P. 162

Tellier, Isabelle 301
Third, Allan 317

Villaret, Mateu 221

Yaseen, Mustafa 147
Yoshinaka, Ryo 330

Zuber, Richard 347