Glyn Morrill
Mark-Jan Nederhof (Eds.)

LNCS 8036

Formal Grammar

17th and 18th International Conferences
FG 2012 Opole, Poland, August 2012, Revised Selected Papers
FG 2013 Düsseldorf, Germany, August 2013, Proceedings

Springer

Lecture Notes in Computer Science 8036

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

FoLLI Publications on Logic, Language and Information

Subline of Lectures Notes in Computer Science

Glyn Morrill Mark-Jan Nederhof (Eds.)

Formal Grammar

17th and 18th International Conferences, FG 2012/2013
Opole, Poland, August 2012, Revised Selected Papers
Düsseldorf, Germany, August 2013, Proceedings

Springer

Volume Editors

Glyn Morrill
Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Jordi Girona Salgado 1-3
08034 Barcelona, Spain
E-mail: morrill@lsi.upc.edu

Mark-Jan Nederhof
University of St Andrews
School of Computer Science
North Haugh
St. Andrews KY16 9SX, UK
E-mail: markjan.nederhof@googlemail.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-39997-8 e-ISBN 978-3-642-39998-5
DOI 10.1007/978-3-642-39998-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013943908

CR Subject Classification (1998): F.4, I.1, I.2.7

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

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Formal Grammar is a conference dedicated to empirical and theoretical linguistics expressed technically and mathematically. Topics addressed include:

- Formal and computational phonology, morphology, syntax, semantics and pragmatics
- Model-theoretic and proof-theoretic methods in linguistics
- Logical aspects of linguistic structure
- Constraint-based and resource-sensitive approaches to grammar
- Learnability of formal grammar
- Integration of stochastic and symbolic models of grammar
- Foundational, methodological and architectural issues in grammar and linguistics
- Mathematical foundations of statistical approaches to linguistic analysis

Formal Grammar has been celebrated in Barcelona (1995), Prague (1996), Aix-en-Provence (1997), Saarbrücken (1998), Utrecht (1999), Helsinki (2001), Trento (2002), Vienna (2003), Nancy (2004), Edinburgh (2005), Malaga (2006), Dublin (2007), Hamburg (2008), Bordeaux (2009), Copenhagen (2010), Ljubljana (2011), Opole (2012) and Düsseldorf (2013).

The present volume collects the papers from the 17th Conference on Formal Grammar held in Opole in 2012 and the 18th Conference on Formal Grammar held in Düsseldorf in 2013. Together, these two conferences comprise 18 contributed papers selected from 27 high-quality submissions. In addition, the 2012 meeting featured invited talks by Larry Moss and Reinhard Muskens, and the 2013 meeting featured invited talks by Carl Pollard and Chung-chieh Shan, for which we are most grateful. We would also like to thank the local organizers of ESSLLI 2012 and ESSLLI 2013, with which the conferences were colocated.

May 2013

Glyn Morrill
Mark-Jan Nederhof

Organization

Program Committee 2012

Alexander Clark	Royal Holloway University, UK
Berthold Crysmann	CNRS - LLF, France
Alexandre Dikovsky	Université de Nantes, France
Denys Duchier	Université d'Orleans, France
Annie Foret	IRISA - IFSIC, France
Nissim Francez	Technion, Israel
Laura Kallmeyer	University of Düsseldorf, Germany
Makoto Kanazawa	National Institute of Informatics, Japan
Greg Kobele	University of Chicago, USA
Valia Kordoni	Universität des Saarlandes, Saarbrücken, Germany
Stefan Müller	Freie Universität Berlin, Germany
Gerald Penn	University of Toronto, Canada
Christian Retoré	Université Bordeaux 1, France
Manfred Sailer	Goethe University Frankfurt, Germany
Ed Stabler	UCLA, USA
Anders Søgaard	University of Copenhagen, Denmark
Jesse Tseng	CNRS - CLLE-ERSS, France

Standing Committee 2012

Markus Egg (Program Co-chair)	Humboldt-Universität Berlin, Germany
Glyn Morrill (Program Co-chair)	Universitat Politècnica de Catalunya, Spain
Mark-Jan Nederhof (Program Co-chair)	University of St Andrews, UK
Frank Richter (Program Co-chair)	University of Tübingen, Germany

Program Committee 2013

Alexander Clark	King's College London, UK
Benoit Crabbé	Université Paris 7, France
Berthold Crysmann	CNRS - LLF, France
Denys Duchier	Université d'Orleans, France
Annie Foret	IRISA - IFSIC, France

Nissim Francez	Technion, Israel
Laura Kallmeyer	University of Düsseldorf, Germany
Makoto Kanazawa	National Institute of Informatics, Japan
Greg Kobele	University of Chicago, USA
Valia Kordoni	Humboldt-Universität Berlin, Germany
Wolfgang Maier	University of Düsseldorf, Germany
Stefan Müller	Freie Universität Berlin, Germany
Gerald Penn	University of Toronto, Canada
Christian Retoré	Université Bordeaux 1, France
Manfred Sailer	Goethe University Frankfurt, Germany
Anders Søgaard	University of Copenhagen, Denmark
Ed Stabler	UCLA, USA
Jesse Tseng	CNRS - CLLE-ERSS, France

Standing Committee 2013

Glyn Morrill	Universitat Politècnica de Catalunya, Spain
Mark-Jan Nederhof (Program Co-chair)	University of St Andrews, UK
Rainer Osswald	Heinrich-Heine-Universität Düsseldorf, Germany
Frank Richter (Program Co-chair)	University of Tübingen, Germany

Table of Contents

Formal Grammar 2012

Formal Grammar 2013

On IO-Copying and Mildly-Context Sensitive Formalisms

Pierre Bourreau[1], Laura Kallmeyer[2], and Sylvain Salvati[1]

[1] Université Bordeaux 1
351, Cours de la Libération
33405 Talence Cedex, France
{bourreau,salvati}@labri.fr
[2] Heine-Heinrich Universität Düsseldorf
Universitätstr. 1
40225 Düsseldorf, Germany
kallmeyer@phil.uni-duesseldorf.de

Abstract. The class of mildly context-sensitive languages is commonly regarded as sufficiently rich to capture most aspects of the syntax of natural languages. Many formalisms are known to generate families of languages which belong to this class. Among them are tree-adjoining grammars, multiple context-free grammars and abstract categorial grammars. All these formalisms have in common that they are based on operations which do not copy already derived material in the course of a derivation. We propose an extension of the class of languages captured by these formalisms that is arguably mildly context-sensitive. This extension is based on a mild use of a copying operation we call IO-substitution.

Keywords: mildly context-sensitive formalisms, abstract categorial grammars, semilinear and constant-growth languages, IO grammars.

1 Introduction

The question of the amount of expressive power a formalism needs so as to adequately model natural language syntax is still open. Nevertheless the notion of *mildly context sensitive languages* that Joshi [Jos85] proposes plays a structuring role in this debate. A class **L** of languages is said to be *mildly context-sensitive* [Jos85, Wei88] if it satisfies the following properties:

1. **L** contains all context-free languages.
2. if L is in **L** then L describes only some restricted crossing dependencies.
3. every language $L \in \mathbf{L}$ satisfies the constant-growth property.
4. for every language $L \in \mathbf{L}$, recognition can be performed in polynomial-time.

Some well-known formalisms are widely regarded as defining mildly context-sensitive classes of languages, like Linear Context-Free Rewriting Systems [VSWJ87] (LCFRSs), Multiple Context-Free Grammars [SMMK91] (MCFGs), Minimalist Grammars [Sta96] (MGs) and set-local multicomponent Tree Adjoining Grammars [Wei88] (MCTAGs). All those formalisms actually define the same class of languages. Nevertheless, we

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 1–16, 2013.

think it is more appropriate to say that a class of languages is mildly context-sensitive rather than speaking of *the* class of mildly context-sensitive languages. Indeed, property 2 is mostly intuitive and not really well formalized. It therefore induces some subjective judgments about whether a class of languages is mildly context-sensitive. Indeed, while MCFLs are usually regarded as mildly context sensitive, this second property is interpreted in [JSW91] as *perhaps [excluding] the so-called language MIX*, which has recently been shown to be in the class of MCFLs [Sal11].

Property 3 defining a mildly context-sensitive class of languages is also open to debate. For example, Weir [Wei88] considers that *the constant-growth property is too weak to capture [the linguistic] intuition fully [...] the slightly stronger property of semilinearity may come closer*. As it happens, all the formalisms we mention above are semilinear. So as to understand what difference it makes to consider languages that satisfy the constant-growth property rather than the semilinearity property, we explore a way of defining interesting families of languages that are not semilinear but that satisfy the constant-growth property. For this goal, we extend the class of LCFRL/MCFL following ideas from [Kal10] by adding to these formalisms a limited capacity of copying during derivations. This approach differs from that of [Kal10] in that it does not try to find the most general way of achieving this goal. In particular, we try to obtain formalisms whose derivations are similar to those of context-free grammars, and as a consequence we secure the decidability of the emptiness problem for the languages we define.

Our proposal is based on an operation that can be traced back to Fischer [Fis68a] and that we call *IO-substitution*. Roughly speaking, an IO-substitution of a language L_2 in a language L_1 amounts, for every word w of L_2, to replacing in every word of L_1 all the occurrences of a given letter x by w. In such an IO-substitution, one can see the words of L_1 as *parametrized by the letter* x, this parameter being instantiated with words taken from L_2. In general, when recursively applied, such an operation gives rise to languages that may have an exponential growth. Interestingly, we show that under certain simple conditions, a class of languages satisfying the constant-growth property is closed under IO-substitution. Thus taking the closure of classes of semilinear languages, such as MCFLs/LCFRLs, under this operation gives rise to classes that have the constant-growth property but which may not be semilinear anymore.

The structure of the paper is as follows: in the first part of the article, we recall the notions of semilinearity and constant-growth for languages. We then introduce the operation of IO-substitution on languages and show how the constant-growth (*resp.* semilinearity) property can be preserved when applied on constant-growth (*resp.* semilinear) languages. Thanks to this result, we define a new class of languages we call $IO(\mathbf{MCFL})$, as a candidate for the biggest mildly context-sensitive class. In the second part of the article, we characterize a grammatical formalism which exactly captures languages in $IO(\mathbf{MCFL})$. Such grammars are particular abstract categorial grammars of second-order, the generated languages of which are tree languages. The given formalism enjoys the property of being analyzable in polynomial-time, which confirms $IO(\mathbf{MCFL})$ as a candidate for being a mildly context-sensitive class of languages.

2 IO-Substitutions, Semilinearity and Constant-Growth

2.1 Basic Definitions

Given a finite set Σ, we write Σ^* for the set of words built on Σ, and ϵ for the empty word. Given w in Σ^*, we write $|w|$ for its length, and $|w|_a$ for the number of occurrences of a letter a of Σ in w. A language on Σ is a subset of Σ^*. Given two languages $L_1, L_2 \subseteq \Sigma^*$, $L_1 \cdot L_2$, the concatenation of L_1 and L_2, is the language $\{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$; the union of L_1 and L_2 is written $L_1 + L_2$.

We write $\mathbb{N}$ for the set of natural numbers. For a given alphabet Σ, $\mathbb{N}^\Sigma$ is the set of vectors whose coordinates are indexed by the letters of Σ. Given $a \in \Sigma$ and $\overrightarrow{v} \in \mathbb{N}^\Sigma$, $\overrightarrow{v}[a]$ denotes the value of the coordinate a of $\overrightarrow{v}$. We write $\overrightarrow{e_a}$ for the unit vector on the a-dimension, *i.e.* the vector such that $\overrightarrow{e_a}[c] = 0$ when $c \neq a$ and $\overrightarrow{e_a}[a] = 1$.

The constant-growth property expresses some constraint about the distribution of the possible length of the words in a language:

Definition 1. *A language $L \subseteq \Sigma^*$ is said to be* constant-growth *if there exist $k, c \in \mathbb{N}$ such that, given $w \in L$ if $|w| > k$, then there is $w' \in L$ such that $|w| < |w'| \leq |w| + c$.*

A class of languages $\mathbf{L}$ *is* constant-growth *when every L in* $\mathbf{L}$ *is constant-growth.*

Most of the mildly context-sensitive classes of languages commonly used verify the stronger property of semilinearity, which is based on the notion of the Parikh image.

Definition 2. *Let us consider a word w in a language $L \subseteq \Sigma^*$. The* Parikh image of w, *written $\overrightarrow{p}(w)$ is the vector of $\mathbb{N}^\Sigma$ such that, for every $a \in \Sigma$, $\overrightarrow{p}(w)[a] = |w|_a$. The Parikh image of L is defined as $\overrightarrow{p}(L) = \{\overrightarrow{p}(w) \mid w \in L\}$.*

Definition 3. *A set V of vectors of $\mathbb{N}^\Sigma$ is called* linear *when there are vectors $\overrightarrow{x_0}, \ldots, \overrightarrow{x_n}$ in $\mathbb{N}^\Sigma$ such that $V = \{\overrightarrow{x_0} + k_1\overrightarrow{x_1} + \ldots + k_n\overrightarrow{x_n} \mid k_1, \ldots, k_n \in \mathbb{N}\}$. V is called* purely linear *when $\overrightarrow{x_0} = \overrightarrow{0}$.*

A set of vectors is called semilinear *(resp.* purely semilinear*) when it is a finite union of linear (resp. purely linear) sets.*

Given two sets of vectors V_1 and V_2, we will denote by $V_1 + V_2$ the set $\{\overrightarrow{v_1} + \overrightarrow{v_2} \mid \overrightarrow{v_1} \in V_1, \overrightarrow{v_2} \in V_2\}$. Similarly, given $k \in \mathbb{N}$ and a set of vectors V, we will write $kV = \{k\overrightarrow{v} \mid \overrightarrow{v} \in V\}$.

Definition 4. *A language L is called* semilinear *when $\overrightarrow{p}(L)$ is a semilinear set. A class of languages* $\mathbf{L}$ *is* semilinear *when all the languages it contains are semilinear.*

Given two alphabets Σ_1 and Σ_2, a string homomorphism h from Σ_1^* to Σ_2^* is a function such that $h(\epsilon) = \epsilon$ and $h(w_1 w_2) = h(w_1)h(w_2)$. Given $L \subseteq \Sigma_1^*$, we write $h(L)$ for the language $\{h(w) \in \Sigma_2^* \mid w \in L\}$. The homomorphism h is *alphabetic* when for each a in Σ_1, $h(a)$ is in Σ_2.

Definition 5. *Let us consider a class* $\mathbf{L}$ *of languages which share the same alphabet Σ. Given an n-ary operation* $\mathsf{op} : (\Sigma^*)^n \mapsto \Sigma^*$, *where $n \in \mathbb{N}$, we say that* $\mathbf{L}$ *is closed under* op *if for every $L_1, \ldots, L_n \in \mathbf{L}$, $\mathsf{op}(L_1, \ldots, L_n) \in \mathbf{L}$.*

In the rest of the paper, we will be particularly interested in **MCFL**, the class of *multiple context-free languages*. This class of languages is known to be mildly context-sensitive and semilinear. It is also closed by homomorphism, intersection with regular sets, union and concatenation [SMMK91]. Note that our study could also be done on other mildly context-sensitive classes of string languages, such as the classes of context free-languages, or yields of tree-adjoining grammars, or well-nested mildly context-sensitive grammars.

2.2 IO-Substitution and Copies

In this section, we introduce an operation on languages, which we will use to define constant-growth languages. This operation, which we call *IO-substitution*, enables a specific kind of copying mechanism, which we want to use so as to capture some languages that do not belong to **MCFL**, but still satisfy the constant-growth property:

Example 1. The language $L_{a,b} = \{(a^n b^n)^m \mid n,m \in \mathbb{N}\}$ is not a **MCFL** (see [Kal10] for a proof). Nevertheless, it is a semilinear language, as its Parikh image is $\{k(1,1) \mid k \in \mathbb{N}\}$, a semilinear set.

The language $L_{count} = \{(a^n d)^m a^n \mid n,m \in \mathbb{N}\}$ is not a semilinear language, but satisfies the constant-growth property: given a word w in L_{count} such that $|w| \geq 0$ it holds that $|w| < a a^{|w|} \leq |w| + 1$.

The language $L_{exp} = \{a^{2^n} \mid n \in \mathbb{N}\}$ is neither semilinear, nor constant-growth.

Interestingly, these three languages are also known to be generated by IO-macro grammars [Fis68b, Fis68a], in which copying operations are allowed in a specific way, which we will not discuss in detail in this article. We here introduce the notion of IO-substitution that allows us to mimic in a non-recursive way the copying operations that are used in IO-macro grammars. This notion of substitution can be compared with the notion of *language substitution* which corresponds to the non-recursive counterpart of the copying operations in OI-macro grammars (another notion of macro grammars defined by Fischer [Fis68b, Fis68a]). While language substitution consists in replacing each occurrence of a letter a in the words of a language L_1 by some word of a language L_2, IO-substitution requires that every such occurrence of a is replaced by the very same word of L_2.

Definition 6. *Let us consider an alphabet Σ, and two languages $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$. Given a word $w \in L_2$, and a symbol $a \in \Sigma$, we define the homomorphism $h_{a,w} : \Sigma^* \mapsto \Sigma^*$ by $h_{a,w}(c) = c$ for $c \in \Sigma - \{a\}$ and $h_{a,w}(a) = w$.*

The language $L_1[a := L_2]_{IO}$ is then defined as $\bigcup_{w \in L_2} h_{a,w}(L_1)$.

Note that $L_1[a := L_2]_{IO} = L_1$ if for every word $w \in L_1$, a has no occurrence in w.

As an example, we can consider the languages $L_1 = a^*$ and $L_2 = ab + c$; the language $L_1[a := L_2]$ is then defined as $(ab + c)^*$.

Definition 7. *Given a class of languages* **L** *built on an alphabet Σ, we define the class $IO_n(\mathbf{L})$ by induction on $n \in \mathbb{N}$ as*

- $IO_0(\mathbf{L}) = \mathbf{L}$ *and*

– $IO_{n+1}(\mathbf{L}) = \bigcup_{L_1,L_2 \in IO_n(\mathbf{L})} \bigcup_{x \in \Sigma} L_1[x := L_2]_{IO}$, *for every* $n \geq 0$

We define $IO(\mathbf{L})$ *as* $\bigcup_{n \in \mathbb{N}} IO_n(\mathbf{L})$.

Notice that $IO(\mathbf{L})$ is the smallest class of languages that contains $\mathbf{L}$ and that is closed under IO-substitution.

Based on this operation, we can build the languages $L_{a,b}$ and L_{count} easily from context-free languages. Indeed, given $L_1 = \{x^n \mid n \in \mathbb{N}\}$ and $L_2 = \{a^m b^m \mid m \in \mathbb{N}\}$, the language $L_{a,b}$ is the language $L_1[x := L_2]_{IO}$. In the same way, $L_{count} = L'_1[x := L'_2]_{IO}$, where $L'_1 = \{(xd)^m x \mid m \in \mathbb{N}\}$ and $L'_2 = \{a^n \mid n \in \mathbb{N}\}$. On the other hand, there is no trivial way of using the IO-substitution operation on multiple context-free languages to generate L_{exp}. One can also remark that L_1, L_2, L'_1 and L'_2 are context-free languages, hence semilinear languages and they satisfy the constant-growth property. It is therefore natural to investigate the conditions under which semilinearity and the constant-growth property are preserved by the IO-substitution operation.

2.3 Preserving the Semilinearity and Constant-Growth Properties

In the previous examples, we have seen that the newly introduced operation of IO-substitution can lead in some particular cases to the construction of semilinear or constant-growth languages when applied to semilinear languages. In this section, we investigate the conditions which lead to the preservation of these properties.

Definition 8. *Given an alphabet* Σ *and* $a \in \Sigma$*:*

– *a set of vectors* $V \subseteq \mathbb{N}^\Sigma$ *is* a-independent *when for every* $\overrightarrow{v} \in V$, $\overrightarrow{v}[a] = 0$.
– *a set of vectors* V *is* a-isolating *if it is of the form* $U + c\{k\overrightarrow{e_a} \mid k \in \mathbb{N}\}$ *where* U *is* a*-independent, and* $c \in \mathbb{N} - \{0\}$.
– *a semilinear set is* a*-isolating when it is a finite union of linear* a*-isolating sets of vectors.*
– *a semilinear language is* a*-isolating when its Parikh image is a semilinear* a*-isolating set of vectors.*

Example 2. Let us consider various languages so as to illustrate a-isolating languages:

– the language $\{x^n \mid n \in \mathbb{N}\}$ is obviously x-isolating, as its Parikh image is $\{n(1) \mid n \in \mathbb{N}\}$.
– the language $abxa^*x^*$ is not x-isolating as its Parikh image is $\{(1,1,1) + n(1,0,0) + m(0,0,1) \mid n,m \in \mathbb{N}\}$ (with $\overrightarrow{e_a} = (1,0,0)$, $\overrightarrow{e_b} = (0,1,0)$ and $\overrightarrow{e_x} = (0,0,1)$).
– the language aba^*x^* is x-isolating, since its Parikh image is $\{(1,1,0) + n(1,0,0) + m(0,0,1) \mid n,m \in \mathbb{N}\} = \{(1,1,0) + n(1,0,0) \mid n \in \mathbb{N}\} + 1\{n(0,0,1) \mid n \in \mathbb{N}\}$.
– the language $(xa)^*x$ is not x-isolating as its Parikh image is $\{(0,1) + n(1,1) \mid n \in \mathbb{N}\}$ (with $\overrightarrow{e_a} = (1,0)$, and $\overrightarrow{e_x} = (0,1)$).

One can remark that, given a word w in a a-isolating language, the number of occurrences of a in w is completely independent from the number of occurrences of any other symbol in w.

We now show that the IO-substitution of a letter a of a semilinear language in a a-isolating language generates a semilinear language.

Lemma 1. *Given $c \in \mathbb{N}$, if V is a purely semilinear set then the set $\{k\overrightarrow{c}\overrightarrow{v} \mid \overrightarrow{v} \in V \wedge k \in \mathbb{N}\}$ is equal to cV.*

Proof. To prove the Lemma, it suffices to prove that when V is a purely linear set of vectors, then $\{k\overrightarrow{v} \mid \overrightarrow{v} \in V \wedge k \in \mathbb{N}\}$ is equal to cV. If V is purely linear, let $\overrightarrow{x_1}, \ldots, \overrightarrow{x_n}$ be the vectors such that $V = \{\sum_{i=1}^{n} k_i\overrightarrow{x_i} \mid k_1, \ldots, k_n \in \mathbb{N}\}$. We obviously have $cV \subseteq \{kc\overrightarrow{v} \mid \overrightarrow{v} \in V \wedge k \in \mathbb{N}\}$, by considering $k = 1$. Moreover given $\overrightarrow{u}$ in $\{kc\overrightarrow{v} \mid \overrightarrow{v} \in V \wedge k \in \mathbb{N}\}$, we must have $\overrightarrow{u} = kc\overrightarrow{v}$ with $\overrightarrow{v}$ in V, so that $\overrightarrow{u} = kc\sum_{i=1}^{n} k_i\overrightarrow{x_i} = c\sum_{i=1}^{n} kk_i\overrightarrow{x_i}$ and is thus also in cV.

Theorem 1. *Given a semilinear language $L_1 \subseteq \Sigma^*$, which is a-isolating, and a purely semilinear language L_2, the language $L_1[a := L_2]_{IO}$ is semilinear.*

Proof. Since L_1 is semilinear and a-isolating, its Parikh image is a finite union of linear a-isolating sets $V_1, \ldots, V_n$. It is then easy to show that the Parikh image of $L_1[a := L_2]_{IO}$ is the finite union of $V'_1, \ldots, V'_n$ where, for every $1 \leq i \leq n$, $V'_i = U_i + \{kc_i\overrightarrow{u} \mid k \in \mathbb{N}, \overrightarrow{u} \in \overrightarrow{p}(L_2)\}$ (where $c_i \in \mathbb{N}$ and U_i is a-independent). Since $\overrightarrow{p}(L_2)$ is purely semilinear, by Lemma 1, $\{kc_i\overrightarrow{u} \mid k \in \mathbb{N}, \overrightarrow{u} \in \overrightarrow{p}(L_2)\} = c_i\overrightarrow{p}(L_2)$ which is purely semilinear. Thus, V'_i is the sum of two semilinear sets and is semilinear. Therefore $\bigcup_{i=1}^{n} V'_i$ is a semilinear set. As a conclusion, $L_1[a := L_2]_{IO}$ is semilinear.

While the semilinearity property is preserved by IO-substitution under some rather restrictive constraints, we show that for certain classes of languages, some of their properties are also satisfied by their closure under IO-substitution. Indeed, it is also easy to see that $L_1[x := L_2]_{IO}$, where L_1 and L_2 are semilinear languages, is of constant-growth while not necessarily semilinear (as shown with the languages in Example 2). Interestingly, the constant-growth property can be verified by closure under IO-substitution on classes of languages which are closed under homomorphism.

Lemma 2. *Given $L_1, L_2 \subseteq \Sigma^*$, if L_1 is an infinite language that is of constant-growth and $L_1 \subseteq L_2$, then L_2 is of constant-growth.*

Proof. Let k and c be as in Definition 1 for L_1. Since L_1 is infinite there are some words w in L_1 such that $|w| > k$. We let w_0 be a shortest word so that $|w_0| > k$. Without loss of generality we may assume that $|w_0| = k + 1$. Then using the fact that L_1 is constant-growth we construct a sequence $(w_i)_{i\in\mathbb{N}}$ such that $|w_i| < |w_{i+1}| \leq |w_i| + c$. Now, given a word w in L_2, if $|w| > k$ we let n be the integer such that $|w_n| \leq |w| < |w_{n+1}|$. By definition of $(w_i)_{i\in\mathbb{N}}$ we have $|w_{n+1}| \leq |w_n| + c$ so that $|w_{n+1}| \leq |w| + c$. Therefore $|w| < |w_{n+1}| \leq |w| + c$, and L_2 is constant-growth.

Lemma 3. *If a class of languages* **L** *is closed under (alphabetic) homomorphism, then $IO_n(\mathbf{L})$ is closed under (alphabetic) homomorphism, for every $n \in \mathbb{N}$.*

Proof. We proceed by induction on n. For $n = 0$, $IO_n(\mathbf{L})$ is closed under (alphabetic) homomorphism by hypothesis. For $n = k + 1$, let us consider an alphabet Σ, two languages $L_1, L_2 \subseteq \Sigma^*$ in $IO_k(\mathbf{L})$, and a (alphabetic) homomorphism $h : \Sigma^* \mapsto \Sigma^*$. Given a symbol $a \in \Sigma$, we define the (alphabetic) homomorphism $g : \Sigma^* \mapsto \Sigma^*$ such that $g(a) = a$ and $g(b) = h(b)$ when $b \in \Sigma - \{a\}$. It is then easy to check that $h(L_1[a := L_2]_{IO}) = g(L_1)[a := h(L_2)]_{IO}$. But, by the induction hypothesis, both $g(L_1)$ and $h(L_2)$ are in $IO_k(\mathbf{L})$ which proves that $h(L_1[a := L_2]_{IO})$ is in $IO_n(\mathbf{L})$.

We are now in a position to prove the following theorem:

Theorem 2. *If a class of languages* $\mathbf{L}$ *is constant-growth and closed under homomorphism, then* $IO_n(\mathbf{L})$ *is constant-growth, for every* $n \in \mathbb{N}$.

Proof. We consider the alphabet Σ on which the languages of $\mathbf{L}$ are built. We show by induction on n that the class $IO_n(\mathbf{L})$ is constant-growth. From Lemma 3, we have that $IO_n(\mathbf{L})$ is closed under homomorphism.

The case $n = 0$ is immediate. For $n = k+1$, given L_1 and L_2 in $IO_k(\mathbf{L})$, we consider several cases. Let us first suppose that L_1 is infinite; then for w in L_2 and $a \in \Sigma$, we define $h_{a,w}$ to be the homomorphism such that $h_{a,w}(a) = w$ and $h_{a,w}(b) = b$ if $b \in \Sigma - \{a\}$. By definition $L_1[a := L_2]_{IO} = \bigcup_{w \in L_2} h_{a,w}(L_1)$. The cases where $L_2 = \emptyset$ or $L_2 = \{\epsilon\}$ are trivial. So let us suppose that there is $w \neq \epsilon$ in L_2, then $h_{a,w}(L_1)$ is infinite and, by induction hypothesis and because $IO_k(\mathbf{L})$ is closed under homomorphism, $h_{a,w}(L_1)$ is constant-growth. As $h_{a,w}(L_1) \subseteq L_1[a := L_2]_{IO}$, Lemma 2 shows that $L_1[a := L_2]_{IO}$ is constant-growth.

Let us now suppose that L_1 is finite. The cases where, for every w in L_1, $|w|_a = 0$ or where L_2 is finite are trivial. So let us consider w in L_1 such that $|w|_a > 0$ and L_2 infinite. Then $\{w\}[a := L_2]_{IO}$ is infinite and we are going to show that $\{w\}[a := L_2]_{IO}$ is constant growth. Let k and c be integers as in Definition 1 for L_2. We consider $l = k|w|_a + |w| - |w|_a$ (*i.e.* the length of w when weighting an occurrence of a by k, and an occurrence of any other letter of Σ by 1), $d = c|w|_a$, and a word $u \in \{w\}[a := L_2]_{IO}$ such that $|u| > l$. This implies that there is v in L_2 such that $u = h_{a,v}(w)$, and which satifies $|u| = |v||w|_a + |w| - |w|_a$. Moreover, $|u| > l$ implies $|v| > k$; but then, because L_2 is constant-growth, there is v' in L_2 such that $|v| < |v'| \leq |v| + c$. Let u' be $h_{a,v'}(w)$; then $|u'| = |v'||w|_a + |w| - |w|_a$ so that $|u| < |u'| \leq |u| + d$. This finally shows that $\{w\}[a := L_2]_{IO}$ is constant-growth. As $\{w\}[a := L_2]_{IO}$ is infinite and $\{w\}[a := L_2]_{IO} \subseteq L_1[a := L_2]_{IO}$, Lemma 2 implies that $L_1[a := L_2]_{IO}$ is constant-growth.

The string languages generated by multiple-context free-grammars, or second-order ACGs, for example, are known to be closed under homomorphism [SMMK91, Kan06], and satisfy the semilinearity property. As a consequence, the closure of these languages by IO-substitution forms classes of languages which are constant-growth.

As a remark, the following example illustrates the need of the condition of closure by homomorphism for the class of languages we close by IO-substitution:

Example 3. The language $L_1 = \{a^n b^{2^m} \mid n, m \in \mathbb{N}\}$ is a constant-growth language (because a^* is constant-growth), but is not a MCFL. The language $L_1[a := \{\epsilon\}]_{IO} = \{b^{2^m} \mid m \in \mathbb{N}\}$ is not constant-growth.

We know prove other desirable closure properties of classes of languages which are preserved by IO-substitution.

Lemma 4. *Given a class of languages* $\mathbf{L}$ *that is closed under alphabetic homomorphism, the following holds:*

1. *if* $\mathbf{L}$ *is closed under union, then so is* $IO(\mathbf{L})$*,*
2. *if* $\mathbf{L}$ *is closed under concatenation, then so is* $IO(\mathbf{L})$*,*

Proof. Lemma 3 shows that $IO_n(\mathbf{L})$ is closed under alphabetic homomorphism.

By induction on n, we show that if L_1 and L_2 are in $IO_n(\mathbf{L})$ then $L_1 \cup L_2$ (*resp.* $L_1 \cdot L_2$) is in $IO(\mathbf{L})$. The base case is given by the fact that $\mathbf{L}$ is closed under union (*resp.* concatenation). So let us suppose that $n = k+1$, given L_1, L_2, L'_1 and L'_2 in $IO_k(\mathbf{L})$ we show that $L_1[a := L_2]_{IO} \cup L'_1[b := L'_2]_{IO}$ (*resp.* $L_1[a := L_2]_{IO} \cdot L'_1[b := L'_2]_{IO}$) is in $IO_n(\mathbf{L})$. We suppose that $L_1 \subseteq \Sigma_1^*$ and $L'_1 \subseteq {\Sigma'}_1^*$. We let a' and b' be two distinct letters so that both a' and b' are not in $\Sigma_1 \cup \Sigma'_1$ and we let h be the alphabetic homomorphism from Σ_1 to $\Sigma_2 = (\Sigma_1 - \{a\}) \cup \{a'\}$ such that $h(a) = a'$ and $h(c) = c$ when $c \neq a$; similarly we define the alphabetic homomorphism g from Σ'_1 to $\Sigma'_2 = (\Sigma_1 - \{b\}) \cup \{b'\}$ such that $h(b) = b'$ and $h(c) = c$ when $c \neq b$. Then it is easy to check that $h(L_1)[a' := L_2]_{IO} = L_1[a := L_2]_{IO}$ and $g(L'_1)[b' := L'_2]_{IO} = L'_1[b := L'_2]_{IO}$ and moreover, with the induction hypothesis, we get that $L_1[a := L_2]_{IO} \cup L'_1[b := L'_2]_{IO} = ((h(L_1) \cup g(L'_1))[a' := L_2]_{IO})[b' := L'_2]_{IO}$ (*resp.* $L_1[a := L_2]_{IO} \cdot L'_1[b := L'_2]_{IO} = ((h(L_1) \cdot g(L'_1))[a' := L_2]_{IO})[b' := L'_2]_{IO}$) is in $IO_n(\mathbf{L})$.

Lemma 5. *If* $\mathbf{L}$ *is a class of languages that is closed under union, alphabetic homomorphism and intersection with regular sets, then* $IO(\mathbf{L})$ *is closed under intersection with regular sets.*

Proof. The proof of Lemma 3 entails that $IO_n(\mathbf{L})$ is closed under alphabetic homomorphism. We prove the theorem by induction on n. Let $L_1, L_2 \subseteq \Sigma^*$ be in $IO_n(\mathbf{L})$ and R be a regular set whose syntactic monoid is $\mathbb{M} = (\mathcal{M}, \cdot, 1)$ and is recognized by $\mathcal{N} \subseteq \mathcal{M}$ using the monoid homomorphism φ. We are going to show that $L_1[a := L_2]_{IO} \cap R$ is in $IO(\mathbf{L})$. For every m in $\mathcal{M}$, we define R_m to be the regular language recognized by $\{m\}$ using φ and we define the monoid homomorphism φ_m from Σ^* to $\mathbb{M}$ so that $\varphi_m(a) = m$ and for every c in Σ, $c \neq a$ implies $\varphi_m(c) = \varphi(c)$. We then let Q_m be the regular language recognized by $\mathcal{N}$ with the monoid homomorphism φ_m. For each m in $\mathcal{M}$, we let $L_m = (L_1 \cap Q_m)[a := L_2 \cap R_m]_{IO}$, it is then easy to see that $L_1[a := L_2]_{IO} \cap R = \bigcup_{m \in \mathcal{M}} L_m$. But by induction hypothesis, for every $m \in \mathcal{M}$, $L_1 \cap Q_m$ and $L_2 \cap R_m$ are in $IO(\mathbf{L})$ so that L_m is in $IO(\mathbf{L})$. From Lemma 4, since $\mathbf{L}$ is closed under union and under alphabetic homomorphism, $IO(\mathbf{L})$ is closed under union. As R is regular, $\mathcal{M}$ is finite, and $\bigcup_{m \in \mathcal{M}} L_m$ (*e.g.* $L_1[a := L_2]_{IO}$) is in $IO(\mathbf{L})$.

As a consequence of the previous Theorems and of the fact that **MCFL** is closed under union, concatenation, homomorphism and intersection with regular sets [SMMK91], we have the following corollary:

Corollary 1. *The class of languages* $IO(\boldsymbol{MCFL})$ *satisfies the following properties:*

1. *it is constant-growth,*
2. *it is closed under homomorphism, union, concatenation and intersection with regular sets.*

In the following section, we seek a grammatical formalism that captures the class $IO(\mathbf{MCFL})$. Additionally, we show that the membership problem in $IO(\mathbf{MCFL})$ can be solved in polynomial-time. This result shows that $IO(\mathbf{MCFL})$ can arguably be considered as a mildly context-sensitive class of languages bigger than **MCFL**.

3 IO-Multiple Context-Free Languages

We have introduced in the previous section the notions of IO-substitution, and of IO-closure of a class of languages. We have showed that $IO(\mathbf{MCFL})$ is of constant-growth. We are now going to give a grammatical formalism that precisely describes $IO(\mathbf{MCFL})$. This will allow us to see that languages in $IO(\mathbf{MCFL})$ have a polynomial (more precisely **LOGCFL**) membership problem. In order to do so, we are going to encode an IO(MCFL) as a second order Abstract Categorial Grammars (ACG) ([dG01, Mus01]).

3.1 Abstract Categorial Grammars

Given a set of atomic types A, the set of simple types $\mathcal{T}(A)$ on A is the smallest set containing A and such that $(\alpha \to \beta)$ is in $\mathcal{T}(A)$ whenever α and β are in $\mathcal{T}(A)$. We take the usual conventions of writing $\alpha_1 \to \cdots \to \alpha_n \to \beta$ instead of $(\alpha_1 \to (\cdots \to (\alpha_n \to \beta)\cdots))$. The order $\mathtt{ord}(\alpha)$ of a type $\alpha \in \mathcal{T}(A)$ is defined as $\mathtt{ord}(\alpha) = 1$ when α is atomic and $\mathtt{ord}(\alpha) = max(1 + \mathtt{ord}(\alpha_1), \mathtt{ord}(\alpha_2))$ when $\alpha = \alpha_1 \to \alpha_2$.

Definition 9. *A* higher-order signature *(HOS)* $\Sigma = (A, C, \tau)$ *is a tuple made of a finite set of atomic types* A*, a finite set of constants* C *and a function* τ *which associates types in* $\mathcal{T}(A)$ *to constants in* C*.*

The order of a higher-order signature $\Sigma = (A, C, \tau)$ is defined as $max_{c \in C}(\mathtt{ord}(\tau(c)))$. Moreover, we say that a HOS $\Sigma = (A, C, \tau)$ is a *tree signature* (*resp.* a *string signature*) if $\mathtt{ord}(\Sigma) = 2$ (*resp.* $\tau(c)$ is of the form $o \to o$ for every $c \in C$).

Given a HOS $\Sigma = (A, C, \tau)$, we define $(\Lambda^{\alpha}_{\Sigma})_{\alpha \in \mathcal{T}(A)}$ as the family of the smallest sets verifying:

1. a variable x^{α} belongs to $\Lambda^{\alpha}_{\Sigma}$
2. a constant $c \in C$ is in $\Lambda^{\tau(c)}_{\Sigma}$
3. if $M \in \Lambda^{\beta}_{\Sigma}$, then $\lambda x^{\alpha}.M \in \Lambda^{\alpha \to \beta}_{\Sigma}$,
4. if $M_1 \in \Lambda^{\beta \to \alpha}_{\Sigma}$, $M_2 \in \Lambda^{\beta}_{\Sigma}$, then $(M_1 M_2) \in \Lambda^{\alpha}_{\Sigma}$.

The terms are typed *a la Church* so that variables are explicitly carrying their types as exponents, but for the sake of readability, we will often omit this typing notation when it is unnecessary to the understanding.

As usual, we write $M_0 M_1 \dots M_n$ instead of $(\dots(M_0 M_1)\dots M_n)$ and $\lambda x_1 \dots x_n.M$ for $\lambda x_1 \dots . \lambda x_n.M$. We take for granted the notions of β-reduction ($\to^*_{\beta}$) and of the normal form $|M|_{\beta}$ of a term M.

Note that given a tree signature Σ, the construction of a term $M \in \Lambda^{o}_{\Sigma}$ does not make use of the rule 3. Its general form is $xM_1 \dots M_n$ which can be interpreted as a tree $x(t_1, \dots, t_n)$ (t_i being the interpretation of M_i as a tree, for every $i \in \{1, \dots, n\}$). Similarly, if Σ is a string signature, a term $M \in \Lambda^{o \to o}_{\Sigma}$ is of the general form $\lambda x.a_1(\dots(a_n x))$, which can be interpreted as a string $a_1 \dots a_n$.

Example 4. Let us consider a tree HOS $(\{o\}, \{\wedge, \vee, 0, 1\}, \tau)$ where $\tau(\wedge) = \tau(\vee) = o \to o \to o$ and $\tau(0) = \tau(1) = o$. The terms in this signature are boolean formulas made of the connectors $\wedge$ and $\vee$, on the values 0 and 1.

The set $FV(M)$ of free variables of a term M is defined as usual. Moreover, a term M is called closed if it contains no free variable.

Definition 10. *Given a HOS $\Sigma = (A, C, \tau)$, a λ-term M is called* linear *if:*

1. *$M = x$ is a variable or $M = c \in C$, or*
2. *$M = \lambda x.N$, $x \in FV(N)$ and N is linear, or*
3. *$M = M_1 M_2$ if M_1 and M_2 are linear and $FV(M_1) \cap FV(M_2) = \emptyset$.*

The term M is called almost affine *if*

1'. *$M = x$ is a variable or $M = c \in C$, or*
2'. *$M = \lambda x.N$ and N is almost affine, or*
3'. *$M = M_1 M_2$ if M_1 and M_2 are almost affine and if $x^\alpha \in FV(M_1) \cap FV(M_2)$, then $\alpha \in A$.*

Note that given a linear term M and a term M' such that $M \to_\beta^* M'$, M' is also linear.

Given two HOS $\Sigma_1 = (A_1, C_1, \tau_1)$ and $\Sigma_2 = (A_2, C_2, \tau_2)$, we say that Σ_1 and Σ_2 are disjoint if $C_1 \cap C_2 = \emptyset$ and $A_1 \cap A_2 = \emptyset$.

Given a constant c, we also define

- $\Sigma_1 \cup \Sigma_2 = (A_1 \cup A_2, C_1 \cup C_2, \tau(c))$ where Σ_1 and Σ_2 are disjoint and

$$\tau(c) = \begin{cases} \tau_1(c) & \text{if } c \in C_1 \\ \tau_2(c) & \text{if } c \in C_2 \end{cases}$$

- $\Sigma_1 - c = (A_1, C_1 - \{c\}, \tau_1')$, where the domain of τ_1' is $C_1 - \{c\}$, and $\tau_1'(c') = \tau_1(c')$, for every $c' \in C_1 - \{c\}$.

Definition 11. *Given two HOS $\Sigma_1 = (A_1, C_1, \tau_1)$ and $\Sigma_2 = (A_2, C_2, \tau_2)$, a homomorphism $\mathcal{H}$ from Σ_1 to Σ_2 is a function that maps $\mathcal{T}(A_1)$ to $\mathcal{T}(A_2)$, $\Lambda^\alpha_{\Sigma_1}$ to $\Lambda^{\mathcal{H}(\alpha)}_{\Sigma_2}$ for every $\alpha \in \mathcal{T}(A_1)$ and verifies:*

1. *$\mathcal{H}(\alpha \to \beta) = \mathcal{H}(\alpha) \to \mathcal{H}(\beta)$,*
2. *$\mathcal{H}(\lambda x^\alpha.M) = \lambda x^{\mathcal{H}(\alpha)}.\mathcal{H}(M)$, $\mathcal{H}(MN) = \mathcal{H}(M)\mathcal{H}(N)$ and $\mathcal{H}(x^\alpha) = x^{\mathcal{H}(\alpha)}$,*
3. *$\mathcal{H}(c)$ is a closed β-normal λ-term of $\Lambda^{\mathcal{H}(\tau_1(c))}_{\Sigma_2}$.*

Finally, an Abstract Categorial Grammar $G = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$ is a tuple where:

1. $\Sigma_1 = (A_1, C_1, \tau_1)$ and $\Sigma_2 = (A_2, C_2, \tau_2)$ are HOS, respectively called the *abstract* and the *object* signatures of G.
2. $\mathcal{H}$ is a homomorphism from Σ_1 to Σ_2, called *the lexicon*.
3. $s \in A_1$ is *the distinguished type*.

An ACG defines two languages:

- its *abstract language* as $\mathcal{A}(\mathcal{G}) = \{M \in \Lambda^s_{\Sigma_1} \mid FV(M) = \emptyset \text{ and } M \text{ is linear}\}$,
- its *object language* as $\mathcal{O}(\mathcal{G}) = \{M \in \Lambda^{\mathcal{H}(s)}_{\Sigma_2} \mid \exists M' \in \mathcal{A}(\mathcal{G}),\ |\mathcal{H}(M')|_\beta = M\}$.

An ACG is called *linear* (*resp. almost affine*) when for every c in Σ_1, $\mathcal{H}(c)$ is linear (*resp.* almost affine). It is called a *second-order* ACG when Σ_1 is a second-order signature. It is called a *tree-ACG* when Σ_2 is a tree signature and $\mathcal{H}(s)$ is atomic.

It is known that the string languages generated by second-order linear ACGs are precisely **MCFL** ([dGP04] and [Sal06]). In particular the yields of the tree-languages generated by second-order linear ACGs are MCFLs, the yield of a tree being the concatenation of its leaves from left to right. Given a tree language L we write yL for the string language obtained by taking the yields of its trees. In particular given a tree ACG, G, we write $y\mathcal{O}(G)$ for the string language of the yields of the trees in $\mathcal{O}(G)$.

Example 5. Let us consider the tree signatures $\Sigma_1 = (\{s, t_1, t_2\}, \{c, c_1, c_2, c_3, c_4\}, \tau_1)$ and $\Sigma_2(\{o\}, \{a, b, c, d, \epsilon, e, f\}, \tau_2)$ where:

- $\tau_1(c) = t_1 \to t_2 \to s$, $\tau_1(c_1) = t_1 \to t_1$, $\tau_1(c_2) = t_1$, $\tau_2(c_3) = t_2 \to t_2$ and $\tau_1(c_4) = t_2$.
- $\tau_2(a, b, c, d, \epsilon) = o$, $\tau_2(e) = o \to o \to o$ and $\tau_2(f) = o \to o \to o \to o \to o$

We define the second-order tree-ACG $G = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$ such that:

- $\mathcal{H}(t_1) = \mathcal{H}(t_2) = (o \to o \to o) \to o$, $\mathcal{H}(s) = o$
- $\mathcal{H}(c_2) = \mathcal{H}(c_4) = \lambda z.z\epsilon\epsilon$
- $\mathcal{H}(c_1) = \lambda Pz.P(\lambda x_1 x_2.z(eax_1)(ecx_2))$ and $\mathcal{H}(c_3) = \lambda Pz.P(\lambda x_1 x_2.z(ebx_1)(edx_2))$
- $\mathcal{H}(c) = \lambda P_1 P_2.P_1(\lambda x_1 x_3.P_2(\lambda x_2 x_4.f x_1 x_2 x_3 x_4))$

Then, a term is in the abstract language of this ACG if it is of the form

$$c\,\underbrace{(c_1(\ldots(c_1}_{n}\,c_2)\ldots))\,\underbrace{(c_3(\ldots(c_3}_{m}\,c_4)\ldots))$$

for some $n, m \in \mathbb{N}$.

The trees in $\mathcal{O}(G)$ are therefore of the following form:

$$f\,\underbrace{(ea(\ldots(ea\epsilon)\ldots))}_{n}\,\underbrace{(eb(\ldots(eb\epsilon)\ldots))}_{m}\,\underbrace{(ec(\ldots(ec\epsilon)\ldots))}_{n}\,\underbrace{(ed(\ldots(ed\epsilon)\ldots))}_{m}$$

Finally, the yield of this language is the language $\{a^n b^m c^n d^m \mid n, m \in \mathbb{N}\}$, which is known to be a MCFL.

We finish this section by mentioning the following complexity results, which will be used in the next section. We recall that **LOGCFL** is the set of problems which can be reduced in logarithmic space into the problem of recognizability of context-free grammars. This class is known to be a subclass of the problems solvable in polynomial-time (see [Ven87] for more details).

Theorem 3 ([Yos06, Kan07]). *The membership problem of a second-order almost affine ACG is in* ***LOGCFL****.*[1]

3.2 IO-MCFGs as Almost Affine ACGs

As mentioned in the previous section, abstract categorial grammar gives a general framework to speak about **MCFL** as the strings generated by ACGs or, alternatively, as

[1] A polynomial recognizer of second-order almost affine ACGs is given in [BS11]; this algorithm is not known yet to be in **LOGCFL**.

the string languages generated by some tree grammars (which are linear second-order tree-ACGs). In what follows, we seek the characterization of some tree-ACGs which generate tree languages such that their yields form exactly IO-MCFLs. While the construction we give can be applied to string-ACGs, we use tree-ACGs in order to prove the membership problem is **LOGCFL**; indeed the grammars constructed in this way are almost affine tree-ACGs, for which such a complexity result is known [Kan07, Yos06].

Intuitively, given two multiple context-free languages $L_1 \subseteq (\Sigma \cup \{x\})^*$ and $L_2 \subseteq \Sigma^*$, we can first remark that the letter x may have many occurrences in $w_1 \in L_1$. Given a tree-ACG $G_1 = (\Sigma_1, \Sigma_2, \mathcal{H}, s_1)$ such that $y\mathcal{O}(G_1) = L_1$, the idea is to consider x not as a constant in the trees derived by G_1, but as a variable which can be substituted by a word $w_2 \in L_2$ (*i.e.* a tree in a tree-ACG G_2 such that $y\mathcal{O}(G_2) = L_2$). Therefore, we need to build a tree-ACG G_1' such that $t \in \mathcal{O}(G_1)$ iff $\lambda x.t \in \mathcal{O}(G_1')$, and then use a constant $\mathbf{c} : s_1' \to s_2 \to s$ such that $\mathcal{H}(\mathbf{c}) = \lambda x_1 x_2 . x_1 x_2$ to simulate the IO-substitution of x by L_2 into L_1, where s_1' and s_2 are the distinguished types of G_1' and G_2 respectively.

Remark that, because x will now be considered as a variable, the contexts appearing in the left-hand side of a production rule in G_1' will not be linear as in the case of linear ACGs defining MCFLs. But x being a leaf, its type is $\tau(x) = o$, and the contexts will therefore be almost affine, which still ensures the membership problem belongs to **LOGCFL**, hence to the class of problems solvable in polynomial-time.

Definition 12. *Given a second-order ACG $G = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$, where $\Sigma_i = \{A_i, C_i, \tau_i\}$ for $i \in \{1,2\}$, and $x \in C_2$, we define the second-order ACG $\mathsf{abs}(G, x) = (\Sigma_1, \Sigma_2 - \{x\}, \mathcal{H}', s)$ as follows:*

- *given a type $a \in A_1$, $\mathcal{H}'(a) = \tau_2(x) \to \mathcal{H}(a)$*
- *for every $c \in C_1$, given $\tau_1(c) = \alpha_1 \to \ldots \to \alpha_n \to a$, and $\mathcal{H}(c) = \lambda y_1 \ldots y_n . M$, $\mathcal{H}'(c) = \lambda z_1 \ldots z_n x . |\mathcal{H}(c)(z_1\, x) \ldots (z_n\, x)|_\beta$*

Note that this construction transforms the constant x of the object signature into a variable. Moreover, one should remark that given a constant $c \in C_2$, the free occurrences of x in $\mathcal{H}(\mathbf{c})$ are bound by a λ-abstraction in $\mathcal{H}'(\mathbf{c})$. Finally, we should note that $\mathcal{A}(\mathsf{abs}(G)) = \mathcal{A}(G)$.

Example 6. Let us consider the second-order tree-ACG $G_1 = \{\Sigma_1, \Sigma, \mathcal{H}_1, s_1\}$ where $\Sigma_1 = (\{s_1\}, \{c_1, c_2\}, \tau_1)$ where $\tau_1(c_1) = s_1$, $\tau_1(c_2) = s_1 \to s_1$, and $\Sigma = (\{o\}, \{e, x, \epsilon\}, \tau)$, where $\tau(x) = \tau(\epsilon) = o$, $\tau(e) = o \to o \to o$; moreover $\mathcal{H}_1(s_1) = o$ and $\mathcal{H}_1(c_1) = \epsilon$, $\mathcal{H}_1(c_2) = \lambda y.cxy$.

The abstract language of this grammar is made of trees of the form $\underbrace{c_2(\ldots(c_2}_{n} c_1)\ldots)$, where $n \in \mathbb{N}$.

The tree language derived by G_1 is $\{\underbrace{cx(cx(\ldots(cx}_{n} \epsilon)\ldots)) \mid n \in \mathbb{N}\}$.

The grammar $\mathsf{abs}(G_1, x) = \{\Sigma_1, \Sigma_2 - \{x\}, \mathcal{H}_1', s_1\}$ is such that:

$$\mathcal{H}_1'(c_1) = \lambda x.\epsilon$$
$$\mathcal{H}_1'(c_2) = \lambda zx.|(\lambda y.cxy)(zx)|_\beta = \lambda zx.cx(zx)$$

and derives tree contexts of the form $\lambda x.\{\underbrace{cx(cx(\ldots(cx}_{n} \epsilon)\ldots))$, where $n \in \mathbb{N}$.

The transformation given by $\mathtt{abs}$ intuitively results in considering x as a variable, on which substitutions can be applied. This is made explicit in the following theorem.

Lemma 6. *Given a second-order ACG $G = (\Sigma_1, \Sigma_2, \mathcal{H}, s)$, the language $O(\mathtt{abs}(G, x))$ is equal to $\{\lambda x.M \in \Lambda_{\mathcal{H}'(s)}(\Sigma_2) \mid M \in O(G)\}$.*

Proof. G being a second-order ACG, so is G'. Moreover, G and G' share the same abstract language. Let us consider a term $M \in O(G)$. By definition, there exists a term N in $\mathcal{A}(G)$, such that $|\mathcal{H}(N)|_\beta = M$. We prove by induction on N that $|\mathcal{H}'(N)|_\beta = \lambda x.M$. The general form of N is $cN_1 \dots N_p$, for some $p \in \mathbb{N}$. If $p = 0$, then $\tau_1(c) = s$, and by construction, we obtain $\mathcal{H}'(s) = \tau_2(x) \to \mathcal{H}(s)$ and $\mathcal{H}'(c) = \lambda x.|\mathcal{H}(c)|_\beta$. Otherwise, by induction hypothesis, $|\mathcal{H}'(N_i)|_\beta = \lambda x.|\mathcal{H}(N_i)|_\beta = P_i$ such that P_i is a term the type of which is $a_i \in A_1$, as the grammar is a second-order ACG. Then $P = \mathcal{H}'(cN_1 \dots N_p) = \mathcal{H}'(c)P_1 \dots P_p$. Given the general form $\lambda y_1 \dots y_n x.\mathcal{H}(c)(y_1 x) \dots (y_n x)$ of $\mathcal{H}'(c)$, we obtain

$$\begin{aligned} P &=_\beta \lambda x.|\mathcal{H}(c)(P_1 x) \dots (P_n x)|_\beta \\ &=_\beta \lambda x.|\mathcal{H}(c)(|\mathcal{H}(c_1)|_\beta \dots |\mathcal{H}(c_n)|_\beta)|_\beta \\ &=_\beta \lambda x.|\mathcal{H}(N)|_\beta \end{aligned}$$

Given two MCFLs L_1 and L_2, we are now in a position to build the ACG which produces the substitution of x by L_2 in L_1. This operation is simulated by application in the lambda-calculus. Indeed, it suffices to consider the tree-ACGs G_1 and G_2 which have languages L_1 and L_2 as respective yields. We then build $\mathtt{abs}(G_1, x)$ and add a constant the image of which is $\lambda x_1 x_2.x_1 x_2$. Because the IO-substitution substitutes every occurrence of a symbol x by a word, we must constrain the corresponding x in the ACG to be of type o.

First, let us consider that two abstract categorial grammars $G_1 = (\Sigma_{11}, \Sigma_{12}, \mathcal{H}_1, s_1)$ and $G_2 = (\Sigma_{21}, \Sigma_{22}, \mathcal{H}_2, s_2)$ are disjoint if Σ_{11} and Σ_{21} are disjoint.

Definition 13. *A tree-ACG G is called a* tree-IO(ACG) *if*

1. *G is a linear tree-ACG or*
2. *there exist two disjoint tree-IO(ACGs) $G_1 = (\Sigma_1, \Sigma, \mathcal{H}_1, s_1)$ and $G_2 = (\Sigma_2, \Sigma, \mathcal{H}_2, s_2)$, a constant x in the HOS $\Sigma = (A, C, \tau)$ that satisfies $\tau(x) \in A$, such that, given $\mathtt{abs}(G_1, x) = (\Sigma_1, \Sigma - \{x\}, \mathcal{H}', s_1)$, then $G = (\Sigma', \Sigma, \mathcal{H}, s)$ where:*
   (a) $\Sigma' = \Sigma_1 \cup \Sigma_2 \cup \Sigma_s$ where $\Sigma_s = (\{s_1, s_2, s\}, \{c\}, \{c \mapsto s_1 \to s_2 \to s\})$ and, given $\Sigma_i = (A_i, C_i, \tau_i)$ for $i \in \{1, 2\}$, $c \notin C_1 \cup C_2$, $s \notin A_1 \cup A_2$.
   (b) $\mathcal{H}'(e) = \mathcal{H}_i(e)$ if e belongs to the abstract signature of G_i (where $i \in \{1, 2\}$) and $\mathcal{H}'(c) = \lambda x_1 x_2.x_1 x_2$.

We now prove that this formalism exactly captures tree languages the yields of which are IO-MCFLs:

Theorem 4. *A language L is a IO(**MCFL**) iff there exists a tree IO(ACG) such that $L = yO(G)$.*

Proof. If L is in $IO(\mathbf{MCFL})$, then for some n in $\mathbb{N}$, L is in $IO_n(\mathbf{MCFL})$. We prove the Theorem by induction on n. When $n = 0$, L is a **MCFL**, and from [dGP04], there exists a linear tree-ACG G such that $yO(G) = L$, and G is a tree-IO(ACG) by definition. Conversely, for G a linear tree-ACG, $yO(G)$ is a IO-MCFL ([Kan10], [Sal06]).

Now, suppose there exist two languages L_1, L_2 in $IO_n(\mathbf{L})$, such that $L = L_1[x := L_2]_{IO}$. By induction hypothesis, there exist two tree-IO(ACGs) $G_1 = (\Sigma_1, \Sigma, \mathcal{H}_1, s_1)$ and $G_2 = (\Sigma_2, \Sigma, \mathcal{H}_2, s_2)$, which without loss of generality can be assumed to be disjoint, such that $yO(G_1) = L_1$ and $yO(G_2) = L_2$. Moreover, x is a constant of $\Sigma = (A, C, \tau)$, $\tau(x) = o$. According to Lemma 6, $O(\mathtt{abs}(G_1, x)) = \{\lambda x.M \in \Lambda_{\Sigma}^{\mathcal{H}'(s)} \mid M \in O(G_1)\}$. We consider the grammar $G = (\Sigma', \Sigma, \mathcal{H}, s)$ where:

1. $\Sigma' = \Sigma_1 \cup \Sigma_2 \cup \Sigma_s$ where $\Sigma_s = (\{s_1, s_2, s\}, \{c\}, \{c \mapsto s_1 \to s_2 \to s\})$ and $c \notin C_{11} \cup C_{21}$, $s \notin A_{11} \cup A_{21}$.
2. $\mathcal{H}'(e) = \mathcal{H}_i(e)$ if e belongs to the abstract signature of G_i (where $i \in \{1, 2\}$) and $\mathcal{H}'(c) = \lambda x_1 x_2 . x_1 x_2$.

A term M is recognized by this grammar iff there exist $M_1 \in O(\mathtt{abs}(G_1, x))$ and $M_2 \in O(G_2)$ such that $M = |M_1 M_2|\beta$ which is the result of substituting every occurrence of x in M_1 by M_2. Finally, we can conclude $yO(G) = \{h_{x,w_2}(w_1) \mid w_1 \in L_1, w_2 \in L_2\}$. Conversely, we prove that the yield of the language of a tree-IO(ACG) is an IO-MCFL, with similar arguments.

Example 7. Let us consider the grammar G_1 given in the previous example, and a grammar $G_2 = \{\Sigma_2, \Sigma', \mathcal{H}_2, s_2\}$, where:

- $\Sigma_2 = (\{s_2\}, \{c_1', c_2'\}, \tau_2)$ and $\tau_2(c_1') = s_2$, $\tau_2(c_2') = s_2 \to s_2$
- $\Sigma' = (\{o\}, \{a, b, d\}, \tau')$ such that $\tau'(a) = \tau'(b) = o$ and $\tau'(d) = o \to o \to o \to o$
- $\mathcal{H}_2(s_2) = o$, and $\mathcal{H}_2(c_1') = \epsilon$, $\mathcal{H}(c_2') = \lambda y.dayb$

The yield of $O(G_2)$ is $\{a^n b^n \mid n \in \mathbb{N}\}$.

By considering the grammar $abs(G_1)$ in Example 6, we build the grammar $G_{a,b} = \{\Sigma_1 \cup \Sigma_2 \cup \Sigma_s, \Sigma \cup \Sigma' - \{x\}, \mathcal{F}, s)$ where $\Sigma_s = (\{s\}, \{c\}, \{c \mapsto s_1 \to s_2 \to s\})$ and $\mathcal{F}(e) = \mathcal{H}_1'(e)$ (*resp.* $\mathcal{F}(e) = \mathcal{H}_2(e)$) if e is a constant in Σ_1 (*resp.* in Σ_2), and $\mathcal{F}(c) = \lambda x_1 x_2 . x_1 x_2$. This grammar is a tree-IO(ACG). Moreover, the yield of the tree language generated by this grammar is $L_{a,b} = \{(a^n b^n)^m \mid n, m \in \mathbb{N}\}$.

Corollary 2. *The membership problem of an IO(**MCFL**) is* **LOGCFL**.

Proof. This is direct consequence of Theorem 3, as we build almost affine ACGs which capture $IO(\mathbf{MCFL})$.

4 Conclusion

In this paper, we presented the operation of IO-substitution on languages, which introduces some form of copying mechanism. We investigated how the properties of semilinearity and constant-growth can be preserved by this operation, and exhibited a new family of languages, which we call $IO(\mathbf{MCFL})$. This class is not semilinear, but is constant-growth. Moreover, it has a membership problem that is polynomial and it

contains context-free languages. It can be thus considered as mildly context-sensitive. Using abstract categorial grammars, we show how to define actual grammars that capture $IO(\mathbf{MCFL})$. In the meantime we also proved that IO-substitution preserves certain combinations of the closure properties of classes of languages: closure under homomorphism, union, concatenation and intersection with a regular set.

Nevertheless, the class of $IO(\mathbf{MCFL})$ was not proved to be closed under inverse homomorphism, and we conjecture that it is not. Clearly the class of $IO(\mathbf{MCFL})$ is strictly weaker than the class of languages derived by CNL-LMGs in [Kal10], in particular because CNL-LMGs contain languages that are intersections of context-free languages. While $IO(\mathbf{MCFL})$ and CNL-LMGs share some properties, $IO(\mathbf{MCFL})$ and CNL-LMG use different implementations of copying: in $IO(\mathbf{MCFL})$, copying is based on copying certain objects that have already been derived; in CNL-LMGs copying is the result of checking the equality of substrings that are derived independently. In $IO(\mathbf{MCFL})$, the reason why we do not go beyond constant-growth languages is due to a relative independence between the recursive nature of derivations and copying. It is likely that we may find a less syntactic characterization of second-order constant-growth ACG by studying more carefully how copying and recursion may interact. Yet another question consists in giving a characterization of the Parikh images of $IO(\mathbf{MCFL})$ that still enjoy some properties of linearity. Finally, some linguistic examples should be given in order to give arguments of whether the IO-substitution operation is needed to give account of the syntactic structure of languages or not. One could, for instance, use IO-substitution to simulate deletion of material as in gapping, thus:

- L_1 = {Peter likes Mary, Jim likes_g the dog and Paul likes_g the cat} and,
- $L_2 = \{\epsilon\}$

then $L_1[\text{likes\_g} := L_2]_{IO}$ = {Peter likes Mary, Jim the dog, and Paul the cat}.

References

[BS11] Bourreau, P., Salvati, S.: A Datalog recognizer for almost affine λ-CFGs. In: Kanazawa, M., Kornai, A., Kracht, M., Seki, H. (eds.) MOL 12. LNCS, vol. 6878, pp. 21–38. Springer, Heidelberg (2011)

[dG01] de Groote, P.: Towards abstract categorial grammars. In: Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference, pp. 148–155 (2001)

[dGP04] de Groote, P., Pogodalla, S.: On the expressive power of abstract categorial grammars: Representing context-free formalisms. Journal of Logic, Language and Information 13(4), 421–438 (2004)

[Fis68a] Fischer, M.J.: Grammars with macro-like productions. In: IEEE Conference Record of 9th Annual Symposium on Switching and Automata Theory, pp. 131–142. IEEE (1968)

[Fis68b] Fischer, M.J.: Grammars with macro-like productions. PhD thesis, Harvard University (1968)

[Jos85] Joshi, A.K.: Tree-adjoining grammars: How much context-sensitivity is required to provide reasonable strucutral descriptions? In: Natural Language Parsing: Psychological, Computational and Theoretical Perspectives, pp. 206–250 (1985)

[JSW91] Joshi, A.K., Shanker, V.K., Weir, D.J.: The converence of mildly context-sensitive grammar formalisms. In: Sells, P., Shieber, S.M., Wasow, T. (eds.) Foundational Issues in Natural Language Processing, pp. 31–81. The MIT Press (1991)

[Kal10] Kallmeyer, L.: On mildly context-sensitive non-linear rewriting. Research on Language and Computation 8(2), 341–363 (2010)

[Kan06] Kanazawa, M.: Abstract families of abstract categorial grammars. In: Proceedings of WoLLIC, Stanford University CSLI (2006)

[Kan07] Kanazawa, M.: Parsing and generation as Datalog queries. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Prague, pp. 176–183. Association for Computational Linguistics (2007)

[Kan10] Kanazawa, M.: Second-order abstract categorial grammars as hyperedge replacement grammars. Journal of Logic, Language and Information 19(2), 137–161 (2010)

[Mus01] Muskens, R.: Lambda Grammars and the Syntax-Semantics Interface. In: van Rooy, R., Stokhof, M. (eds.) Proceedings of the Thirteenth Amsterdam Colloquium, Amsterdam, pp. 150–155 (2001)

[Sal06] Salvati, S.: Encoding second-order ACGs with deterministic tree walking transducers. In: Proceedings of Formal Grammar, Malaga, Spain (2006)

[Sal11] Salvati, S.: MIX is a 2-MCFL and the word problem in $\mathbb{Z}^2$ is captured by the IO and the OI hierarchies. Technical report, INRIA (2011)

[SMMK91] Seki, H., Matsamura, T., Mamoru, F., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science 88(2), 191–229 (1991)

[Sta96] Stabler, E.P.: Derivational minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)

[Ven87] Venkateswaran, H.: Properties that characterize LOGCFL. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, pp. 141–150. ACM (1987)

[VSWJ87] Vijay-Shanker, K., Weir, D.J., Joshi, A.K.: Characterizing structural descriptions produced by various grammatical formalisms. In: Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics, Stanford (1987)

[Wei88] Weir, D.: Characterizing Mildly Context-Sensitive Grammar Formalisms. PhD thesis, University of Pennsylvania (1988)

[Yos06] Yoshinaka, R.: Linearization of affine abstract categorial grammars. In: Proceedings of the 11th Conference on Formal Grammar, Malaga, Spain, pp. 185–199 (2006)

The Distribution and Interpretation of Hausa Subjunctives: An HPSG Approach

Berthold Crysmann

CNRS — Laboratoire de linguistique formelle (UMR 7110) — Paris-Diderot
crysmann@linguist.jussieu.fr

Abstract. Hausa subjunctive markers have been claimed (Wolff, 1993; Newman, 2000; Jaggar, 2001) to correspond to two functionally distinct paradigms: a true subjunctive, and a "neutral", unspecified for tense, aspect and mood, the latter being used, inter alia, in constructions involving sequences of events. In this paper, I shall demonstrate that the deletion approach advanced by Newman (2000) is not only limited in its empirical scope, but also theoretically questionable. Building on previous work by Schuh (2003), I shall argue instead for a single category and propose a formal treatment in HPSG which crucially builds on Schuh's notion of the subjunctive as a dependent category, modelled as an embedding (qeq) requirement in Minimal Recursion Semantics. Using asymmetric projection of TAM information, together with constructional introduction of content, the present paper derives the distribution of the Hausa subjunctive and its specific interpretation by means of constraint interaction.

1 Data

1.1 The Hausa TAM System

Hausa[1] marks tense, aspect, mood (TAM) categories, as well as polarity (negation) by means of a set of discrete markers syntactically preceding the verb (VP). In contrast to, e.g., Creole languages, these markers cannot be stacked. Furthermore, TAM markers agree with the subject in person, number, and gender (2nd and 3rd singular). While in some paradigms (e.g. future, habitual, continuous, rhetorical), TAM and agreement information can be segmented, other TAM

[1] Hausa is an Afroasiatic language spoken mainly in Northern Nigeria and bordering areas of Niger. Both tone (high, low, falling) and length (long vs. short) are lexically and grammatically distinctive. Following common practise, I shall mark low tone with a grave accent and falling tone with a circumflex. Vowels unmarked for tone are high. Length distinctions are signalled by macron.

I shall make use of the following inventory of morphological tags in the glosses: S = singular, P = plural, M = masculine, F = feminine, IOM = indirect object marker, REL = relativiser, COMP = complementiser, FOC = focus marker, COMPL = completive aspect, CONT = continuous aspect, HAB = habitual, POT = potential, FUT = future, SUBJ = subjunctive, and NEUT.

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 17–31, 2013.

markers are fusional (e.g., completive, subjunctive, potential), using suprasegmental means, like grammatical tone and vowel length, to mark TAM information. Where TAM is marked segmentally, the order of agreement and TAM markers varies according to paradigm: while in the future, exponents of agreement follow those of tense, they precede the TAM marker in the habitual and rhetorical.

Table 1. Hausa TAM markers (excerpt)

	1.SG	2.SG.M	2.SG.F	3.SG.M	3.SG.F	4	1.PL	2.PL	3.PL
Abs. Completive	nā	kā	kin	yā	tā	an	mun	kun	sun
Rel. Completive	na	ka	kikà	ya	ta	akà	mukà	kukà	sukà
Abs. Continuous	'nà	kanà	kinà	yanà	tanà	anà	munà	kunà	sunà
Rel. Continuous	nakè	kakè	kikè	yakè	takè	akè	mukè	kukè	sukè
Potential (Abs.)	nâ	kâ	kyâ	yâ	tâ	â	mâ	kwâ	sâ
Rhetorical (Rel.)	nikà	kakà	kikà	yakà	takà	akà	mukà	kukà	sukà
Habitual	nakàn	kakàn	kikàn	yakàn	takàn	akàn	mukàn	kukàn	sukàn
Future	zân	zā kà	zā kì	zâi	zā tà	zā à	zā mù	zā kù	zā sù
Subjunctive	'n/nà	kà	kì	yà	tà	à	mù	kù	sù

Absolute vs. Relative Paradigms. Certain TAM paradigms, namely the completive, continuous, and potential/rhetorical[2] display an alternation between absolutive and relative paradigms, depending on the presence vs. absence of a wh-filler at the top of the local clause: thus, with wh-extraction, focus fronting, and relativisation, forms of the relative paradigm are used, instead of the general or absolute paradigm employed elsewhere. Choice of relative forms is triggered by the presence of a local filler, not by the presence of a gap (Tuller, 1986; Davis, 1986)[3]:

(1) mè sukè / *sunà fatan sun / *sukà
what 3.P.CONT.REL 3.P.CONT.ABS hoping 3.P.COMPL.ABS 3.P.COMPL.REL
gamà
finish
'What did they hope they have finished?'

As illustrated by the example above, long extraction of the complement of the embedded verb *gamà* 'finish' only features relative marking in the matrix clause, i.e., in the local domain of the wh-filler *mè*, whereas the completive marker in the embedded clause appears in the general or absolute form.

1.2 The Subjunctive TAM

As illustrated in Table 1, the subjunctive TAM is characterised by an all-L short vowel paradigm. The Hausa subjunctive is used to express a variety of

[2] The observation that the potential alternates with the rhetorical in extraction contexts is due to Newman. See Newman (2000) for details.

[3] See Crysmann (2005) for an in-depth discussion of extraction marking in Hausa.

functions. In independent clauses, the subjunctive is used to express commands, prohibitions and wishes. In subordinate contexts, the subjunctive appears as the complement of permissive, volitional, and causal verbs, as well as modal verbs (*kàmātà* 'should', *càncantà* 'deserve', *yìwu* 'be possible') and non-verbal modal predicates, such as *dōlè/tīlàs* 'necessary' or , *gāra* 'better'. Furthermore, subjunctive is also governed by certain subordinating conjunctions, including *don/dȍmin* 'in order to' or *kȁfin* 'before'.

Syntactic Constraints on Subjunctives. Despite the plurifunctionality of the subjunctive marker, there are a couple of syntactic environments in which the use of this marker is apparently illicit: as noted, e.g., by Schuh (2003), subjunctives cannot be used as complements to epistemic verbs such as *sanī* 'know'. Another constraint regarding its syntactic distribution is the ban on subjunctives in clauses involving a filler (Tuller, 1986; Newman, 2000; Jaggar, 2001): thus, wh-fronting, focus fronting, and relative clauses are equally incompatible with subjunctive TAM.

(2) ** mùtumìn dà yà kāwō ruwā
man REL 3.S.M.SUBJ bring water
'the man who should bring water' (Newman, 2000, 537)

Subjunctives are fine, however, in these environments, once they are more deeply embedded (Newman, 2000), e.g., under a modal predicate like *dōlè*.

(3) à cikin dākìn wândà dōlè nē kà yi barcī
at inside hut REL.M.S necessity COP 2.S.M.SUBJ do sleep
'inside the room which you must sleep in' (Newman, 2000, 537)

Given the fact that only direct use of a subjunctive to express necessity or obligation is ruled out in these constructions, but embedding under an explicit modal predicate having the same denotation is fine, I conclude that the constraint under consideration is a syntactic, rather than semantic one. This conclusion appears to be supported by the fact that Hausa TAM markers independently display sensitivity to these contexts (see above).

1.3 The Neutral TAM

Subjunctive form TAM markers also surface in certain constructions without any modal force. Recent grammars of Hausa (Wolff, 1993; Newman, 2000; Jaggar, 2001) unanimously distinguish this form from the subjunctive, calling it the "neutral" aspect, or "Grundaspekt". Newman (2000), e.g., argues that Hausa synchronically features two functionally distinct, though fully homophonous paradigms (cf. Wolff, 1993).

Newman (2000) cites three major constructions in which the "neutral" TAM surfaces, which I shall discuss in turn:

– Non-initial conjuncts in sequences of events in the continuous, future, potential, and habitual

- Main clauses following conditionals
- Initial and non-initial TAM in a sequence of two events receiving a habitual interpretation

Sequences of Events. In sequences of non-completive TAMs, i.e., continuous, future, potential, and habitual, only the first event is marked with a non-completive, non-subjunctive TAM, whereas non-initial conjuncts typically surface in the subjunctive or "neutral".

(4) kullum yanǎ gyārà ɗākìn yà tsabtàcē shi yà
every day 3.S.M.CONT tidy hut 3.S.M.NEUT clean 3.S.M 3.S.M.NEUT
shārȅ
sweep
'Every day he tidies the room and cleans it and sweeps (it).' (Jaggar, 2001, 191)

(5) zā tà karàntà littāfìn tà rubȕtà takàrdā tà kai wà
3.S.F.FUT read book 3.S.F.NEUT write paper 3.S.F.NEUT carry IOM
mālàmintà
teacher.3.S.F.POSS
'She will read the book, write a paper and take (it) to her teacher.' (Jaggar, 2001, 192)

(6) mâ gamà aikìn mù tāshì mù tàfi gidā dà yâmmā
1.P.POT finish work 1.P.NEUT leave 1.P.NEUT go home with afternoon
'We'll probably finish the work, leave and go home in the afternoon.' (Jaggar, 2001, 192)

(7) Nakàn tāshì dà ƙarfȇ bakwài ìn yi wankā ìn ci àbinci
1.S.HAB get.up clock seven 1.S.NEUT do washing 1.S.NEUT eat food
'I get up at seven, wash and eat.' (Jaggar, 2001, 191)

As witnessed by the data above, in particular the examples involving the potential (6) and habitual (7), TAM interpretation is not confined to the one conjunct explicitly marked, but rather extends to all conjuncts (or the coordinate event as a whole).

A second observation regarding the peculiarity of the "neutral" in this construction pertains to the fact that across-the-board (ATB) extraction apparently triggers relative TAM marking in the first conjunct, but surprisingly does not impede presence of what looks like a subjunctive marker in the second or later conjuncts, as illustrated for continuous (8) and rhetorical (=relative potential; 9) below.

(8) don mȅ yârā sukȅ masà ba'ȁ sù yi
why children 3.P.CONT.REL to.him mockery 3.P.NEUT do
ta jīfànsà dà dutsȅ
throwing.3.S.M.POSS with stone
'Why are the children mocking him and throwing stones at him?' (Newman, 2000, 595)

(9) rìkicī irìn wândà yakà̀ iyà tāsôwā yà tādà̀ manà hankàlī
crisis kind.of REL 3.S.M.RHET can arise 3.S.M.NEUT raise to.us concern
'the kind of crisis that could arise and worry us' (Jaggar, 2001, 192)

Newman (2000) explicitly states that this constitutes a reliable syntactic test to tell the subjunctive and the homophonous "neutral" apart.

Conditional Construction. The second construction where the supposed "neutral" TAM surfaces, is in the consequent of conditional (and temporal) constructions. Again, as detailed by Newman (2000), no subjunctive interpretation necessarily ensues. Rather, what looks like a subjunctive marker will be interpreted as future, potential (Newman, 2000) or habitual (Jaggar, 2001).

(10) In sun zō, ìn tàfi kà̀suwā
if 3.P.COMPL come 1.S.NEUT go market
'If/when they come, I'll go to the market.' (Newman, 2000, 594)

(11) dà cêwā yā shā wùyā, sai yà fashè̀ dà
as soon as COMP 3.S.M.COMPL drink trouble then 3.S.M.NEUT break with
kūkā
crying
'As soon as he has any trouble, then he bursts out crying.' (Jaggar, 2001, 193)

Habitual Construction. The third construction cited by Newman (2000) and Jaggar (2001) as an instance of the "neutral" concerns sequences of events, both marked with a subjunctive or "neutral" marker, yet receiving a generic, habitual interpretation. Again, none of the interpretations typically associated with the Hausa subjunctive, like purpose, volition, obligation, etc. can be detected.

(12) dalà̃ tà hau tà sàuka
dollar 3.S.F.NEUT rise 3.S.F.NEUT fall
'The dollar rises and falls.' (Newman, 2000, 596)

(13) kò̃gîn yà shìga nân yà fita cân
river 3.S.M.NEUT enter here 3.S.M.NEUT exit there
'The river meanders here and there.' (Jaggar, 2001, 193)

In contrast to event sequences initiated by non-subjunctive TAMs, however, the habitual interpretation of the subjunctives does not derive from any overt TAM marker, but is associated with the construction as a whole.

To summarise, the Hausa "neutral" TAM is a category postulated in all three contemporary grammars of Hausa (Wolff, 1993; Newman, 2000; Jaggar, 2001) that is morphologically indistinguishable from the subjunctive, yet apparently fails to receive standard subjunctive interpretation. Rather, its semantic value is derived from the construction it appears in.

2 Previous Approaches

2.1 Newman's TAM Deletion Account

In order to account for the apparent lack of inherent TAM information of "neutral" markers in event sequences, together with the fact that interpretation derives somewhat parasitically from the TAM category overtly marked on the first conjunct, Newman (2000) analyses the neutral marker in non-initial conjuncts as the result of deletion of the TAM part of the agreement-TAM complex, leaving behind a pure "reference tracking" weak subject pronoun.

(14) sukàn tằru sukàn shā tî ↦ sukàn tằru [sù ()] shā tî
(Newman, 2000, 595)

In order to account for the tonal (=all low) properties of the neutral marker which may contrast with that of the agreement part of the TAM category it is derived from (as e.g., in the habitual), Newman (2000) suggests a treatment in phonological terms: assuming that the agreement part of a complex marker is not inherently specified for tone, but rather polar to that of the TAM marker (an observation which is indeed correct for non-fusional markers), deletion of the TAM part leads to assignment of default low tone, as schematised in (15) below.

(15) a. su+kàn ↦ súkàn (Newman, 2000, 595)
b. su+() ↦ sù (Newman, 2000, 595)

In order to account for the equally low-tone ordinary subjunctive, Newman (2000) draws a distinction between the TAM-less "neutral", as a result of TAM deletion, and a zero-marked subjunctive TAM. Since absence of TAM and zero realisation are phonologically indistinguishable, Newman's default L tone assignment successfully derives the tonal identity of subjunctive and "neutral".

Problems with TAM Deletion. In this paragraph, I shall discuss different possible interpretations of a deletion approach to the neutral in sequences of events and show that neither a pure surface-phonological nor a pure surface-syntactic interpretation of this rule is viable, both for theoretical and empirical reasons. Furthermore, I shall argue that the deletion account is empirically fairly restricted in that it fails to generalise, e.g. to habitual interpretation of all subjunctive or "neutral" sequences.

The first observation that should cast some doubt on the viability of a deletion operation like the one advanced by Newman (2000) pertains to phonological predictability: as witnessed by the putative derivations in (16), the deletion operation cannot be a fully regular surface-phonological process. While, e.g., potential *mâ* gets reduced to *mù*, there is a round vowel in the target, but no such round vowel present in the source, despite the fact that the sequence *mwa* is phonotactically possible and even attested in the Hausa negative continuous *bā mwà*. Similar observations can be made regarding the future or rhetorical: again, neither do all the possible realisations in the source have a segmentally

predictable counterpart in the target, nor do all target realisations have an attested correspondence in the source form.

(16) a. mâ/*mwâ ↦ mù/*m

b. zā nì/*zā 'ǹ/*zā nà ↦ 'ǹ/nà/*nì

c. nikǎ̀/*nakǎ̀/*'nkǎ̀ ↦ 'ǹ/nà/*nì

Thus, rather than being the product of a general surface-phonological rule, the inventory of attested "neutral" forms is bounded by the set of available forms in the subjunctive paradigm.

The second possible interpretation, namely that of surface-syntactic deletion, does not appear to be a viable option either. In Hausa, TAM markers select for the form of their complement, either a verb, or, in the continuous, a verbal noun. In event sequences initiated by a continuous, however, only the first conjunct features a verbal noun, whereas in non-initial conjuncts, the "neutral" marker selects a standard verb. Under a surface-deletion approach this is quite unexpected, since selectional restrictions established at the base should actually be preserved.

(17) wǎcē cȅ takȅ dĩnkìn hǔlā tà kai / *kâiwā
who FOC 3.S.F.CONT.REL sowing.VN cap 3.S.F.NEUT carry.V carrying.VN
kàsuwā
market
'Who is sowing the cap and taking it to the market?' (Newman, 2000, 595)

If, however, deletion operates at the base already, it will fail to capture reconstruction of TAM information. Moreover, the idea that TAM information be completely absent from non-initial conjuncts appears counter-intuitive, since choice in the form of complement is otherwise clearly conditioned by an aspectual distinction.

Finally, the surface-syntactic deletion begs the question as to why the resulting exponents are necessarily taken from the same set of forms as the subjunctive: an approach that simultaneously makes reference to morphological paradigms and syntactic configuration is incompatible with basic assumptions of almost every current grammatical theory.

To conclude our discussion of the deletion approach, I should like to point out that its empirical scope is pretty much limited to cases of event sequences initiated by non-subjunctive TAMs: in non-initial consequent clauses of conditionals, no overt full TAM is actually present which can serve as the basis for deletion under identity. Yet, interpretation of neutral draws on the same range of TAM categories (future, potential, habitual, continuous) as those observed in sequences of events. A similar criticism applies to habitual interpretation of subjunctive-only sequences: a deletion account is either incapable of deriving the interpretation of the initial TAM, or else fails to constrain the operation of TAM deletion, harbouring the risk of considerable overgeneration.

2.2 Schuh's Criticism of the "Neutral" TAM

Schuh (2003), in a reply to the analyses advanced by Newman (2000) and Jaggar (2001), questions the validity of a TAM-less paradigm homophonous with the subjunctive, both synchronically and diachronically. Instead, he suggests a largely underspecified denotation of the subjunctive, namely "dependent subordinate inception" which characterises the use of the subjunctive in all contexts, including those referred to as the "neutral".

> *The Subjunctive signals an event which will have its inception subsequent to the moment of speaking and/or to an event in a superordinate clause. The [...] TAM interpretation of the event represented by the Subjunctive is dependent on that of the superordinate clause or operator. This statement has the caveat that the Subjunctive can never function to show simple sequentiality in a string initiated by the Completive or Preterite.*
> (Schuh, 2003, p. 20)

The most fundamental insight of Schuh's approach is that different interpretations of the subjunctive that are typically associated with the subjunctive, like, e.g., hortative, purposive, or volitional, and which clearly go beyond this very basic concept of dependent subsequent inception are contributed not by the subjunctive TAM itself, but rather by the constructions the subjunctive is used in.

In order to substantiate his claim that the Hausa subjunctive is a full TAM category, Schuh (2003) shows that in sequences of events initiated by the continuous, the interpretation of a subjunctive in the second conjunct is distinct from that of an overt continuous: while the use of two continuous markers denotes simultaneity of individual events, use of the subjunctive implies sequentiality of the second event to the first.

(18) sunà tāshì sunà gudù
3.P.CONT arise 3.P.CONT run
'They are arising and running.'

(19) sunà tāshì sù gudù
3.P.CONT arise 3.P.CONT run
'They arise and run.'

Schuh further observes that in embedded contexts, typical functions of the subjunctive, such as permission, prohibition, volition, purpose etc. are provided by the embedding semantic relation. Thus, "subjunctive" functions in addition to dependent subordinate inception are imposed by the embedding context. In matrix clauses, however, an embedding context on which the subjunctive could be dependent is obviously absent. He sketches two possible solutions, namely, to either extend the licensing conditions of the subjunctive to include reference to extra-sentential context, or else to capture the hortative interpretation associated with matrix subjunctive by the introduction of an implicit hortative operator.

Schuh tries to extend the scope of his definition of the subjunctive to explain its unacceptability in relative and focus contexts. With respect to relatives he states that they "assert a property of the antecedent", something he claims to be "antithetical to the dependent nature of the subjunctive." (Schuh, 2003, p. 30). However, the ban on subjunctive TAM in relative clause constructions appears to me to be a syntactic, rather than semantic constraint, since, e.g., interpretation as an implicit hortative is equally impossible in this environment. Conversely, the possibility to employ paraphrases to circumvent the apparent restriction towards the subjunctive also militates in favour of the syntactic nature of the constraint. Moreover, the syntactic perspective relates the ban on subjunctives to the equally syntactically conditioned (Tuller, 1986) alternation between relative and absolute paradigms. Similar to his take on relatives, Schuh also pursues a semantic approach to explain the impossibility of the subjunctive as complement to verbs of perception and cognition, claiming *independence* of events. Again, since acceptability does not improve with semantic reinterpretation, ban on subjunctive is best understood as a syntactic restriction.

Before I close the discussion of Schuh's approach to the Hausa subjunctive, I shall point out a few remaining issues with his account that I should hope to resolve in the formal analysis in the next section. First, the proposal is not always very explicit about the syntactic analysis assumed, in particular the question of whether sequences are treated as coordinations or as subordinations. Similarly, it remains unclear whether "dependent" is supposed to be a syntactic or a semantic concept, and whether this concept is to be interpreted in terms of dominance or precedence: while the discussion of embedded subjunctives indeed implies dominance, that of sequences in relative contexts makes reference to precedence:

> *But the governing factor for the TAM [...] is not the focus [...] construction [...], but rather sequentiality to the preceding Continuous TAM.*
> (Schuh, 2003, p. 31)

Similarly, subjunctives in conditional constructions correspond to the matrix event, so, again, the exact notion of dependence will need to be subjected to some further refinement.

Once these notions are made precise the approach by Schuh (2003) according to which both subjunctive and "neutral" TAM receive a unified account as a temporally dependent category can provide a sound basis for a formal account of the Hausa subjunctive that directly constrains its distribution and interpretation. Given the theoretical and empirical limitations of the deletion approach discussed in the previous section, I conclude that a direct, surface-syntactic approach to the distribution of the subjunctive is clearly to be preferred.

3 An HPSG Account

In this section, I shall outline a formal syntactic analysis of the Hausa subjunctive, developed within the framework of Head-driven Phrase Structure Grammar (HPSG; Pollard and Sag, 1987, 1994), using Minimal Recursion Semantics (MRS; Copestake et al., 2005) as meaning representation language.

As for the linguistic analysis, I shall follow Schuh (2003) in assuming a single subjunctive TAM category devoid of any modal force. Essentially, I shall derive modal properties occasionally associated with particular uses of the subjunctive by means of either overt embedding predicates/operators or else by means of modal coercion, implemented as properties of particular constructions.

In particular, I shall interpret the subordination requirement identified by Schuh (2003) as a semantic condition which will enforce introduction of implicit modal operators where appropriate, i.e., in matrix contexts. The net effect of such a semantic constraint will be coercion into a hortative interpretation. Syntactic constructions that impose an absolute ban on subjunctive TAMs, by contrast, will be analysed in syntactic rather than semantic terms.

As for event sequences, I shall formally model these constructions as a specific subtype of coordinate structures where the TAM information expressed on the first conjunct is asymmetrically projected to the group event, thereby capturing the intuition expressed by Newman (2000) and Jaggar (2001) that TAM-marking of the first conjunct has scope over the entire sequence.

As a first step towards an HPSG treatment of the Hausa subjunctive we need to make precise the condition regarding the subjunctive as a subjoined, i.e. embedded TAM category. As we have seen in the previous section, obligatory hortative coercion is essentially limited to root contexts, i.e., it is triggered, whenever the subjunctive would otherwise remain semantically undominated. Thus, I shall depart from the hypothesis that Schuh's empirical generalisation of the subjunctive as a dependent TAM category can be largely equated with a *qeq* relation in Minimal Recursion Semantics.

$$\left[\text{SYNSEM}\,|\,\text{LOC}\,|\,\text{CONT}\,|\,\text{RELS}\left\langle \ldots \left[\begin{array}{ll}\text{ARG0} & \left[\text{TAM}\quad \textit{subj}\right] \\ \text{LBL} & \boxed{l}\end{array}\right]\ldots\right\rangle\right]$$

$$\rightarrow \left[\text{SYNSEM}\,|\,\text{LOC}\,|\,\text{CONT}\left[\begin{array}{ll}\text{RELS} & \left\langle \ldots \left[\alpha \quad \boxed{h}\right]\ldots\right\rangle \\ \text{HCONS} & \left\langle \ldots \boxed{h} =_q \boxed{l}\ldots\right\rangle\end{array}\right]\right]$$

where $\alpha \in \{\text{ARG1},\text{ARG2},\text{ARG3},\text{ARG4},\text{L-HNDL},\text{R-HNDL}\}$

Fig. 1. Subjunctive subordination constraint (root condition)

As depicted by the root condition in Figure 1, the distinguished label of an event marked with the subjunctive must be equal (modulo quantification) to an argument handle of some other elementary predication, i.e., must be semantically embedded as an argument of some other predicate. Some trivial cases include conjunctions and complement taking predicates, comprising modal, volitional, and permissive ones.

Implicit Modal Force. If subjunctive mood is a largely underspecified TAM category, modal force, if present, must be introduced by independent predicates.

In the absence of an appropriate governing lexical predicate, such introduction can only be constructional.

In the discussion of the empirical data, we have identified exactly two such situations where the meaning of the whole was not strictly composed of the meaning of its parts, i.e. where the subjunctive was associated with a meaning that cannot be identified as part of its inherent core meaning as a TAM category. The first one is the hortative interpretation that ensues with root subjunctives, the second one a conventionalised habitual generic interpretation associated with sequences of subjunctive events.

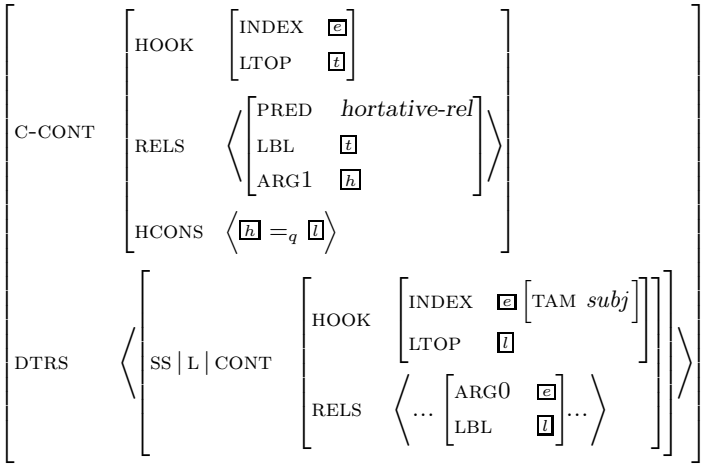


Fig. 2. Hortative construction

In order to capture the hortative, I shall suggest a unary construction as schematised in Figure 2. Essentially, this unary phrase structure schema syntactically dominates a subjunctive event and contributes an implicit hortative relation that outscopes an elementary predication corresponding to the subjunctive event. Together with the general constraint on semantic dependence introduced above, this construction successfully models obligatory hortative coercion in root contexts. I.e., introduction of an implicit dominating relation is the only way to license root subjunctives in Hausa.

Event Sequence Constructions. Before I enter into the discussion of how event sequences can be modelled, I shall briefly summarise how coordinate structures are treated in HPSG and then proceed towards an analysis of event sequences as a special subtype of asyndetic coordination.

The standard treatment of coordinated events in HPSG is symmetric: i.e., coordinate constructions do not possess a unique head, but rather identify the categorial and non-local information of the mother with that of both conjunct daughters (Pollard and Sag, 1994). Categorial (CAT) values comprise

both HEAD information, i.e., basic part of speech, as well as valence information, capturing the observation that degree of saturation enters into categorial likeness conditions in coordinate structures. Identity of NON-LOCAL values implements the Across-the-board (ATB) constraint. Semantically, however, event coordinations contribute a group event which embeds the events of the conjuncts (Copestake et al., 2005) via distinguished argument roles (L-HNDL and R-HNDL). TAM information of coordinate structure is typically not shared across conjuncts, as illustrated by the fact that sentences with different TAM specifications can easily coordinate.

(20) I bought a record yesterday and will probably resell it, once I have made a copy.

The assumption that Hausa event sequences are best understood as coordinate structures is motivated by the fact that asyndetic coordination is independently a common strategy for coordinations of events, but not for coordinations of individuals.

The event sequence construction, however, differs from ordinary coordinations in that it places tighter restrictions on the TAM categories of the conjuncts. A trivial observation pertains to the fact that the second conjunct appears in the subjunctive. Furthermore, Newman (2000) and Jaggar (2001) observe that the TAM information encoded on the first verb is pertinent to all events in the sequence. Thus, even if the second sub-event is sequentially dependent on the first (Schuh, 2003), it is still interpreted as part of a group event whose TAM value is fixed by the first conjunct. I shall therefore assume that in this construction, the interpretation of, e.g., habitual, potential, and (generic) continuous clearly shows that TAM information of the initial event projects to the group event.

Under this perspective, subjunctive TAM in non-initial conjuncts merely marks a subsequence relation between subevents.

Note that the absence of any modal force in event sequence constructions (Figure 3) that was the major motivation behind the postulation of a "neutral" TAM distinct from the subjunctive is readily accounted for by the fact that conjuncts in coordinated constructions are always dominated by the group event. Thus, there is no obligatory coercion into, e.g., a hortative reading.

Subjunctive in Relative Contexts. We have seen above that subjunctive TAM observes a blanket ban to surface in relative contexts, independent of interpretation. This ban on subjunctives can be modelled straightforwardly on the assumption that head-filler structures constrain the head not to be specified with an absolute TAM, but that the subjunctive is indeed an inherently absolute TAM category. Embedding of the subjunctive under some predicate, e.g., *dōlè*, renders it acceptable in clauses containing a local filler, since *dōlè*, rather than the more deeply embedded subjunctive will be the relevant head. Put differently, syntactic intervention of *dōlè* (and other predicates) will remove the subjunctive from the local configuration.

Embedding under sequential coordination apparently has the same effect: because the conjoined structure as a whole will be the sister of the filler in a

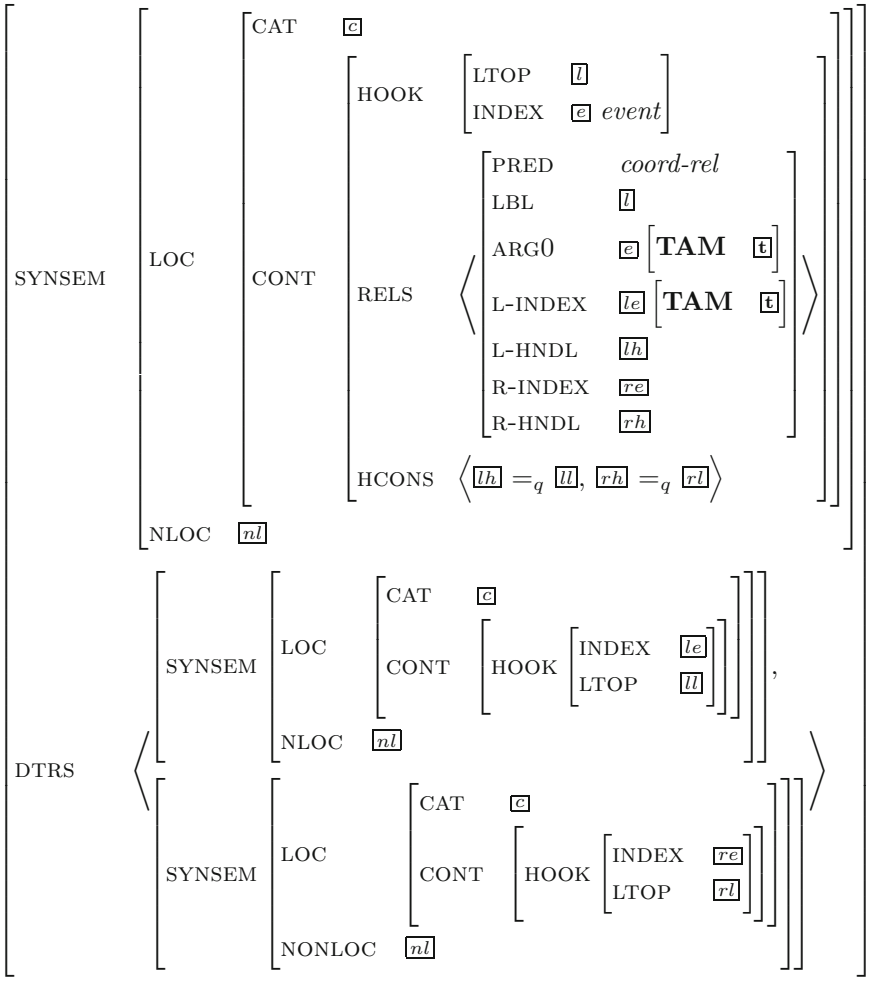


Fig. 3. Event sequence construction

filler-head structure, it is the conjoined structure's TAM specification that is constrained to relative TAM. Since TAM information is asymmetrically projected from the initial conjunct, the left conjunct-daughter is equally constrained to relative TAM, ruling out all-subjunctive sequences. The right conjunct daughter, however, is unconstrained by the head-filler structure, since it is syntactically and semantically embedded.

Constructional Interpretation of Event Sequences — Habitual/generic. The last construction we will address concerns sequences of subjunctives receiving a habitual interpretation, a construction which has so far not received an analysis in the linguistic literature on Hausa.

Note that sequential coordination opens up the possibility for both events to be in the subjunctive. Furthermore, our current outscoping requirement licenses non-modal interpretation for both conjuncts, since we merely require semantic dominance. Projection of the left conjunct TAM, however, subjects the entire coordinate structure to licensing, giving rise to the following possibilities: subordination under an explicit predicate, be it a conjunction, or a suitable verb, or else embedding under an implicit modal, such as the hortative construction introduced above. Thus, in order to capture the habitual interpretation, all that is required is to expand our inventory of constructions.

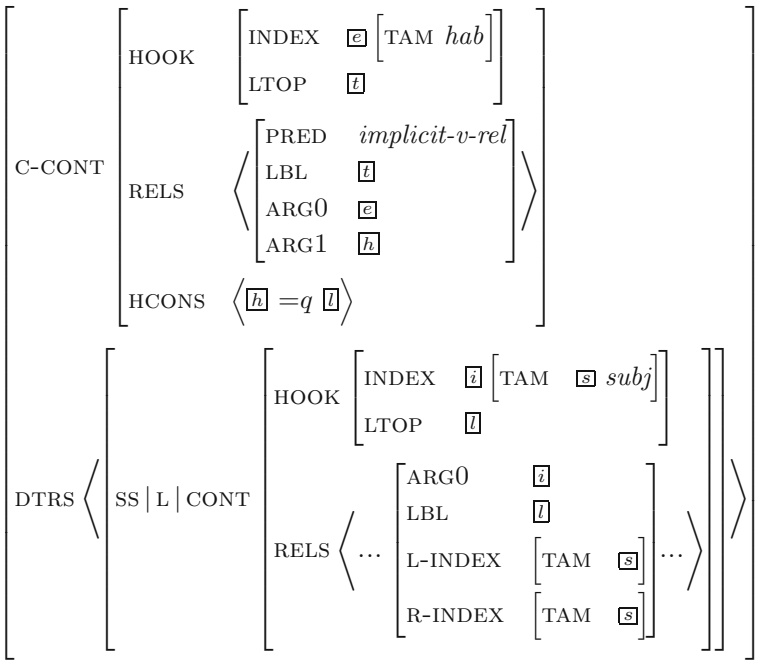


Fig. 4. Habitual construction

The unary schema in Figure 4 introduces an implicit habitual relation that takes a coordinated event structure as its complement. The fact that habitual interpretation only arises with subjunctive TAM sequences can be captured quite naturally by placing appropriate constraints on the semantics of the daughter.

4 Conclusion

In this paper, I have addressed the distribution and interpretation of the Hausa subjunctive, investigating contrasts between constructions where the subjunctive is or fails to be associated with additional modal force.

I have shown in particular that a deletion account of the Hausa subjunctive/neutral in event sequence constructions (Newman, 2000) is not only plagued with conceptional problems, like the postulation of two homophonous paradigms, but also theoretically questionable. I have argued that the underspecified characterisation of the subjunctive as a "dependent subsequent inception" (Schuh, 2003) paves the way for a unified analysis of modal and non-modal subjunctive uses.

By giving the dependent nature of the Hausa subjunctive a formal interpretation as an *outscopes* requirement on some elementary predication, we were able to capture not only the modal coercion facts, but also to extend the analysis to cover event sequences, including generic interpretation. Finally, we saw how asymmetric projection of TAM values from initial conjuncts could capture the intuition of Newman (2000), regarding interpretation of "neutral" as indirectly TAM-marked, as well as make the right predictions with respect to relative contexts.

References

Copestake, A., Flickinger, D., Pollard, C., Sag, I.: Minimal recursion semantics: an introduction. Research on Language and Computation 3(4), 281–332 (2005)

Crysmann, B.: An inflectional approach to Hausa final vowel shortening. In: Booij, G., van Marle, J. (eds.) Yearbook of Morphology 2004, pp. 73–112. Kluwer (2005)

Davis, A.: Syntactic binding and relative aspect markers in Hausa. In: Proceedings of the Fifteenth Annual Conference on African Linguistics, Los Angeles, CA (1984, 1986)

Jaggar, P.: Hausa. John Benjamins, Amsterdam (2001)

Newman, P.: The Hausa Language. An Encyclopedic Reference Grammar. Yale University Press, New Haven (2000)

Pollard, C., Sag, I.: Information–Based Syntax and Semantics, vol. 1. CSLI, Stanford (1987)

Pollard, C., Sag, I.: Head–Driven Phrase Structure Grammar. CSLI and University of Chicago Press, Stanford (1994)

Schuh, R.: The functional unity of the Hausa and Chadic subjunctive. Kandybowicz (ed.) Papers in African Linguistics 3. UCLA Working Papers in Linguistics, vol. (9). UCLA (2003)

Tuller, L.A.: Bijective Relations in Universal Grammar and the Syntax of Hausa. PhD thesis, UCLA, Ann Arbor (1986)

Wolff, E.: Referenzgrammatik des Hausa. LIT, Münster (1993)

Memory Resource Allocation in Top-Down Minimalist Parsing*

Gregory M. Kobele[1], Sabrina Gerth[2], and John Hale[3]

[1] University of Chicago, Chicago, Illinois, USA
[2] Universität Potsdam, Potsdam, Germany
[3] Cornell University, Ithaca, New York, USA

Abstract. This paper provides a linking theory between the minimalist grammar formalism and off-line behavioural data. We examine the transient stack states of a top-down parser for Minimalist Grammars as it analyzes embedded sentences in English, Dutch and German. We find that the number of time steps that a derivation tree node persist on the parser's stack derives the observed contrasts in English center embedding, and the difference between German and Dutch embedding. This particular stack occupancy measure formalizes the leading idea of "memory burden" in a way that links predictive, incremental parsing to specific syntactic analyses.

1 Introduction

An important goal of linguistics is to account for human linguistic behavior. A cognitive scientist might, following David Marr [25], attempt to analyze behavioral data by viewing the syntactician's grammatical analysis as a high level description of a parser (cf. [20]). Deviations from the categoricity 'predicted' by the grammar could be given a natural explanation using the more refined vocabulary of parser states, and memory resource consumption.

A number of theoretical proposals within psycholinguistics over the years [13,42,6,22] have been built around the idea that our capacity to remember words and phrases plays an important role in normal human language comprehension. These theories link observed processing contrasts between sentences to differences in theorized memory requirements. These contrasts are observed empirically in sentence types whose grammatical analysis is a matter of active research within the field of syntax. This presents a problem: ideally, the same analysis that is supported by comparative linguistic research ought to derive observed processing contrasts. But in some previous memory burden theories, syntactic assumptions were left implicit or oversimplified (but cf. [32]). Under these conditions, it becomes difficult to say exactly which aspect or aspects of sentence structure motivates the memory burdens to which the theory appeals as explanations. Yet an emerging body of work suggests that the choice of syntactic analysis may matter greatly [9,40] (cf. section 5.2).

* An anonymous reviewer provided very helpful comments, which have greatly improved the quality of this paper.

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 32–51, 2013.

This work strives to address this problem. We proceed by defining grammar fragments whose syntactic commitments are clear. Starting from a mildly context sensitive grammar [37] we assume a top-down parsing strategy [36]. This additional assumption is motivated by evidence for predictive human sentence comprehension [26,39]. Using independently motivated syntactic analyses, we derive complexity profiles consistent with two key contrasts that have been traditionally acknowledged in the literature. An additional benefit of this is that the pathway to semantic interpretation is entirely standard [30,18].

2 Methodology

We investigate a measure of parsing complexity for minimalist grammars [37], a formalism based on transformational generative grammar [4]. Parsing, like other nondeterministic algorithms, is naturally viewed as a *search* problem [14], where the nondeterminicity is resolved by an independent search strategy such as depth-first, breadth-first or A*. In an ideal parser, heuristics would perfectly guide its actions at every choice point. In this optimal case, the amount of resources consumed during a parse of a sentence is identical to the amount of resources needed to traverse its parse tree (no additional resources are consumed by working on ultimately 'incorrect' parses, and backtracking). We adopt this simplifying assumption here, noting that something like it (beam search with a narrow beam) has been proposed as a natural way to capture the 'dual nature – generally good and occasionally pathological – of human linguistic performance' [5].

Our methodology extends [12]. The main idea is to advance a particular automaton model of sentence processing, examining certain aspects of the automaton's configuration as it parses strings whose abstract structure mirrors that of the sentences used in comprehension experiments with humans. We explore a measure of parsing complexity based on the allocation of memory resources during a successful parse. One memory unit is considered allocated per item on the parser stack, where the parser stack holds predictions yet to be verified. Following Joshi and Rambow [12,33], our complexity metric reflects the amount of "time" that an item is retained in memory; we call this *stack tenure* (even though the actual data structure is a priority queue, cf. Section4.1). From the state-trajectory of the top-down automaton over the correct derivation tree, we calculate the number of time steps that an item is retained. The length of the longest sequence of automaton transitions for which the same item remains in memory we identify as the *maximal tenure* of the parse.

The remainder of this paper is structured as follows. Section 3 briefly introduces the processing phenomena that this modelling work addresses. Section 4 then introduces Minimalist Grammars, and a top-down parser for them. Section 5 reports the stack tenure measures obtained by simulating the parser on the sentences in question. Section 6 takes up the relationship between this approach and other related proposals in the literature. Section 7 concludes, and mentions some directions for future work.

3 Embedding Phenomena

The notion of tenure (to be introduced in Section 4.1) can be thought of as a way of formalising intuitions about dependency length [6]. The following phenomena are naturally understood in these terms, and we will see (Section 5) that this intuitive understanding can in fact be cashed out rigorously in terms of tenure.

3.1 English Center Embedding, as Compared to Right-Branching

Center embedding in English is notoriously difficult to comprehend, in comparison to a right-branching alternative [28]. Center-embedded examples like 1 below can be made more and more difficult by interposing additional relative clause modifiers after each new subject noun phrase. This embedded material intervenes between the subject and the verb. By contrast, in right-branching examples like 2, it is the object that is modified. In these cases, the distance between subject and verb remains the same.

(1) The boy that the girl that the cat licked loves laughed.
(2) The cat licked the girl that loves the boy that laughed.

Resnik [34] (cf. [11,1]) expresses what has become the standard account of this contrast. He proposes that an explanation for the radical unacceptability of center embedded sentences can be made to follow elegantly from constraints on memory resources in a parser. He shows that a left-corner, but not a bottom-up or a top-down, parser for a context-free grammar requires more memory to process center embedded structures than peripherally embedded ones.

It is, however, widely accepted [35] that context-free grammars are unable to assign structures to sentences which allow for a transparant description of their meaning. Unfortunately, once we move to more sophisticated grammar formalisms, which *do* seem able to assign semantically appropriate structures to sentences, the notion of left-corner parsing is either ill-defined or not yet discovered. In other words, the explanation of the processing difficulty of center embedded structures based on imposing memory restrictions on a left-corner parser does not transfer directly to linguistically plausible grammar formalisms. To reconcile the evidence motivating mild context-sensitivity with the selectively greater difficulty of center-embedding requires some alternative automaton model, such as the one to be presented in section 4.

3.2 Embedded vs Cross-Serial Verb Clusters

Bach et al. [2] observe that the rated comprehensibility of German embeddings with sentence-final verb clusters (3) increases more sharply than does the corresponding rating of Dutch cross-serial items (4), as the number of verbs in the cluster grows.

(3) daß Hans Peter Marie schwimmen lassen sah
that Hans Peter Mary swim let saw
"that Hans saw Peter let Mary swim"

(4) dat Jan Piet Marie zag laten zwemmen
that Jan Peter Mary saw let swim
"that Jan saw Peter let Mary swim"

In other words, German examples such as 3 are more difficult to process than Dutch examples such as 4.

From a formal perspective, this is surprising, as the Dutch cross serial pattern, under the semantically appropriate pairing of nouns and verbs, is mildly context sensitive ($N_1\ N_2\ N_3\ V_1\ V_2\ V_3 \in ww$), whereas the German nested pattern ($N_1\ N_2\ N_3\ V_3\ V_2\ V_1 \in ww^R$) can be generated by a less expressive context-free grammar. This is thus an example where language theoretic complexity does not coincide with 'behavioural complexity'. A natural intuition is to link this contrast to the length of the dependencies between nouns and their selecting verbs – in the Dutch case, the longest dependency is checked first, whereas in German it is checked last (N_1 and V_1).

4 Minimalist Grammars

Minimalist grammars make use of two syntactic structure building operations; binary **merge** and unary **move**. Whether a structure building operation is defined on a particular object in its domain (a pair of expressions or a single expression) is determined solely by the syntactic categories of these objects. In minimalist grammars, syntactic categories are finite sequences of 'features'. The currently accessible feature is the feature at the beginning (leftmost) position of the list. In order for **merge** to apply, the heads of its arguments must have matching first features. These features are eliminated in the derived structure which results from their merger. In the case of **move**, the head of its argument must have a feature matching a feature of the head of one of its subconstituents. In the result, both features are eliminated. Each feature type has an attractor and an attractee variant, and for two features to match, one must be an attractor and the other an attractee. For **merge**, the attractee feature is a simple categorial feature, written `x`. There are two kinds of attractor features, `=x` and `x=`, depending on whether the selected expression is to be merged on the right (`=x`) or on the left (`x=`). For the **move** operation, there is a single attractor feature, written `+y`, and a single attractee, `-y`.

We write lexical items using the notation $\langle \sigma, \delta \rangle$, where σ is a (phonological) string, and δ is a feature bundle. Complex expressions are written using the notation of [37] for the 'bare phrase structure' trees of [4]. These trees are essentially X-bar trees without phrase and category information represented at internal nodes. Instead, internal nodes are labeled with 'arrows' > and <, which point to the head of their phrase. A tree of the form $[_< \ \alpha \ \beta]$ indicates that the head is to be found in the subtree α, and we say that α projects over β, while one of the form $[_> \ \alpha \ \beta]$ that its head is in β, and we say that β projects over α. Leaves are labeled with lexeme/feature pairs (and so a lexical item $\langle \alpha, \delta \rangle$ is a special case of a tree with only a single node). The head of a tree t is the leaf one arrives at from the root by following the arrows at the internal nodes. If t is a bare phrase structure tree with head H, then we will write $t[\text{H}]$ to indicate this. (This means we can write lexical items $\langle \alpha, \delta \rangle$ as $\langle \alpha, \delta \rangle[\langle \alpha, \delta \rangle]$.) The **merge** operation is defined on a pair of trees t_1, t_2 if and only if the head of t_1 has a feature bundle which begins with either `=x` or `x=`, and the head of t_2 has a feature bundle beginning with the matching `x`

feature. The bare phrase structure tree which results from the merger of t_1 and t_2 has t_1 projecting over t_2, which is attached either to the right of t_1 (if the first feature of the head was =x) or to the left of t_1 (if the first feature of the head was x=). In either case, both selection features are checked in the result.

$$\textbf{merge}(t_1[\langle \alpha, \texttt{=x}\delta\rangle], t_2[\langle \beta, \texttt{x}\gamma\rangle]) = \begin{array}{ccc} & < & \\ t_1[\langle \alpha, \delta\rangle] & & t_2[\langle \beta, \gamma\rangle] \end{array}$$

$$\textbf{merge}(t_1[\langle \alpha, \texttt{x=}\delta\rangle], t_2[\langle \beta, \texttt{x}\gamma\rangle]) = \begin{array}{ccc} & > & \\ t_2[\langle \beta, \gamma\rangle] & & t_1[\langle \alpha, \delta\rangle] \end{array}$$

If the selecting tree is both a lexical item and an affix (which I notate by means of a hyphen preceding/following the lexeme in the case of a suffix/prefix), then head movement is triggered from the head of the selected tree to the head of the selecting tree.

$$\textbf{merge}(\langle \text{-}\alpha, \texttt{=x}\delta\rangle, t_2[\langle \beta, \texttt{x}\gamma\rangle]) = \begin{array}{ccc} & < & \\ \langle \beta\text{-}\alpha, \delta\rangle & & t_2[\langle \epsilon, \gamma\rangle] \end{array}$$

The operation **move** applies to a single tree $t[\langle \alpha, \texttt{+y}\delta\rangle]$ only if there is *exactly one* leaf ℓ in t with matching first feature -y or $\ominus$y. This is a radical version of the shortest move constraint [4], and will be called the SMC – it requires that an expression move to the first possible landing site. If there is competition for that landing site, the derivation crashes (because the losing expression will have to make a longer movement than absolutely necessary). If it applies, **move** moves the maximal projection of ℓ to a newly created specifier position in t, and deletes both licensing features. To make this precise, let $t\{t_1 \mapsto t_2\}$ denote the result of replacing all subtrees t_1 in t with t_2, for any tree t, and let ℓ_t^M denote the maximal projection of ℓ in t, for any leaf ℓ.

$$\textbf{move}(t[\langle \alpha, \texttt{+y}\delta\rangle]) = \begin{array}{ccc} & > & \\ t'[\langle \beta, \gamma\rangle] & & t[\langle \alpha, \delta\rangle]\{t' \mapsto \langle \epsilon, \epsilon\rangle\} \end{array}$$

(where $t' = \langle \beta, \texttt{-y}\gamma\rangle_t^M$)

A derivation tree is an element of the term language over the ranked alphabet $A_0 \cup A_1 \cup A_2$, where $A_0 = Lex$ is the set of nullary symbols, $A_1 = \{\mathbf{v}\}$ is the set of unary symbols, and $A_2 = \{\mathbf{r}\}$ the set of binary symbols. As a consequence of the translation of minimalist grammars into multiple context free grammars [27,10], the set of derivation trees in a minimalist grammar of an expression with unchecked feature string γ at the root and no features anywhere else is regular.

As an example, the lexical item *John* in figure 3(b) has the feature sequence '`d -k`', which indicates that it must first be the second argument to the merge operation (where the first argument has a matching =d feature), and then it will, as part of a larger expression whose head has a +k feature, be targeted by the move operation. Similarly, the lexical item *laugh* with feature sequence '`=d v`' indicates that it must first be the

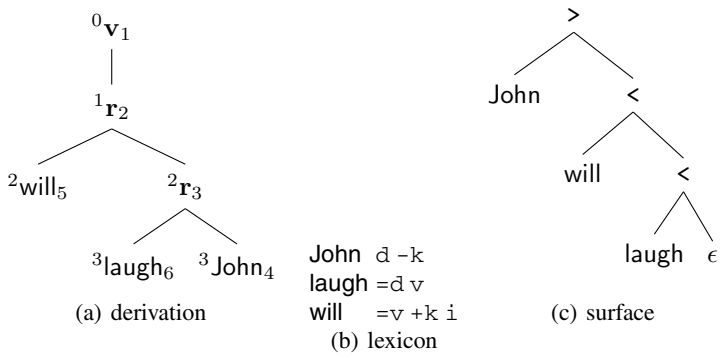


Fig. 1. Structures for "John will laugh"

first argument to the merge operation (where the second argument has a matching d feature), and then it may be the second argument to the merge operation (where the first argument has matching =v feature). The sequence of rule applications used in the construction of a sentence can itself be viewed as a tree, as in figure 1(a), which describes the derivation of the surface structure in 1(c). In figure 1(a), internal nodes are labeled either **v** (for move) or **r** (for merge), and leaves are labeled with lexical items (we have suppressed the features for reasons of space). The internal node immediately dominating the leaves *laugh* and *John* is labeled **r**, which indicates that the lexical items *laugh* and *John* were merged together. The parent of this node is also labeled **r**, and indicates that the lexical item *will* was merged together with the result of merging together *laugh* and *John*. The derived tree in figure 1(c) is therefore the result of applying the move rule (**v**) to the result of merging *will* together with the result of merging together *laugh* and *John*.

The nodes of the derivation tree in figure 1(a) are superscripted (on the left) and subscripted (on the right). Derivation trees marked up in this way are a very condensed yet *complete* representation of a parse of a sentence. Nodes represent parser items, the superscript indicates at which parsing step that node is put into the parser's stack and the subscript indicates at which parsing step it is removed from the parser's stack.[1] The order in which predicted items are expanded is determined by their order in the surface tree, which is only computed implicitly by the parser. Compare the order of terminals in the surface tree 1(c) with the order the leaves of the derivation tree are expanded. Table 1 reconstructs the parser stack at each step from the marked-up derivation tree in 1(a). The underlining in table 1 indicates which item is operated on in the subsequent step. Items are represented in the table as terms, where **S** is the initial item, and for α an item, $\mathbf{v}(\alpha)$, $\mathbf{r}(\alpha)_1$, and $\mathbf{r}(\alpha)_2$ is the result of applying an unmove rule to α, the first element of the pair resulting from applying an unmerge rule to α, and the second

[1] A more precise characterization of the relation between a marked up derivation tree and the sequence of parser states in a successful top down parse of that derivation is that an item corresponding to a node ${}^{i}\alpha_{j}$ is in the stack at time t iff $i \leq t < j$.

Table 1. The sequence of parser configurations corresponding to figure 1(a)

0	—	$\{\underline{\mathbf{S}}\}$
1	—	$\{\underline{\mathbf{v}(\mathbf{S})}\}$
2	—	$\{\underline{\mathbf{r}(\mathbf{v}(\mathbf{S}))_1},\ \underline{\mathbf{r}(\mathbf{v}(\mathbf{S}))_2}\}$
3	—	$\{\underline{\mathbf{r}(\mathbf{v}(\mathbf{S}))_1},\ \underline{\mathbf{r}(\mathbf{r}(\mathbf{v}(\mathbf{S}))_2)_1},\ \underline{\mathbf{r}(\mathbf{r}(\mathbf{v}(\mathbf{S}))_2)_2}\}$
4	—	$\{\underline{\mathbf{r}(\mathbf{v}(\mathbf{S}))_1},\ \mathbf{r}(\mathbf{r}(\mathbf{v}(\mathbf{S}))_2)_1\}$
5	—	$\{\underline{\mathbf{r}(\mathbf{r}(\mathbf{v}(\mathbf{S}))_2)_1}\}$
6	—	$\emptyset$

element of the same, respectively. (See Section 4.1 for more details.) Observe that in, for example, step 2, all and only items corresponding to nodes in figure 1(a) with a superscript less than or equal to 2 and a subscript greater than 2 appear in the stack – at parsing step 2, there are two items in the parser state; one corresponding to a prediction of the lexical item *will* ($\mathbf{r}(\mathbf{v}(\mathbf{S}))_1$), and one to a prediction of a VP-like constituent ($\mathbf{r}(\mathbf{v}(\mathbf{S}))_2$) – the node labelled ${}^2\mathbf{r}_3$). In the next step, instead of scanning a word from the input as would a context-free parser, the prediction ${}^2\mathbf{r}_3$ is expanded. The prediction of *will* remains in the parser state until the fifth step.

Because the marked up derivation tree concisely encodes the entire parse history of an expression, we use it exclusively in the remainder of this paper.

4.1 Parsing

A top down minimalist parser explores a search space defined by inverting the operations of merge and move (and called in [10] *unmerge* and *unmove*). As mentioned in section 2, we assume that the parser is equipped with a perfect oracle, and will ignore the bookkeeping necessary to perform backtracking.[2] This assumption amounts to claiming that the asymmetries in acceptability judgements in the constructions we examine here are not due to (local) ambiguities; either because there are none, as we assume here, or because all sentences involved have roughly the same amount.

As in the case of context-free parsing, a minimalist parser item corresponds to a node in a derivation tree, and a minimalist parser state is a sequence of minimalist parser items. Just like with context-free parsing, a (top-down) parser state represents the set of derivation tree contexts with the same open positions; the parser items it contains correspond to the categories of the open positions, and the order in which these open positions might correspond to the input string. Differences between them, however, stem from the following difference between the grammar formalisms: the language of a given nonterminal in a minimalist grammar consists of *tuples* of derived objects [38], as opposed to single derived object as in the case of context-free grammars. Accordingly, the minimalist parser's 'stack' needs to take the form of a priority queue. As shown by [24], an ordering on derivation tree nodes reflecting the corresponding node in the surface tree can be computed efficiently by the parser online (a finite copying transduction

[2] This decision is motivated also by the fact that search strategy and oracle are highly underdetermined by the search space, and that we do not know how to select among the alternatives in a principled way.

relates the two [19,29]). The order on queue elements is given by this relation. In some sense, the crucial difference between minimalist parsing and context-free parsing is that the ordering relation between nodes in the minimalist derivation tree (defined as per the above in terms of the surface string position of the leftmost component) is *not* inherited through dominance, whereas that in the context-free derivation tree *is*.[3] This allows the priority queue for context-free parsing to behave as a simple stack.

We do not present the full set of parser rules here, for reasons of space and simplicity (we leave out the rules for head movement, as this complicates things even more; see footnote 5); see [24] and [36] for more optimized versions hereof.[4] The parser rules are presented as (upside down) inference rules, such as the below.

$$\frac{P}{R_1 \dots R_n}$$

This rule is to be understood as saying that the items $R_1, \dots, R_n$ are derivable in one step from item P. Given a stack (or some similar data structure) whose top item is P, applying this rule removes P from the stack, and adds $R_1 \dots R_n$. If $n < 1$, this means that P is simply removed from the stack.

A parser item takes the form $\langle \gamma_0, \alpha_0; \gamma_1, \alpha_1; \dots; \gamma_n, \alpha_n \rangle$, where for $0 \leq i \leq n$, α_i is the feature sequence of the i^{th} moving expression, and γ_i is the gorn address of its position in the derived tree.[5] The parser items are totally ordered, where $p < p'$ iff the leftmost γ_i in p is to the left of the leftmost γ'_j in p'.[6]

The **scan** rule is given below. Here, a parser item is assumed to correspond to a lexical item, and is removed from the stack.

$$\frac{\langle \gamma, \alpha \rangle}{} \quad \text{where u is the next word and } \langle \mathsf{u}, \alpha \rangle \in Lex \qquad \textbf{(scan)}$$

For conciseness, we write an item $\langle \gamma_0, \alpha_0; \gamma_1, \alpha_1; \dots; \gamma_n, \alpha_n \rangle$ as $\langle \gamma_0, \alpha_0; A \rangle$. We use the notation $\langle \gamma_0, \alpha_0; A[\phi] \rangle$ to indicate that A contains ϕ. If $A[\psi]$ occurs in the antecedent of a rule, then $A[\phi]$ in the consequent indicates that ψ has been replaced by ϕ in A. We write $A[-]$ to indicate that ψ has been removed. We write $A = B \oplus C$ to indicate that A can be partitioned into B and C.

unmerge1s

$$\frac{\langle \gamma, \alpha; A \oplus B \rangle}{\langle \gamma 1, \mathtt{x}{=}\alpha; A \rangle \quad \langle \gamma 0, \mathtt{x}; B \rangle}$$

unmerge2s

$$\frac{\langle \gamma_0, \alpha_0; A \oplus B[\gamma, \alpha] \rangle}{\langle \gamma_0 1, \mathtt{x}{=}\alpha; A \rangle \quad \langle \gamma, \mathtt{x}\alpha; B[-] \rangle}$$

[3] An ordering ($<$) over derivation tree nodes is inherited through dominance in this sense just in case for any two nodes a and b, $a < b$ implies that, for any children c_a and c_b of a and b, $c_a < c_b$.

[4] In particular, we should restrict the feature sequences in predicted items to lexical feature suffixes.

[5] To incorporate head movement, we need to decompose γ_0 into the triple $\gamma_0^0, \gamma_0^1, \gamma_0^2$; see e.g. [16].

[6] This is a total order because the gorn addresses assigned to items in the stack are unique, corresponding as they do to positions in the derived tree.

$$\textbf{unmerge1c}\quad \frac{\langle \gamma, \alpha; A \oplus B\rangle}{\langle \gamma 0, {=}\mathtt{x}\alpha; A\rangle \quad \langle \gamma 1, \mathtt{x}; B\rangle} \qquad \textbf{unmerge2c}\quad \frac{\langle \gamma_0, \alpha_0; A \oplus B[\gamma, \alpha]\rangle}{\langle \gamma_0 0, {=}\mathtt{x}\alpha; A\rangle \quad \langle \gamma, \mathtt{x}\alpha; B[-]\rangle}$$

In the case of the **unmerge** rules, the moving components (described above as $A \oplus B$) must be split among the two newly predicted items. Illustrative are the gorn addresses of the predicted items. In the case of **unmerge1s** we are assuming that the second argument to the **merge** operation was a specifier (merged on the left – $\mathtt{x}{=}$) and was pronounced there (i.e. it did not later move as it has no further features) – we can therefore conclude that the item representing the second argument to **merge** is the left sister of the item representing the first argument in the derived surface tree.

$$\textbf{unmove1}\quad \frac{\langle \gamma_0, \alpha_0; A[-]\rangle}{\langle \gamma_0 1, {+}\mathtt{x}\alpha; A[\gamma_0 0, {-}\mathtt{x}]\rangle} \qquad \textbf{unmove2}\quad \frac{\langle \gamma_0, \alpha_0; A[\gamma, \alpha]\rangle}{\langle \gamma_0 1, {+}\mathtt{x}\alpha; A[\gamma, {-}\mathtt{x}\alpha]\rangle}$$

In **unmove2**, which corresponds to an application of **move** where the moving expression has not moved its last (i.e. it has movement features left over), the predicted item only updates the gorn address of the head, as the moving item's surface position is not changed by this movement. This contrasts with **unmove1**, which corresponds to an application of **move** where the moving expression moves to its final resting place. Here we know that the moving expression is the left daughter of the head of the popped item, and the head of the predicted item is the right daughter (as movement is to the left).

5 Modeling

Here we report the results of our complexity measures on sentences of the kind in sections 3.1 and 3.2. We begin (in Section 5.1) with an analysis of verb clusters, and show that a simple but linguistically motivated analysis of Dutch and German predicts maximum tenure differences which line up with the behavioural data. Then (in Section 5.2) we consider the stark constrast in English between center embedded and peripherally embedded structures. We provide two syntactic analyses of this phenomenon, which differ from one another only on their analysis of verbal inflection. Somewhat surprisingly, this matters, and highlights the degree to which tenure is dependent upon the particulars of a syntactic analysis.

5.1 Verb Clusters

We assume a verb-raising analysis, depicted in figure 2.[7] On this analysis, verbal complexes in both German and in Dutch have the same deep structure 2(a), one in which verbs and their objects are in a strict sisterhood relation [43]. The different surface word orders arise as a consequence of verbal head movement up and to the left (as shown in 2(c) in the case of German) or to the right (as shown in 3(c) in the case of Dutch). The

[7] This particular analysis is a variation of [31].

only difference between the two grammar fragments is that the verb cluster is formed via *leftward* head movement in German, and via *rightward* head movement in Dutch.

The lexical items in both figures, which are identical but for the direction of head movement (indicated by means of the hyphen attached to the lexeme), are schematic representations of either nominal phrases ('DPs'), sequence initiating verbs (Vi), serial verbs (V) and inflectional heads (I).

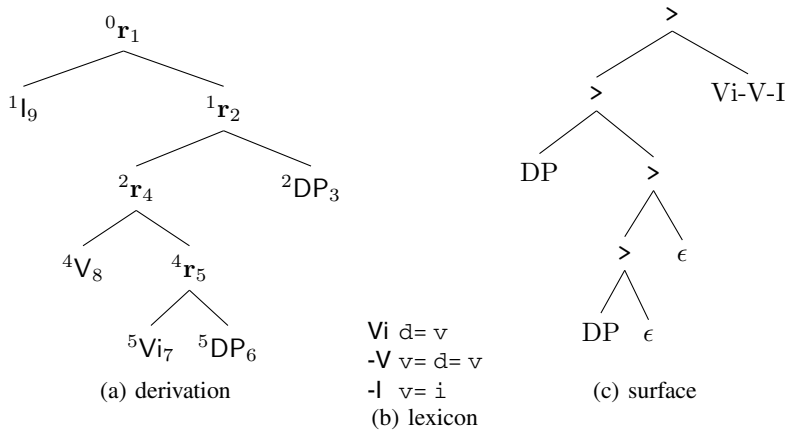


Fig. 2. Structures for German

	Max Tenure	
Embeddings	**German**	**Dutch**
1	8	6
2	12	9
3	16	12
4	20	15

Figures 2 and 3 correspond to the first row in the table above. In the German derivation 2(a) and surface structure 2(c), the maximal tenure is had by the parser item corresponding to the node labelled $^1\mathsf{I}_9$ in the figure, which is predicted at the first step, and which is removed from the queue only at the ninth and last step. The reason why this item is predicted already in the first step is because it is a child of the root/starting item. It is removed from the queue so late because every other parser state contains an item which can be expanded into items which correspond to words in the input which come before this one.

The Dutch deep structure is traversed identically to the German one up until step 7, where, instead of operating on the prediction for a Vi, the prediction of an inflectional element is operated on.

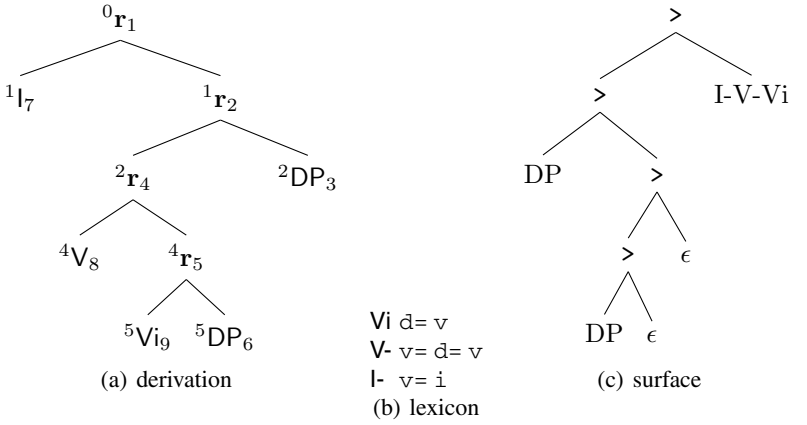


Fig. 3. Structures for Dutch

5.2 English Center and Right Embeddings

We adopt a promotion-style analysis of relative clauses, according to which the relative clause head is an argument of the embedded verb, and then moves to a clause-peripheral position [15,8,17]. We report the results of applying our complexity metric to two grammar fragments which differ in their analyses of verbal inflection. One (4(c)) relies on phrasal movement which conspires to position the verb before the inflectional ending as suggested by [23]. The other (4(a)) uses head movement to build a complex head consisting of a verb and its inflections, an analysis which has its roots in [3].

-s	=v +k s	that	=s +w n	s	=v +z +k s
laugh	=d v	ϵ	=n d -k -w	laugh	=d v -z
-ϵ	=V +k d= v	the	=n d -k	ϵ	=V +k d= v
praise	=d V	boy	n	praise	=d V -z
(a) head		(b) shared		(c) phrasal	

Fig. 4. Lexica

	Head Mvt		Phrasal Mvt	
Depth	**Right**	**Center**	**Right**	**Center**
1	11	23	18	24
2	11	40	36	42
3	11	57	54	60
4	11	74	72	78
5	11	91	90	96
6	11	108	108	114

Under the phrasal movement analyis of inflection, although sentences with right branching embedding have a lower maximal tenure than do those with center embeddings, they have the same rate of growth – in other words, there is no bound on maximal tenure which will correctly rule out (as unacceptable) center embedded sentences of nesting degree greater than (say) 3, but allow peripheral embedding to (much) higher degrees.

Under the head movement analysis of inflection (4(a)), however, sentences with right branching embedding (2) have a lower maximal tenure than those with center embedding (1). Indeed, the maximal tenure of right branching sentences remains constant up to 6 embeddings, whereas the maximal tenure of center embedded structures continues to increase (by seventeen steps) with each additional embedding. Derivation trees for center and peripherally embedded sentences under the head movement analysis of inflection are given in figures 6(a) and 7(a) respectively. In both cases, it is the matrix clause inflectional head (the present tense suffix -*s*) which is the parser item with the maximal tenure. The crucial aspect of the minimalist analysis is that the inflectional head is predicted very early, due to the fact that inflection is assumed to merge with the verb only after all its arguments have been merged with it.

6 General Discussion

Our proposal is that the maximal tenure of an item on the stack reflects a memory requirement that burdens human comprehenders. This is related to different aspects of previous work in psycholinguistics. Since it views a memory cell being occupied as opposed to unoccupied, our proposal can be viewed as a generalization of the HOLD hypothesis [13,42]. One could also view long tenure as an approximation to working memory decay [22].

We have focussed here on off-line difficulty measures. However, various on-line notions of stack tenure are easily derivable, such as maximal tenure up to a particular point in the sentence, or average tenure of stack items. In particular, we derive interesting predictions for on-line judgements in the Dutch versus German data. In figure 5, we report for each of the last six parser steps (steps 13 to 18) of a parse of a sentence with four DPs the maximal tenure of a current stack item, the average tenure of items on the stack, and the sum tenure of all items on the stack. Note that the steps taken by the parser on the Dutch and German sentences are identical up to and including step 13, at which point the verbal cluster begins to be parsed.

6.1 Difficulties with Maximal Tenure

Although simple and well-defined, the notion of maximal tenure (or incremental versions thereof, cf. Section 7) in minimalist grammars is difficult to relate to geometric properties of a derivation. (As, for example, it is possible to relate memory burden in left-corner parsing to the geometric property of center embedding in the context-free parse tree.) Still, a high tenure will obtain whenever an *unmerge* rule introduces derivational sisters, which are separated on the surface by a large number of derived tree leaves. In particular, as pointed out by a reviewer, the top-down minimalist parser should assign high tenure to left-branching structures (as in 5), just as would a top-down context-free parser, which does not seem to accurately reflect the behavioural data.

step	max	average	sum
13	12	6.4	32
14	13	8.5	34
15	14	11	33
16	15	13.5	27
17	16	16	16
18	(empty)		

(a) German

step	max	average	sum
13	12	6.4	32
14	10	6	24
15	8	5.7	17
16	6	5.5	11
17	6	6	6
18	(empty)		

(b) Dutch

$$\max(\mathsf{stack}) = \mathsf{max}_{x \in \mathsf{stack}}\, \mathsf{tenure}(x)$$
$$\text{average}(\mathsf{stack}) = \text{sum}(\mathsf{stack}) / \, |\mathsf{stack}|$$
$$\text{sum}(\mathsf{stack}) = \Sigma_{x \in \mathsf{stack}} \mathsf{tenure}(x)$$

Fig. 5. The parser on the German/Dutch sentence with 3 embeddings

(5) John's neighbor's dentist's uncle laughed.

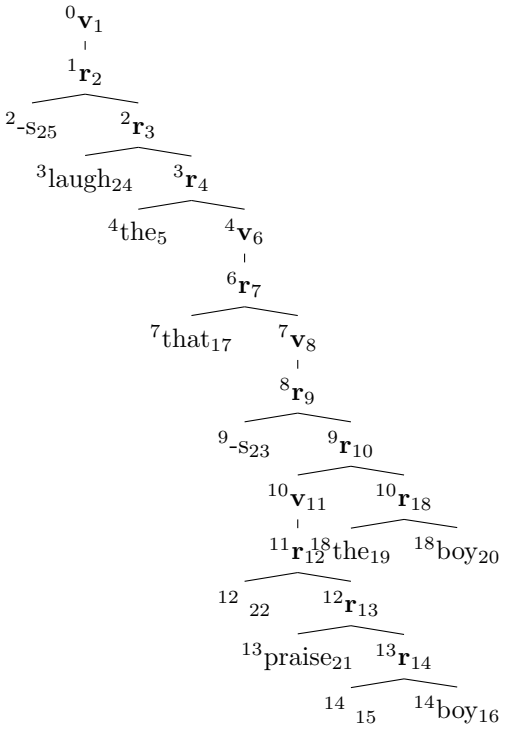
Additionally, the more words which intervene between a *wh*-word and its extraction site, the higher the tenure. Sentence 7 has a higher tenure than 6, despite the fact that the paths between the *wh*-words and their extraction sites are of the same length.

(6) Who did the boy kiss.
(7) Who did the very tall boy kiss.

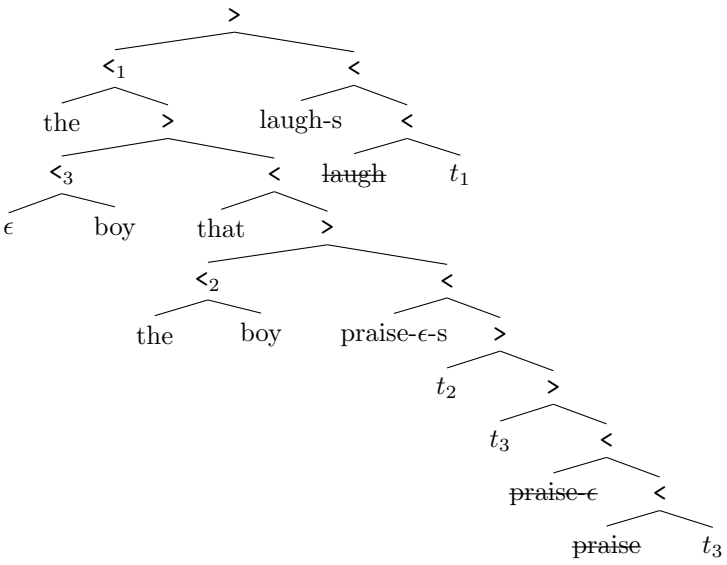
To the extent that this is not reflected in the behavioural data, one potential (but radical) solution could be to change the way tenure is measured.

There are three currently existing proposals in this regard. First, the Dependency Locality Theory [6] can be thought of in the present terms as measuring not the number of steps taken by the parser while a given item is on the stack, but the number of discourse referents introduced. This change would then equate the tenures of 6 and 7, but would still predict a high tenure score for structures like example 5. In the computational literature, Joshi [12] (see also [33]) presents an account of the observed processing contrast between German and Dutch verbal complexes in the context of tree adjoining grammar (TAG). Using the fact that a TAG can be compiled into an equivalent (embedded) push down automaton (EPDA), he notes that the total amount of time the EPDA for the Dutch sentences stores any symbol on its stack is less than that of the EPDA for the German sentences.[8] The complexity measure of 'sum total tenure of stack items' is equivalent to the sum of the sizes of the stack after each step (the sum of the number of symbols on stack one plus the number of symbols on stack two plus ...). However, the notion of a step is taken in [12] to be a scan of an overt word, and intermediate steps are not taken into account. In [33], a step is indeed taken to be a single operation of the automaton, but symbols corresponding to empty nodes in an elementary tree are crucially not counted toward stack size.

[8] He also observes that the maximal stack size is lower for the Dutch EPDA than for the German one.

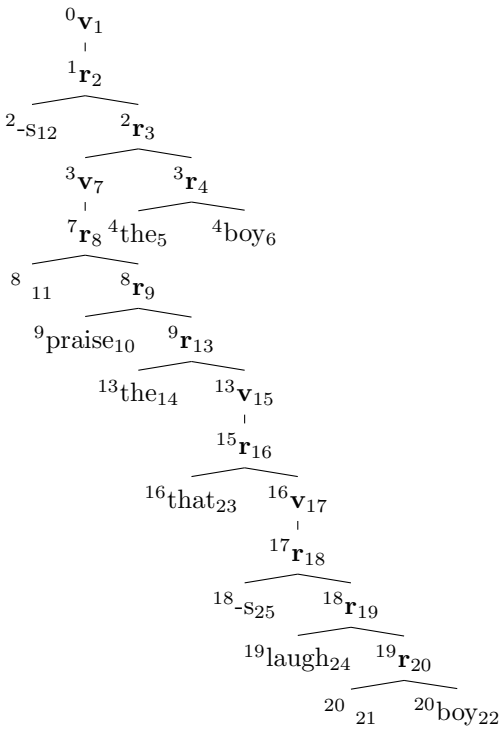


(a) derivation tree

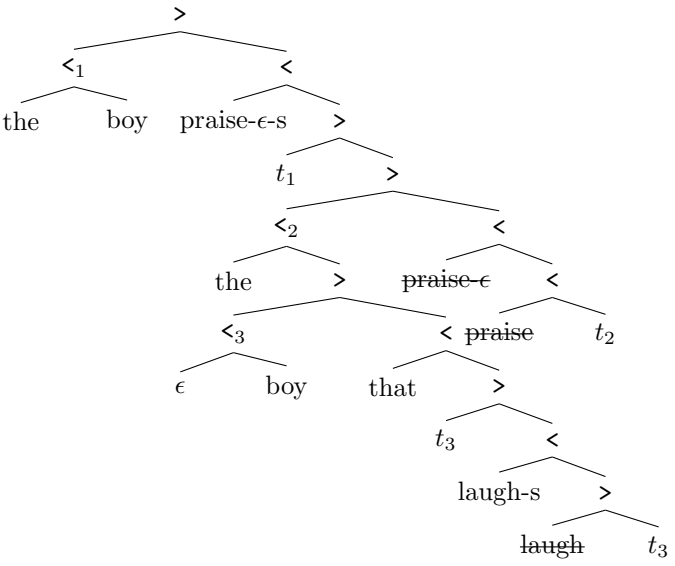


(b) derived tree

Fig. 6. Center Embedding with Head Movement

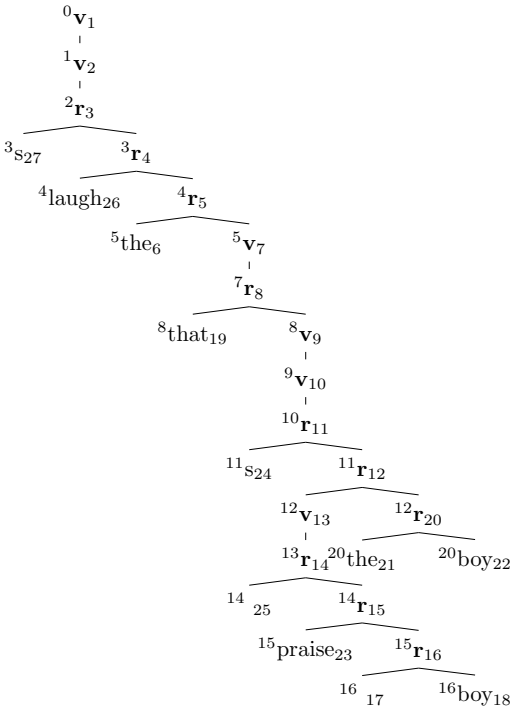


(a) derivation tree

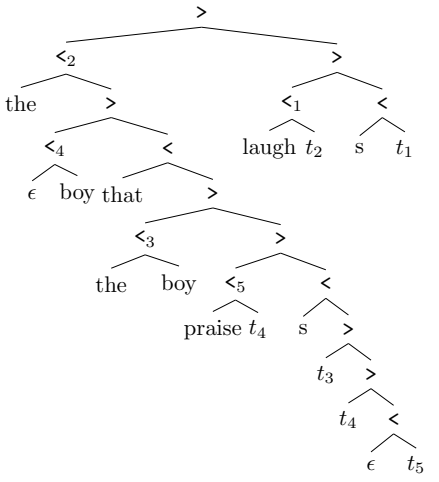


(b) derived tree

Fig. 7. Peripheral Embedding with Head Movement

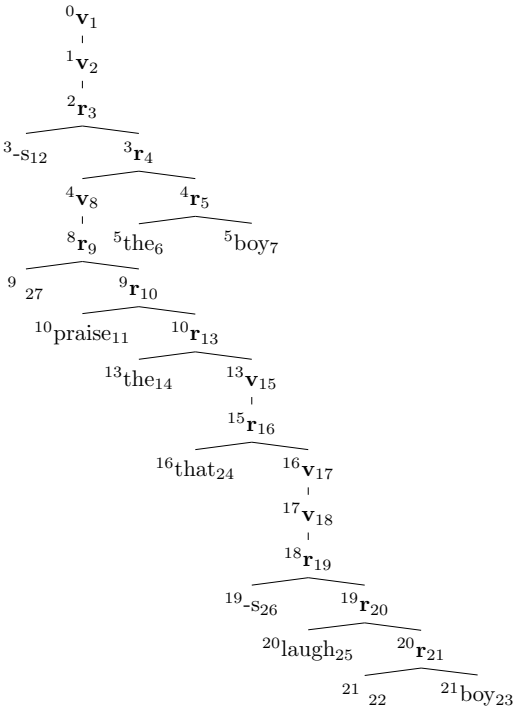


(a) derivation tree

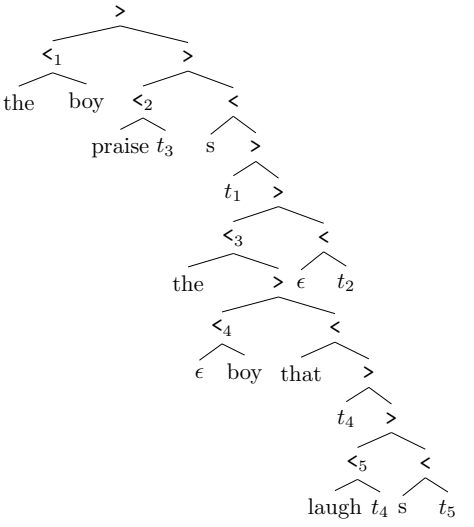


(b) derived tree

Fig. 8. Center Embedding with Phrasal Movement



(a) derivation tree



(b) derived tree

Fig. 9. Peripheral Embedding with Phrasal Movement

7 Conclusion

We have shown that longstanding and influential psycholinguistic ideas about memory resources can be connected with specific and explicit syntactic analyses in rigorous ways. The results of section 5.2 show that the syntactic analysis can indeed play a significant role in the memory requirements of parsing.

The notion of tenure, while useful (at least to a first approximation), is not able to account for all aspects of psycholinguistic data. In particular, we have idealized non-determinism in the parsing process away, while complexity measures which focus on the resolution of non-determinism, such as entropy reduction [8] or surprisal [7], have been demonstrated to have explanatory value [21,41]. Furthermore, although tenure is a measure related to memory burden, we have made very weak assumptions about the nature of memory — incorporating psychological insights into the nature and limitations of human memory [22] may allow a reductive explanation of our tenure measure, or at least for a more refined and nuanced theory.

Future work will investigate how best to relate tenure to online data, how to integrate various independent notions of psycholinguistic 'difficulty', and how to most appropriately account for examples like those presented in Section 6.1.

References

1. Abney, S., Johnson, M.: Memory requirements and local ambiguities of parsing strategies. Journal of Psycholinguistic Research 20(3), 233–249 (1991)
2. Bach, E., Brown, C., Marslen-Wilson, W.: Crossed and nested dependencies in German and Dutch: A psycholinguistic study. Language and Cognitive Processes 1(4), 249–262 (1986)
3. Chomsky, N.: Syntactic Structures. Mouton, The Hague (1957)
4. Chomsky, N.: The Minimalist Program. MIT Press, Cambridge (1995)
5. Crocker, M.W., Brants, T.: Wide-coverage probabilistic sentence processing. Journal of Psycholinguistic Research 29(6), 647–669 (2000)
6. Gibson, E.: The dependency locality theory: A distance-based theory of linguistic complexity. In: Miyashita, Y., Marantz, A., O'Neil, W. (eds.) Image, Language, Brain, pp. 95–126. MIT Press, Cambridge (2000)
7. Hale, J.: A Probabilistic Earley Parser as a Psycholinguistic Model. In: Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics (2001)
8. Hale, J.T.: Grammar, Uncertainty and Sentence Processing. Ph.D. thesis, The Johns Hopkins University (2003)
9. Hale, J.T.: Uncertainty about the rest of the sentence. Cognitive Science 30, 643–672 (2006)
10. Harkema, H.: Parsing Minimalist Languages. Ph.D. thesis, University of California, Los Angeles (2001)
11. Johnson-Laird, P.N.: Mental Models. Cambridge University Press (1983)
12. Joshi, A.K.: Processing crossed and nested dependencies: An automation perspective on the psycholinguistic results. Language and Cognitive Processes 5(1), 1–27 (1990)
13. Kaplan, R.M.: Transient Processing Load in Relative Clauses. Ph.D. thesis, Harvard (1975)
14. Kay, M.: Algorithm schemata and data structures in syntactic processing. In: Grosz, B.J., Jones, K.S., Webber, B.L. (eds.) Readings in Natural Language Processing. Morgan Kaufman (1986)

15. Kayne, R.: The Antisymmetry of Syntax. MIT Press, Cambridge (1994)
16. Kobele, G.M.: Formalizing mirror theory. Grammars 5(3), 177–221 (2002)
17. Kobele, G.M.: Generating Copies: An investigation into structural identity in language and grammar. Ph.D. thesis, University of California, Los Angeles (2006)
18. Kobele, G.M.: Importing montagovian dynamics into minimalism. In: Béchet, D., Dikovsky, A. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 7351, pp. 103–118. Springer, Heidelberg (2012)
19. Kobele, G.M., Retoré, C., Salvati, S.: An automata theoretic approach to minimalism. In: Rogers, J., Kepser, S. (eds.) Proceedings of the Workshop Model-Theoretic Syntax at 10; ESSLLI 2007, Dublin (2007)
20. Kowalski, R.: Algorithm = logic + control. Communications of the ACM 22(7), 424–436 (1979)
21. Levy, R.: Expectation-based syntactic comprehension. Cognition 106, 1126–1177 (2008)
22. Lewis, R.L., Vasishth, S.: An activation-based model of sentence processing as skilled memory retrieval. Cognitive Science 29, 375–419 (2005)
23. Mahajan, A.: Word order and (remnant) VP movement. In: Karimi, S. (ed.) Word Order and Scrambling, ch. 10. Blackwell (2003)
24. Mainguy, T.: A probabilistic top-down parser for minimalist grammars. CoRR abs/1010.1826 (2010)
25. Marr, D.: Vision. W. H. Freeman and Company, New York (1982)
26. Marslen-Wilson, W.: Linguistic structure and speech shadowing at very short latencies. Nature 244, 522–523 (1973)
27. Michaelis, J.: On Formal Properties of Minimalist Grammars. Ph.D. thesis, Universität Potsdam (2001)
28. Miller, G.A., Chomsky, N.: Finitary models of language users. In: Luce, R.D., Bush, R.R., Galanter, E. (eds.) Handbook of Mathematical Psychology, ch. 13, pp. 419–491. John Wiley, New York (1963)
29. Mönnich, U.: Minimalist syntax, multiple regular tree grammars and direction preserving tree transductions. In: Rogers, J., Kepser, S. (eds.) Proceedings of the Workshop Model-Theoretic Syntax at 10; ESSLLI 2007, Dublin (2007)
30. Montague, R.: The proper treatment of quantification in ordinary English. In: Hintikka, J., Moravcsik, J., Suppes, P. (eds.) Approaches to Natural Language, pp. 221–242. D. Reidel, Dordrecht (1973)
31. Morawietz, F.: Two-Step Approaches to Natural Language Formalisms, Studies in Generative Grammar, vol. 64. Mouton de Gruyter (2003)
32. Rambow, O., Joshi, A.K.: A processing model for free word order languages. In: Clifton, C., Frazier, L., Rayner, K. (eds.) Perspectives on Sentence Processing, pp. 267–301. Lawrence Erlbaum (1994)
33. Rambow, O., Joshi, A.K.: A processing model for free word order languages. Tech. Rep. IRCS-95-13, University of Pennsylvania (1995)
34. Resnik, P.: Left-corner parsing and psychological plausibility. In: Proceedings of the Fourteenth International Conference on Computational Linguistics, Nantes, France (1992)
35. Shieber, S.M.: Evidence against the context-freeness of natural language. Linguistics and Philosophy 8, 333–343 (1985)
36. Stabler, E.: Top-down recognizers for MCFGs and MGs. In: Proceedings of CMCL (2011)
37. Stabler, E.P.: Derivational minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)
38. Stabler, E.P., Keenan, E.L.: Structural similarity within and among languages. Theoretical Computer Science 293, 345–363 (2003)
39. Tanenhaus, M., Spivey-Knowlton, M., Eberhard, K., Sedivy, J.: Integration of visual and linguistic information in spoken language comprehension. Science 268, 1632–1634 (1995)

40. VanWagenen, S., Brennan, J., Stabler, E.P.: Evaluating parsing strategies in sentence processing. Poster Presented at CUNY 2011 (2011)
41. Vasishth, S., Drenhaus, H.: Locality in German. Dialogue and Discourse 1(2), 59–82 (2011)
42. Wanner, E., Maratsos, M.: An ATN approach to comprehension. In: Halle, M., Bresnan, J., Miller, G.A. (eds.) Linguistic Theory and Psychological Reality, ch. 3, pp. 119–161. MIT Press, Cambridge (1978)
43. Wurmbrand, S.: How complex are complex predicates. Syntax 10(3), 243–288 (2007)

Parsing Pregroup Grammars with Letter Promotions in Polynomial Time

Katarzyna Moroz

Faculty of Mathematics and Computer Science,
Adam Mickiewicz University, Poznań
moroz@amu.edu.pl

Abstract. We consider pregroup grammars with letter promotions of the form $p^{(m)} \Rightarrow q^{(n)}$, $p \Rightarrow 1, 1 \Rightarrow q$. We prove a variant of Lambek's normalization theorem [5] for the calculus of pregroups enriched with such promotions and present a polynomial parsing algorithm for the corresponding pregroup grammars. The algorithm extends that from [8], elaborated for pregroup grammars without letter promotions. The normalization theorem, restricted to letter promotions without 1, was proved in [3,4] while the present version was stated in [4] without proof and used to show that the word problem for letter promotions with unit is polynomial. Our results are contained in the unpublished PhD thesis [9].

1 Introduction and Preliminaries

The paper continues and extends some results of [4] and [8]. [4] considered pregroups and pregroup grammars with letter promotions and with letter promotions with unit. A Lambek-style normalization theorem for pregroups with letter promotions is proved. A similar theorem for pregroups with letter promotions with unit is stated and here we give the proof of the theorem. The proof of the normalization theorem for letter promotions with 1 essentially refines that from [3,4], but does not follow directly from them, since one must handle new contraction and expansion steps. In [4] it is also proved that the word problem and the membership problem for pregroups with letter promotions can be solved in polynomial time. Similar results are given for pregroups with letter promotions with unit. In [8] we propose a polynomial dynamic parsing algorithm for pregroup grammars and give the proof of its correctness. In this paper we show that the algorithm can be modified to work for pregroup grammars with letter promotions with unit. The results were stated in an unpublished PhD thesis [9].

Pregroups were introduced by Lambek [5] as an algebraic tool for the syntactical analysis of sentences. Pregroup grammars belong to lexical grammars since most of linguistic information is encoded in the lexicon. Similarly as in Lambek categorial grammars, syntactical properties of words are described by a finite set of pregroup types. However, the structure of types is different and so is the logic. Pregroup types are elements of a free monoid, generated by iterated adjoints of some atoms, and they are processed using a calculus of free pregroups,

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 52–68, 2013.

also called Compact Bilinear Logic **CBL**. The computational complexity of pregroup grammars is polynomial in contrast to the computational complexity of categorial grammars. Pregroups and pregroup grammars have been successfully applied to parsing diverse natural languages like English, Italian, Polish and Japanese.

However, not all natural language phenomena can be easily described by the formalism of pregroups. Therefore some extensions to pregroup grammars have been proposed. Mater and Fix [7] consider pregroup grammars enriched with some more general assumptions. An interesting problem they propose is called a *letter promotion problem for pregroups*. Their idea is further developed in [3], [4]. Buszkowski and Lin [3] show that the word problem for pregroups with letter promotions, that is assumptions of the form $p^{(m)} \Rightarrow q^{(n)}$, is polynomial when the size of $p^{(n)}$ is counted as $|n|+1$. [4] extends this result for letter promotions with unit $p^{(m)} \Rightarrow 1$, $1 \Rightarrow q^{(n)}$.

CBL enriched with some assumptions is interesting for a few reasons. First of all, we look for extensions to pregroups allowing easier description of some natural language phenomena, see [6]. Moreover, we know that the calculus of pregroups is solvable in polynomial time (see [1]), but the associative Lambek calculus is NP-complete. The calculi admitting letter promotions and letter promotions with unit are still polynomial. It is interesting how far the calculus of pregroups can be generalized while remaining polynomial. Finally, pregroup grammars with letter promotions can directly simulate any cancelation grammar. Cancelation grammars are defined by Buszkowski in [4].

Definition 1. *A* cancelation grammar *is a tuple $G = (\Sigma, V, X, R, I)$, where Σ and V are finite disjoint alphabets (terminal and auxiliary respectively), $X \in V^*$, R is a finite set of cancelation rules and I assigns a finite set of strings over V to any element of Σ. The cancelation rules are of the form $A, B \Rightarrow \varepsilon$ and $A \Rightarrow B$, for $A, B \in V$. G assigns a string $Y \in V^*$ to a string $a_1...a_n$ of elements from Σ if there exist strings $Y_1...Y_n$ such that $Y_i = I(a_i)$, $i = 1, ..., n$ and $Y_1...Y_n$ reduces to Y by a finite number of applications of rules from R.*

The *language of a cancelation grammar* consists of all strings on Σ^+ which are assigned the designated type X by G. Pregroup grammars (also with letter promotions) are a special kind of cancelation grammars. Conversely, every cancelation rule $A, B \Rightarrow \varepsilon$ can directly be simulated by the letter promotion $A \Rightarrow B^l$, i.e. $A \Rightarrow B^{(-1)}$.

Definition 2. *A* pregroup *is an algebra $(M, \leqslant, \cdot, l, r, 1)$, such that $(M, \leqslant, \cdot, 1)$ is a partially ordered monoid and l, r are unary operations on M satisfying* adjoint laws*:*

$$(Al)\ a^l a \leqslant 1 \leqslant a a^l$$

$$(Ar)\ a a^r \leqslant 1 \leqslant a^r a,$$

for all $a \in M$.

The elements a^l and a^r are called *left* and *right adjoint*, respectively. Let us notice that left and right adjoints are unique for each element of M.

One defines *iterated adjoints* $a^{(n)} = a^{rr\ldots r}$ and $a^{(-n)} = a^{ll\ldots l}$, where n is a non-negative integer, ^{r} and ^{l} are iterated n times and a is any element of a pregroup M. By definition $a^{(0)} = a$.

To apply pregroups to parsing natural languages Lambek uses the notion of a *free pregroup* generated by a poset. Let us assume $(P, \leqslant)$ is a nonempty, finite poset. Then elements of P are called *atoms* and they are denoted by letters p, q, r. Expressions of the form $p^{(n)}$, where $p \in P$ and n is any integer, are called *terms* and denoted by t, u. Finite strings of terms are called *types* and denoted by X, Y, Z. Types are assigned to words in the lexicon and they refer to the role the given word takes in a sentence. Usually, one word can be assigned many different types.

The following rules define a binary relation $\Rightarrow$ on the set of types:

(CON) $X, p^{(n)}, p^{(n+1)}, Y \Rightarrow X, Y$,
(EXP) $X, Y \Rightarrow X, p^{(n+1)}, p^{(n)}, Y$,
(POS) $X, p^{(n)}, Y \Rightarrow X, q^{(n)}, Y$, if $p \leqslant q$ and n is even or $q \leqslant p$ and n is odd.

(CON), (EXP), (POS) are called Contraction, Expansion and Poset rules, respectively. Poset rules were originally called Induced Step (IND) by Lambek [5]. Actually, $\Rightarrow$ is a reflexive and transitive closure of the relation defined by these clauses. Let us notice that $X \Rightarrow Y$ is true iff X can be rewritten into Y by a finite number of applications of these rules. This rewriting system is Lambek's original form of the logic of pregroups, which is also called ***CBL***.

Additionaly, one defines a useful rule called *Generalized Contraction*, which combines (CON) and (POS) and similarly Generalized Expansion (GEXP):

(GCON) $X, p^{(n)}, q^{(n+1)}, Y \Rightarrow X, Y$,
(GEXP) $X, Y \Rightarrow X, p^{(n+1)}, q^{(n)} Y$,

where in both cases $p \leqslant q$ and n is even, or $q \leqslant p$ and n is odd.

Lambek proves a normalization theorem for **CBL** [5]:

> if $X \Rightarrow Y$ in **CBL**, then there exist Z and U such that $X \Rightarrow Z$ by applying (GCON) only, $Z \Rightarrow U$ by applying (POS) only, and $U \Rightarrow Y$ by applying (GEXP) only.

Therefore, if Y is a term or $Y \Rightarrow \varepsilon$, then $X \Rightarrow Y$ in **CBL** if and only if X can be reduced to Y without (GEXP), that is using (CON) and (POS) only.

It is useful to define X^r and X^l for any type X:

$$\varepsilon^l = \varepsilon = \varepsilon^r;$$
$$X^r = (p_1^{(n_1)} \ldots p_k^{(n_k)})^r = p_k^{(n_k+1)} \ldots p_1^{(n_1+1)},$$
$$X^l = (p_1^{(n_1)} \ldots p_k^{(n_k)})^l = p_k^{(n_k-1)} \ldots p_1^{(n_1-1)},$$

where $n_1, \ldots, n_m$ are arbitrary integers.

Definition 3. *A* pregroup grammar *is a structure $G = (\Sigma, P, I, s, R)$ where Σ is a finite alphabet (that is a lexicon), P is a finite set of atoms, I is a finite relation assigning types on P to symbols from Σ, s is the denoted type s, and R is a partial ordering on P.*

If R is fixed, one writes $p \leqslant r$ for pRq. For $a \in \Sigma,\ I(a)$ denotes the set of all types X such that $(a, X) \in I$. Let us assume $x \in \Sigma^+,\ x = a_1...a_n\ (a_i \in \Sigma)$. One says that the grammar G *assigns* type Y to x if there exist types $X_i \in I(a_i)$, $i = 1, ..., n$, such that $X_1, ..., X_n \Rightarrow Y$ in **CBL**; we write $x \rightarrow_G Y$. Then *the language of a pregroup grammar* G, denoted by $L(G)$, consists of all strings $x \in \Sigma^+$ to which the grammar G assigns the denoted type s. Due to the normalization theorem while parsing pregroup grammars, one may restrict the rules to only (CON) and (POS).

To describe a pregroup grammar for a given language one has to define a set of terms, fix a lexicon consisting of words of the language and types assigned to them and define a partial order on types, that is, fix the assumptions. One can say that assumptions express different forms of subtyping. Let us consider a few examples. If *he* is assigned type π_3 (subject in third person), *likes* - $\pi_3^r\ s_1\ o^l$ and *books* - n_2 (plural noun), then the sentence *He likes books.* can be parsed as follows: $(\pi_3)(\pi_3^r\ s_1\ o^l)(n_2) \leqslant s_1 o^l n_2 \leqslant s_1 o^l o \leqslant s_1$ (using the assumption $n_2 \leqslant o$, as a plural noun can take part of an object in a sentence). The string is assigned type s_1 i.e. the type of statement in present tense and by the assumption $s_1 \leqslant s$, it is a statement (type s). Clearly, the types assigned above are not unique. For example *books* is also of type $\pi_3^r s_1 o^l$ (a transitive verb in third person, present tense). The above reduction can be depicted by the following links:

He *likes* *books.*
$\underbrace{(\pi_3)(\pi_3^r}\ s_1\ \underbrace{o^l)(n_2)}$

Let us consider a complete system of **CBL** with letter promotions obtained by modifying (POS) to Promotion Rules (PRO):

(PRO) $X, p^{(m+k)}, Y \Rightarrow X, q^{(n+k)}, Y$ if either k is even and $p^{(m)} \Rightarrow q^{(n)}$ is an assumption, or k is odd and $q^{(n)} \Rightarrow p^{(m)}$ is an assumption.

A pregroup grammar with letter promotions is defined similarly to a pregroup grammar.

Definition 4. *A* pregroup grammar with letter promotions *is a pregroup grammar* $G = (\Sigma, P, I, s, R)$ *in which* R *is the set of assumptions extended by a set of letter promotions. We require that* $P(R) \subseteq P$, *where* $P(R)$ *denotes the set of atoms appearing in assumptions from* R.

By $R \vdash_{\mathbf{CBL}} X \Rightarrow Y$ we mean that X can be transformed into Y by a finite number of applications of (CON), (EXP) and (PRO), restricted to the assumption from a set of letter promotions R. (POS) is treated as an instance of (PRO).

Assuming that $t \Rightarrow u$ is an instance of (PRO) restricted to the assumptions from R, that is X, Y are empty, we write $t \Rightarrow_R u$. We write $t \Rightarrow_R^* u$ if there exist terms $t_0, \ldots, t_k$ such that $k \geqslant 0,\ t_0 = t,\ t_k = u$, and $t_{i-1} \Rightarrow_R t_i$, for all $i = 1, \ldots, k$. Hence, $\Rightarrow_R^*$ is the reflexive and transitive closure of $\Rightarrow_R$.

For **CBL** with letter promotions one defines a generalization of the rules (CON) and (EXP), which are derivable in **CBL** with assumptions from R.

(GCON-R) $X, p^{(m)}, q^{(n+1)}, Y \Rightarrow X, Y$ if $p^{(m)} \Rightarrow_R^* q^{(n)}$,
(GEXP-R) $X, Y \Rightarrow X, p^{(m+1)}, q^{(n)}, Y$ if $p^{(m)} \Rightarrow_R^* q^{(n)}$.

Clearly, (CON) is a special instance of (GCON-R) while (EXP) is a special instance of (GEXP-R). One can treat any iteration of (PRO) as a single step.

(PRO-R) $X, t, Y \Rightarrow X, u, Y$ if $t \Rightarrow_R^* u$.

As a consequence of the normalization theorem, we get $R \vdash_{\mathbf{CBL}} t \Rightarrow u$ iff $t \Rightarrow_R^* u$; see [3,4].

Definition 5. *The* letter promotion problem for pregroups **(LPPP)** *is stated as follows: for the given finite set R of letter promotions, and terms t, u, verify whether $t \Rightarrow u$ in **CBL** enriched with all promotions from R as assumptions.*

Shortly, **(LPPP)** consists of verifying whether $t \Rightarrow_R^* u$ for given R, t, u.

Buszkowski and Lin [3] prove that **LPPP** is polynomial, provided that the size of $p^{(n)}$ is counted as $|n| + 1$ (it is natural, since $p^{(n)}$ abbreviates the n-th iteration of l's or r's). This count is used in the present paper. [4] extends this result for letter promotions with unit and shows that the membership problem for the corresponding pregroup grammars is polynomial, and these grammars are equivalent to CFGs (but the latter does not directly imply the polynomiality of the membership problem; see [2] for discussion).

The main result of the present paper is a polynomial parsing algorithm for pregroup grammars admitting letter promotions of the form $p^{(m)} \Rightarrow q^{(n)}$, $p \Rightarrow 1$, $1 \Rightarrow q$. This algorithm refines an earlier one for pregroup grammars [8]; the latter adapts a method of Savateev [11], elaborated for Unidirectional Lambek Calculus. Section 3 presents our algorithm, the proof of its correctness and a related algorithm returning the reduction. Since a Lambek-style normalization theorem is essentially used, we give its full proof in section 2 (the theorem was proved in [9] and stated without proof in [4]).

2 The Normalization Theorem

Letter promotions with unit are promotions allowing 1, that is letter promotions of the form: $p^{(m)} \Rightarrow 1$ or $1 \Rightarrow q^{(n)}$. We add 1 to the set of terms. Notice that in pregroups the assumption $p^{(m)} \Rightarrow 1$ is equivalent to $p \Rightarrow 1$ if m is even and to $1 \Rightarrow p$ if m is odd. Similarly, the assumption $1 \Rightarrow p^{(m)}$ is equivalent to $1 \Rightarrow p$ if m is even and to $p \Rightarrow 1$ if m is odd. Therefore, in what follows, we consider letter promotions with unit only of the form $p \Rightarrow 1$ and $1 \Rightarrow q$.

A complete system of **CBL** with letter promotions with unit is obtained by adding two new rules to **CBL** with letter promotions.

Definition 6. *A* complete system of **CBL** with letter promotions with unit *consists of the following rules:*

(CON) $X, p^{(n)}, p^{(n+1)}, Y \Rightarrow X, Y$,
(EXP) $X, Y \Rightarrow X, p^{(n+1)}, p^{(n)}, Y$,
(PRO) $X, p^{(m+k)}, Y \Rightarrow X, q^{(n+k)}, Y$, *if either k is even and $p^{(m)} \Rightarrow q^{(n)}$ is an assumption, or k is odd and $q^{(n)} \Rightarrow p^{(m)}$ is an assumption.*

(PRO-C) $X, p^{(m)}, Y \Rightarrow X, Y$, *if either* m *is even and* $p \Rightarrow 1$ *is an assumption, or* m *is odd and* $1 \Rightarrow p$ *is an assumption,*
(PRO-E) $X, Y \Rightarrow X, q^{(n)}, Y$, *if either* n *is even and* $1 \Rightarrow q$ *is an assumption, or* n *is odd and* $q \Rightarrow 1$ *is an assumption.*

Consequently, (PRO-C) is a contracting promotion step, (PRO-E) is an expanding promotion step and (PRO) is a neutral promotion step.

We fix a finite set $R1$ of letter promotions, possibly with unit. We write $t \Rightarrow_{R1} u$ if $t \Rightarrow u$ is an instance of (PRO), (PRO-C) or (PRO-E) restricted to the set of assumptions $R1$ (X, Y are empty). We write $t \Rightarrow^*_{R1} u$ if there exist terms $t_0, ..., t_k$ such that $k \geqslant 0,\ t_0 = t,\ t_k = u,\ t_{i-1} \Rightarrow_{R1} t_i$, for all $i = 1, ..., k$.

We introduce some new rules derivable in **CBL** with assumptions from $R1$.

(GCON-R1) $X, p^{(m)}, q^{(n+1)}, Y \Rightarrow X, Y$, if $p^{(m)} \Rightarrow^*_{R1} q^{(n)}$
(GEXP-R1) $X, Y \Rightarrow X, p^{(n+1)}, q^{(m)}, Y$, if $p^{(n)} \Rightarrow^*_{R1} q^{(m)}$
(PRO-R1) $X, t, Y \Rightarrow X, u, Y$ if $t \Rightarrow^*_{R1} u$ and $t \neq 1$ and $u \neq 1$
(PRO-C-R1) $X, t, Y \Rightarrow X, Y$ if $t \Rightarrow^*_{R1} 1$ and $t \neq 1$
(PRO-E-R1) $X, Y \Rightarrow X, u, Y$ if $1 \Rightarrow^*_{R1} u$ and $u \neq 1$

There holds a normalization theorem for **CBL** with letter promotions with unit:

Theorem 1. *If* $R1 \vdash_{\mathbf{CBL}} X \Rightarrow Y$, *then there exist* Z, U *such that* $X \Rightarrow Z$ *by a finite number of instances of (GCON-R1) and (PRO-C-R1),* $Z \Rightarrow U$ *by a finite number of instances of (PRO-R1) and* $U \Rightarrow Y$ *by a finite number of instances of (GEXP-R1) and (PRO-E-R1).*

Proof. Let us start with some notions. A sequence $X_0, \ldots, X_k$ such that $X = X_0,\ Y = X_k$ and, for any $i = 1, \ldots, k,\ X_{i-1} \Rightarrow X_i$ is an instance of (GCON-R1), (GEXP-R1), (PRO-E-R1), (PRO-C-R1) or (PRO-R1) is called a *derivation of* $X \Rightarrow Y$ *from the set of assumptions R1.* Clearly, $R1 \vdash X \Rightarrow Y$ iff there exists a derivation of $X \Rightarrow Y$ of this form. k is the length of the derivation. If a derivation has a form required by the Theorem, then it is called a *normal derivation.*

We prove that every derivation $X_0, \ldots, X_k$ of $X \Rightarrow Y$ can be transformed into a normal derivation of length not greater than k. We proceed by induction on k.

We should notice that for $k = 0$ and $k = 1$ the initial derivation is normal. For $k = 0$, it suffices to take $X = Z = U = Y$. For $k = 1$, if $X \Rightarrow Y$ is an instance of (GCON-R1) or (PRO-C-R1), one takes $Z = U = Y$, if $X \Rightarrow Y$ is an instance of (GEXP-R1) or (PRO-E-R1), one takes $X = Z = U$, and if $X \Rightarrow Y$ is an instance of (PRO-R1), one takes $X = Z$ and $U = Y$.

Assume now $k > 1$. The derivation $X_1, \ldots, X_k$ is shorter, whence it can be transformed into a normal derivation $Y_1, \ldots, Y_l$ such that $X_1 = Y_1,\ X_k = Y_l$ and $l \leqslant k$. If $l < k$, then $X_0, Y_1, \ldots, Y_l$ is a derivation of $X \Rightarrow Y$ of length less than k, whence it can be transformed into a normal derivation, by the induction hypothesis. So assume $l = k$.

Case 1. $X_0 \Rightarrow X_1$ is an instance of (GCON-R1). Then $X_0, Y_1, \ldots., Y_l$ is a normal derivation of $X \Rightarrow Y$ from $R1$.

Case 2. $X_0 \Rightarrow X_1$ is an instance of (PRO-C-R1). Then $X_0, Y_1, \ldots, Y_l$ is a normal derivation of $X \Rightarrow Y$ from $R1$.

Case 3. $X_0 \Rightarrow X_1$ is an instance of (GEXP-R1), assume $X_0 = UV$; $X_1 = Up^{(n+1)}q^{(m)}V$, and $p^{(n)} \Rightarrow^*_{R1} q^{(m)}$. We consider two subcases.

Case 3.1. Neither any (GCON-R1)-step nor any (PRO-C-R1)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrences of $p^{(n+1)}, q^{(m)}$. If also no (PRO-R1)-step of $Y_1, \ldots, Y_l$ acts on these designated terms, then we drop $p^{(n+1)}q^{(m)}$ from all types appearing in (GCON-R1)-steps, (PRO-C-R1)-steps and (PRO-R1)-steps of $Y_1, \ldots, Y_l$, then we introduce $p^{(n+1)}q^{(m)}$ by a single instance of (GEXP-R1), and continue the (GEXP-R1)-steps and (PRO-E-R1)-steps of $Y_1, \ldots, Y_l$; this yields a normal derivation of $X \Rightarrow Y$ of length k. Otherwise, let $Y_{i-1} \Rightarrow Y_i$ be the first (PRO-R1)-step of $Y_1, \ldots, Y_l$ which acts on $p^{(n+1)}$ or $q^{(m)}$.

(I) If $Y_{i-1} \Rightarrow Y_i$ acts on $p^{(n+1)}$, then there exists a term $r^{(m')}$ and types T, W such that $Y_{i-1} = Tp^{(n+1)}W$, $Y_i = Tr^{(m')}W$ and $p^{(n+1)} \Rightarrow^*_{R1} r^{(m')}$. Consequently, $r^{(m'-1)} \Rightarrow^*_{R1} p^{(n)}$, whence $r^{(m'-1)} \Rightarrow^*_{R1} q^{(m)}$. Then we can replace the derivation $X_0, Y_1, \ldots, Y_l$ by a shorter derivation: first apply (GEXP-R1) of the form $U, V \Rightarrow U, r^{(m')}, q^{(m)}, V$, then derive $Y_1, \ldots, Y_{i-1}$ in which $p^{(n+1)}$ is replaced by $r^{(m')}$, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. By the induction hypothesis, this derivation can be transformed into a normal derivation of length less than k.

(II) If $Y_{i-1} \Rightarrow Y_i$ acts on $q^{(m)}$, then there exist a term $r^{(m')}$ and types T, W such that $Y_{i-1} = Tq^{(m)}W$, $Y_i = Tr^{(m')}W$ and $q^{(m)} \Rightarrow^*_{R1} r^{(m')}$. Consequently, $p^{(n)} \Rightarrow^*_{R1} r^{(m')}$, and we can replace the derivation $X_0, Y_1, ldots, Y_l$ by a shorter derivation: first apply (GEXP-R1) of the form $U, V \Rightarrow U, p^{(n+1)}, r^{(m')}, V$, then derive $Y_1, \ldots, Y_{i-1}$ in which $q^{(m)}$ is replaced by $r^{(m')}$, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. Again we apply the induction hypothesis.

Case 3.2. Some (GCON-R1)-step of $Y_1, \ldots, Y_l$ acts on (some of) the designated occurrences of $p^{(n+1)}, q^{(m)}$. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. There are three possibilities.

(I) This step acts on both $p^{(n+1)}$ and $q^{(m)}$. Then, the derivation $X_0, Y_1, \ldots, Y_l$ can be replaced by a shorter derivation: drop the first application of (GEXP-R1), then derive $Y_1, \ldots, Y_{i-1}$ in which $p^{(n+1)}q^{(m)}$ is omitted, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. We apply the induction hypothesis.

(II) This step acts on $p^{(n+1)}$ only. Then, $Y_{i-1} = Tr^{(m')}p^{(n+1)}q^{(m)}W$, $Y_i = Tq^{(m)}W$ and $r^{(m')} \Rightarrow^*_{R1} p^{(n)}$. The derivation $X_0, Y_1, \ldots, Y_l$ can be replaced by a shorter, normal derivation: drop the first application of (GEXP-R1), then derive $Y_1, \ldots, Y_{i-1}$ in which $p^{(n+1)}q^{(m)}$ is omitted, derive Y_i by a (PRO-R1)-step (notice $r^{(m')} \Rightarrow^*_{R1} q^{(m)}$), and continue $Y_{i+1}, \ldots, Y_l$.

(III) This step acts on $q^{(m)}$ only. Then, $Y_{i-1} = Tp^{(n+1)}q^{(m)}r^{(m'+1)}W$, $Y_i = Tp^{(n+1)}W$ and $q^{(m)} \Rightarrow^*_{R1} r^{(m')}$. The derivation $X_0, Y_1, \ldots, Y_l$ can be replaced by a shorter derivation: drop the first application of (GEXP-R1), then derive $Y_1, \ldots, Y_{i-1}$ in which $p^{(n+1)}q^{(m)}$ is omitted, derive Y_i by

a (PRO-R1)-step (notice $r^{(m'+1)} \Rightarrow^*_{R1} p^{(n+1)}$), and continue $Y_{i+1}, \ldots, Y_l$. We apply the induction hypothesis.

Case 3.3. Some (PRO-C-R1)-step of $Y_1, \ldots, Y_l$ acts on (some of) the designated occurrences of $p^{(n+1)}, q^{(m)}$. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. There are two possibilities.

(I) This step acts on $p^{(n+1)}$. Then $Y_{i-1} = Tp^{(n+1)}W$, $Y_i = TW$ and $p^{(n+1)} \Rightarrow^*_{R1} 1$. Thus, $1 \Rightarrow^*_{R1} p^{(n)}$, and hence $1 \Rightarrow^*_{R1} q^{(m)}$. We can replace the derivation $X_0, Y1, \ldots, Y_l$ by a shorter derivation: start with an application of (PRO-E-R1) of the form $UV \Rightarrow Uq^{(m)}V$ derive $Y_1, \ldots, Y_{i-1}$ in which $p^{(n+1)}$ is replaced by 1, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. Again we apply the induction hypothesis.

(II) This step acts on $q^{(m)}$. Then, $Y_{i-1} = Tq^{(m)}W$, $Y_i = TW$ and $q^{(m)} \Rightarrow^*_{R1} 1$. Thus, $p^{(n)} \Rightarrow^*_{R1} 1$, and we can replace the derivation $X_0, Y_1, \ldots, Y_l$ by a shorter derivation: start with an application of (PRO-E-R1) of the form $U, V \Rightarrow U, p^{(n+1)}, V$, derive $Y_1, \ldots, Y_{i-1}$ in which $q^{(m)}$ is replaced by 1, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. Again we apply the induction hypothesis.

Case 4. $X_0 \Rightarrow X_1$ is an instance of (PRO-E-R1), assume $X_0 = UV$, $X_1 = Uq^{(m)}V$, and $1 \Rightarrow^*_{R1} q^{(m)}$. There are three subcases.

Case 4.1. Neither any (GCON-R1)-step nor any (PRO-C-R1) of $Y_1, \ldots, Y_l$ acts on the designated occurrence of $q^{(m)}$. If also no (PRO-R1)-step of $Y_1, \ldots, Y_l$ acts on this designated term, then we drop the first application of the (PRO-E-R1)-step, omit $q^{(m)}$ in all types appearing in (GCON-R1)-steps, (PRO-C-R1)-steps and (PRO-R1)-steps of $Y_1, \ldots, Y_l$, then introduce $q^{(m)}$ by a single instance of (PRO-E-R1), and continue with the (GEXP-R1)-steps of $Y_1, \ldots, Y_l$; this yields a normal derivation of $X \Rightarrow Y$ of length k.

Otherwise, let $Y_{i-1} \Rightarrow Y_i$ be the first (PRO-R1)-step of $Y_1, \ldots, Y_l$ which acts on $q^{(m)}$. Then, there exist a term $r^{(m')}$ and types T, W such that $Y_{i-1} = Tq^{(m)}W$, $Y_i = Tr^{(m')}W$ and $q^{(m)} \Rightarrow^*_{R1} r^{(m')}$. Thus $1 \Rightarrow^*_{R1} r^{(m')}$, and we can replace the derivation $X_0, Y_1, \ldots, Y_l$ by a shorter derivation: first apply (PRO-E-R1) of the form $UV \Rightarrow Ur^{(m')}V$, then derive $Y_1, \ldots, Y_{i-1}$ in which $q^{(m)}$ is replaced by $r^{(m')}$, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$. Again we apply the induction hypothesis.

Case 4.2. Some (GCON-R1)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrence of $q^{(m)}$. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. Then, $Y_{i-1} = Tq^{(m)}r^{(m'+1)}W$, $Y_i = TW$ and $q^{(m)} \Rightarrow^*_{R1} r^{(m')}$. The derivation $X_0, Y_1, \ldots, Y_l$ can be replaced by a shorter derivation: drop the first application of (PRO-E-R1), then derive $Y_1, \ldots, Y_{i-1}$ in which $q^{(m)}$ is omitted, derive Y_i by a (PRO-C-R1)-step (notice $r^{(m'+1)} \Rightarrow^*_{R1} 1$), and continue $Y_{i+1}, \ldots, Y_l$. We apply the induction hypothesis.

Case 4.3. Some (PRO-C-R1)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrence of $q^{(m)}$. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. Then, $Y_{i-1} = Tq^{(m)}W$, $Y_i = TW$ and $q^{(m)} \Rightarrow^*_{R1} 1$. Then we can replace the derivation $X_0, Y_1, \ldots, Y_l$ by a shorter, normal derivation: drop the first

application of the (PRO-E-R1)-step, derive $Y_1, \ldots, Y_{i-1}$ in which $q^{(m)}$ is omitted, drop Y_i, and continue $Y_{i+1}, \ldots, Y_l$.

Case 5. $X_0 \Rightarrow X_1$ is an instance of (PRO-R1), say $X_0 = UtV$, $X_1 = UuV$, $t \Rightarrow^*_{R1} u$, $u \neq 1$ and $t \neq 1$.

Case 5.1. Neither any (GCON-R1)-step nor any (PRO-C-R1)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrence of u. Then $X_0, Y_1, \ldots, Y_l$ can be transformed into a normal derivation of the length k: drop the first application of (PRO-R1), apply all (GCON-R1)-steps of $Y_1, \ldots, Y_l$ in which the designated occurrence of u is replaced by t and apply all (PRO-C-R1)-steps, then apply a (PRO-R1)-step which changes t into u, and continue the remaining steps of $Y_1, \ldots, Y_l$.

Case 5.2. Some (GCON-R1)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrence of u. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. There are two possibilities.
(I) $Y_{i-1} = Tuq^{(n+1)}W$, $Y_i = TW$ and $u \Rightarrow^*_{R1} q^{(n)}$. Since $t \Rightarrow^*_{R1} q^{(n)}$, then $X, Y_1, \ldots, Y_l$ can be transformed into a shorter derivation: drop the first application of (PRO-R1), derive $Y_1, \ldots, Y_{i-1}$ in which the designated occurrence of u is replaced by t, derive Y_i by a (GCON-R1)-step of the form $T, t, q^{(n+1)}, W \Rightarrow T, W$, and continue $Y_{i+1}, \ldots, Y_l$. We apply the induction hypothesis.
(II) $u = q^{(n+1)}, Y_{i-1} = Tp^{(m)}uW$, $Y_i = TW$ and $p^{(m)} \Rightarrow^*_{R1} q^{(n)}$. Let $t = r(n')$. We have $q^{(n)} \Rightarrow^*_{R1} r^{(n'-1)}$, whence $p^{(m)} \Rightarrow^*_{R1} r^{(n'-1)}$. The derivation $X_0, Y_1, \ldots, Y_l$ can be transformed into a shorter derivation: drop the first application of (PRO-R1), derive $Y_1, \ldots, Y_{i-1}$ in which the designated occurrence of u is replaced by t, derive Y_i by a (GCON-R1)-step of the form $T, p^{(m)}, r^{(n')}, W \Rightarrow T, W$, and continue $Y_{i+1}, \ldots, Y_l$. We apply the induction hypothesis.

Case 5.3. Some (PRO-C)-step of $Y_1, \ldots, Y_l$ acts on the designated occurrence of u. Let $Y_{i-1} \Rightarrow Y_i$ be the first step of that kind. Then there exists types T, W, such that $Y_{i-1} = TuW$, $Y_i = TW$ and $u \Rightarrow^*_{R1} 1$. Thus $t \Rightarrow^*_{R1} 1$. The derivation $X_0, Y_1, \ldots, Y_l$ can be transformed into a normal derivation of length k: drop the first application of (PRO-R1), apply a (PRO-C-R1)-step of the form: $TtW \Rightarrow^*_{R1} TW$ derive $Y_1, \ldots, Y_{i-1}$ in which the designated occurrence of u is omitted, drop Y_i and continue $Y_{i+1}, \ldots, Y_l$.

Consequently, there holds: if $R1 \vdash_{\mathbf{CBL}} X \Rightarrow t$, where t is a term, then X can be reduced to t by (GCON-R1), (PRO-C-R1) and (PRO-R1) only. Moreover one can prove that: $R1 \vdash_{\mathbf{CBL}} t \Rightarrow u$ if, and only if, $t \Rightarrow^*_{R1} u$ ([4,9]).

Definition 7. The letter promotion problem for pregroups with unit (**LPPP1**) *can be stated as follows: verify, whether* $t \Rightarrow^*_{R1} u$ *for given* t, u *and* $R1$.

The problem can be solved in polynomial time. The algorithm for solving the problem is based on the idea given for pregroups with letter promotions. For details see [4,9].

3 The Parsing Algorithm

Definition 8. *A* pregroup grammar with letter promotions with unit *is a tuple* $G = (\Sigma, P, R1, s, I)$*, such that* $R1$ *is the set of assumptions (letter promotions) and* $P(R1) \subseteq P$*, where* $P(R1)$ *is the set of atoms appearing in assumptions from* $R1$*.* Σ, P, s, I *are defined in the same way as for pregroup grammars.*

Assume $T^+(G)$ is a set of all types appearing in I and $T(G)$ is a set of all terms forming types from $T^+(G)$. Then all generalized contractions $tu \Rightarrow \varepsilon$ $(t, u \in T(G))$ derivable from $R1$ in **CBL** can be computed in polynomial time.

We define a polynomial, dynamic, parsing algorithm for pregroup grammars with letter promotions with unit on the basis of the algorithm for pregroup grammars described in [8]. The algorithm can also be used for pregroup grammars with letter promotions without unit by omitting the cases when promotions with 1 are considered, see [9]. Our goal is to obtain an appropriate derivation of the given string $x = a_1, \ldots, a_n$, $a_i \in \Sigma$, $i = 1, \ldots, n$ if x is a member of the language generated by a pregroup grammar with letter promotions with unit G, that is if $x \in L(G)$. The algorithm is constructed in a style proposed by Savateev (see [11]) for Unidirectional Lambek Calculus. It is a dynamic algorithm working on a special form of a string, containing all possible type assignments for words of the sentence to parse.

We fix a grammar $G = (\Sigma, P, R1, s, I)$. We take a string of words $x \in \Sigma^+$ such that $x = a_1 ... a_n$. We use special symbols $*, \langle, \rangle$. Let us denote:

- $\mathcal{Z}$ - the set of integers,
- $T = \{p^{(n)} : p \in P,\ n \in \mathcal{Z}\}$ - the set of terms,
- $X, Y, Z, ..$ - elements of T^*,
- $k^a = |I(a)|$,
- X_j^a - the j-th possible assignment of type to a, $1 \leqslant j \leqslant k^a$ (hence $I(a) = \{X_1^a, ...X_{k^a}^a\}$),
- $Q^a = \langle *X_1^a * X_2^a * ... * X_{k^a}^a *\rangle$,
- $W^x = Q^{a_1} ... Q^{a_n} \langle *s^{(1)}$, $W^x \in (T \cup \{*, \langle, \rangle\})^*$
- W_i^x - the i-th symbol of the string W^x, $1 \leqslant i \leqslant |W^x|$,
- $W_{[i,j]}^x = W_i^x W_{i+1}^x ... W_j^x$ - the substring of W^x, $1 \leqslant i \leqslant j \leqslant |W^x|$ ($W_{[i,i]}^x$ stands for W_i^x).

In the following, by a reduction to 1 we mean a reduction to ε in **CBL** with letter promotions with 1. We define an auxiliary function M as follows. Let $M'(i,j), 1 \leqslant i \leqslant j \leqslant |W^x|$ be a function such that $M'(i,j) = 1$ iff one of the following conditions holds:

- **M1.** $W_{[i,j]}^x \in T^+$ and it reduces to 1.
- **M2a.** $W_{[i,j]}^x$ is of the form $\langle ... \rangle ... \langle \mathbf{V} * Z$, where:
  - $Z \in T^+$
  - $\mathbf{V}$ contains no angle brackets
  - in $W_{[i,j]}^x$ there are precisely g $(g \geqslant 0)$ pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $X_1 ... X_g Z$ reduces to 1

- **M2b.** $W^x_{[i,j]}$ is of the form $Y * \mathbf{U}\rangle ... \langle ...\rangle$, where:
  - $Y \in T^+$
  - $\mathbf{U}$ contains no angle brackets
  - in $W^x_{[i,j]}$ there are precisely g ($g \geqslant 0$) pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $YX_1...X_g$ reduces to 1
- **M3.** $W^x_{[i,j]}$ is of the form $Y * \mathbf{U}\rangle ... \langle \mathbf{V} * Z$, where:
  - $Y, Z \in T^+$
  - $\mathbf{U}, \mathbf{V}$ contain no angle brackets
  - in $W^x_{[i,j]}$ there are precisely g ($g \geqslant 0$) pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $YX_1...X_gZ$ reduces to 1
- **M4.** $W^x_{[i,j]}$ is of the form $\langle ...\rangle ... \langle ...\rangle$, where:
  - in $W^x_{[i,j]}$ there are precisely g ($g \geqslant 1$) pairs of matching angle brackets; for the h-th pair of them there is a substring of the form $*X_h*$ in between them such that $X_h \in T^+$ and the string $X_1...X_g$ reduces to 1

In all other cases $M'(i,j) = 0$.

Clearly, the whole string W^x is of the form **M2a**. Therefore, $M'(1, |W^x|) = 1$ entails the existence of a string $X_1 \dots X_n s^r$ reducing to 1. Each X_i is the type that needs to be found for a_i. Thus, a solution to the recognition problem is found, i.e. $x \in L(G)$. On the other hand, if $M'(1, |W^x|) = 0$, then there is no string reducing to 1, in which exactly one element comes from each pair of angle brackets and which reduces to 1. It means $x \notin L(G)$.

We start the algorithm by determining the set $Pairs$ of all pairs $(p^{(m)}, q^{(n+1)})$ such that $p, q \in P$ and $p^{(m)} \Rightarrow^*_{R1} q^{(n)}$, and a set $Reducible$ of terms t such that $t \Rightarrow^*_{R1} 1$, which can be done in polynomial time, see [3].

We compute $M'(i,j)$ dynamically. There are two initial cases. The first one computes $M'(i,i) = 1$ in the case when $W^x_i = t$ and $t \in Reducible$. Secondly, one looks for two adjacent terms W^x_i and W^x_{i+j} belonging to the set Pairs. If $(W^x_i, W^x_{i+j}) \in Pairs$ then we put $M'(i, i+1) = 1$.

When we already know $M'(g,h)$, for all $1 \leqslant g < h \leqslant |W^x|$ such that $h - g < j - i$, we can compute $M'(i,j)$. There are several cases:

- **A0.** $W^x_{[i,j]}$ is of the form $p^{(m)} * \mathbf{U}\rangle\langle \mathbf{V} * q^{(n+1)}$, $(p^m, q^{(n+1)}) \in Pairs$ and strings $\mathbf{U}$, $\mathbf{V}$ contain no angle brackets. Then, we put $M'(i,j) = 1$.
- **A1a.** $W^x_i, W^x_j \in T$. If there exists k such that $i \leqslant k < j$, $W^x_k \in T$, $W^x_{(k+1)} \in T$ and both $M'(i,k)$ and $M'(k+1,j)$ are equal to 1, then we put $M'(i,j) = 1$.
- **A1a'.** $W^x_i, W^x_j \in T$. If there exists k such that $i < k \leqslant j$, $W^x_k \in T$, $W^x_{(k+1)} \in T$ and both $M'(i,k-1)$ and $M'(k,j)$ are equal to 1, then we put $M'(i,j) = 1$.
- **A1b.** $W^x_i, W^x_j \in T$. If there exists k such that $i < k < j-1$, $W^x_k = \rangle$, $W^x_{(k+1)} = \langle$ and both $M'(i,k)$ and $M'(k+1,j)$ are equal to 1, then we put $M'(i,j) = 1$.
- **A2.** $W^x_i = p^{(m)}$, $W^x_j = q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$.
  If $M'(i+1, j-1) = 1$, then $M'(i,j) = 1$.

- **A3a.** $W^x_{[i,j]}$ is of the form $\langle ...\rangle ...\langle ...p^{(m)}$, $p \in P, m \in \mathcal{Z}$. If there exists k such that $i < k < j$, $W^x_k = *, W^x_{[i+1,k]}$ contains no angle brackets and $M'(k+1,j) = 1$, then $M'(i,j) = 1$.
- **A3b.** $W^x_{[i,j]}$ is of the form $p^{(m)} ...\rangle ...\langle ...\rangle$, $p \in P, m \in \mathcal{Z}$. If there exists k such that $i < k < j$, $W^x_k = *, W^x_{[k,j-1]}$ contains no angle brackets and $M'(i,k-1) = 1$, then we put $M'(i,j) = 1$.
- **A4a.** $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle ...\langle ...q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$. If $M'(k,j-1) = 1$, where k is the position of the first left angle bracket in the string $W^x_{[i,j]}$, then we put $M'(i,j) = 1$.
- **A4b.** $W^x_{[i,j]}$ is of the form $p^{(m)} ...\rangle ...\langle ... * q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$. If $M'(i+1,k) = 1$, where k is the position of the last right angle bracket in the string $W^x_{[i,j]}$, then $M'(i,j) = 1$.
- **A4c.** $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle \langle ... * q^{(n+1)}$, where the string "..." in between the angle brackets is not empty and $(p^{(m)}, q^{(n+1)}) \in Pairs$. If $M'(k,k') = 1$, where k is the position of the first left angle bracket in the string $W^x_{[i,j]}$ and k' is the position of the last right angle bracket in the string $W^x_{[i,j]}$, then $M'(i,j) = 1$.
- **A4d.** $W^x_{[i,j]}$ is of the form $p^{(m)} * ...\rangle ...\langle ...$, and $p^{(m)} \in Reducible$. If $M'(k,j) = 1$, where k is the position of the first left angle bracket in the string $W^x_{[i,j]}$, then we put $M'(i,j) = 1$.
- **A4e.** $W^x_{[i,j]}$ is of the form $...\rangle ...\langle ...*q^{(n)}$, and $q^{(n)} \in Reducible$. If $M'(i,k) = 1$, where k is the position of the last right angle bracket in the string $W^x_{[i,j]}$, then $M'(i,j) = 1$.
- **A5.** $W^x_{[i,j]}$ is of the form $\langle ...\rangle ...\langle ...\rangle$. If $M'(k,k') = 1$, where W^x_k is a term in between the first pair of angle brackets, $W^x_{k'}$ is a term in between last pair of angle brackets in the string $W^x_{[i,j]}$ and $W^x_{k-1} = *$ and $W^x_{k'+1} = *$, then $M'(i,j) = 1$.
- **A6a.** $W^x_{[i,j]}$ is of the form $p^{(m)} q^{(n)} ...$, and $p^{(m)} \in Reducible$. If $M'(i+1,j) = 1$, then we put $M'(i,j) = 1$.
- **A6b.** $W^x_{[i,j]}$ is of the form $...p^{(m)} q^{(n)}$, and $q^{(n)} \in Reducible$. If $M'(i,j-1) = 1$, then we put $M'(i,j) = 1$.

In all other cases $M'(i,j) = 0$.

Note that the high number of cases is due to the variety of the form of the string W^x. Moreover, observe that from each pair of matching angle brackets exactly one type must be chosen. Let us call any substring of W^x that satisfies all conditions of any of the forms of **M** *accepted*. We start with finding all terms and pairs of adjacent terms that can be reduced to 1, that is the shortest accepted substrings. Then we try to extend them to obtain a longer accepted substring. Obviously the procedure is continued until there are no more possibilities of extending obtained substrings in an acceptable way or the whole string W^x can be reached. For example, cases **(A1a)**, **(A1a')** and **(A1b)** show which conditions two substrings have to satisfy to be concatenated. Case **(A2.)** explains when the substring can be surrounded by a link (i.e. two terms that reduce to 1) while all

of the cases **A3**, **A4** and **A5** ensure all possibilities of lengthening the substring by terms from the adjacent pair of angle bracket (and all symbols between).

We claim:

Theorem 2. *The algorithm computes $M'(i,j)$ correctly.*

Proof. We will show at first that, if the algorithm computes $M'(i,j)=1$, then $M'(i,j)=1$ according to the definition of M. We will prove it by induction on the length of the string $W^x_{(i,j)}$.

For strings of length one, $M'(i,j)=1$ only in case when $W^x_i = p^{(m)}$ and $p^{(m)} \in Reducible$. $W^x_{[i,i]}$ is then of the form **(M1)**, since $W^x_{[i,i]} \in T^+$ and the string $W^x_{[i,i]}$ reduces to 1. Hence, $M'(i,i)=1$ according to the definition of M.

Consider now the strings of length two. The algorithm computes $M'(i,i+1)=1$ only in case when $W^x_i = p^{(m)}$ and $W^x_{i+1} = q^{(n+1)}$ and $(p^{(m)}, q^{(n+1)}) \in Pairs$. $W^x_{[i,j]}$ is then of the form **(M1)**, since $W^x_{[i,i+1]} \in T^+$ and the string $W^x_{[i,i+1]}$ reduces to 1. Hence, $M'(i,i+1)=1$ according to the definition of M.

Now let us consider the recursive cases when the algorithm computes $M'(i,j)=1$. We present only some cases different from those in the proof for pregroup grammars, given in [8] (see [9] for the full proof).

Case A4d. $W^x_{[i,j]}$ is of the form **(A4d)**. $W^x_{[i,j]} = \underbrace{p^{(m)} * \ldots *}_{i,\ no\ \langle,\rangle} \underbrace{\rangle\langle \ldots \rangle}_{k,\ j,\ M'(k,j)=1}$,

where $p^{(m)} \in Reducible$ and k is the position of the first left angle bracket in the string $W^x_{[i,j]}$. $W^x_{[k,j]}$ is shorter than $W^x_{[i,j]}$. Hence, by the induction hypothesis, $M'(k,j)=1$ according to the definition of M. The string $W^x_{[k,j]}$ must then be of the form:

- **(M2a)**. Then $W^x_{[k,j]} = \langle \ldots \rangle \ldots \langle \mathbf{V} * Z$, where Z is the string of terms, $\mathbf{V}$ contains no angle brackets and there are precisely g ($g \geqslant 0$) pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1 \ldots X_g Z$ reduces to 1. Let $Y = p^{(m)}$. Then $YX_1 \ldots X_g Z$ also reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M3)**. Then $M'(i,j)=1$ in accordance with the definition of M.
- **(M4)**. Then $W^x_{[k,j]} = \langle \ldots \rangle \ldots \langle \ldots \rangle$ and there are precisely g ($g > 0$) pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1 \ldots X_g$ reduces to 1. Let $Y = p^{(m)}$. Then $YX_1 \ldots X_g$ also reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M2b)**. Then $M'(i,j)=1$ in accordance with the definition of M.

Case A6b. $W^x_{[i,j]}$ is of the form $W^x_{[i,j]} = \underbrace{\ldots p^{(m)}}_{i,\ j-1,\ M'(i,j-1)=1} q^{(n)}$, j

where $q^{(n)} \in Reducible$. $W^x_{[i,j-1]}$ is shorter than $W^x_{[i,j]}$. Hence, by the induction hypothesis, $M'(i,j-1)=1$ according to the definition of M. The string $W^x_{[i,j-1]}$ can therefore be of the form:

- **(M1)**. Then $W^x_{[i,j-1]} \in T^+$ and $W^x_{[i,j-1]}$ reduces to 1. Then $W^x_{[i,j]}$ is also of the form **(M4)** and it reduces to 1. Thus, $M'(i,j) = 1$ in accordance with the definition of M..
- **(M2a)**. Then $W^x_{[i,j-1]} = \langle ... \rangle ... \langle \mathbf{V} * Z$, where Z is the string of terms, $\mathbf{V}$ contains no angle brackets and there are precisely g ($g \geqslant 0$) pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $X_1 ... X_g Z$ reduces to 1. Let $Z' = Zq^{(n)}$. Then $X_1 ... X_g Z'$ also reduces to 1, and the string $W^x_{[i,j]}$ is therefore of the form **(M2a)**. Then $M'(i,j) = 1$ in accordance with the definition of M.
- **(M3)**. Then $W^x_{[i,j-1]} = Y * \mathbf{U} \rangle ... \langle ... \rangle \mathbf{V} * Z$, where Y, Z are the strings of terms, $\mathbf{U}, \mathbf{V}$ contain no angle brackets and there are precisely g ($g \geqslant 0$) pairs of matching angle brackets, for the h-th of them there is the substring $*X_h*$ in between them, such that $X_h \in T^+$ and $YX_1 ... X_g Z$ reduces to 1. Let $Z' = Zq^{(n)}$. Then $YX_1 ... X_g Z'$ reduces to 1, and the string $W^x_{[i,j]}$ is therefore also of the form **(M3)**. Then $M'(i,j) = 1$ in accordance with the definition of M.

We will prove that the algorithm correctly finds all substrings for which the function $M'(i,j) = 1$ by induction on the length of the substring of W^x. Note that there are no such substrings that contain asterisks but no angle brackets.

The only strings of length one for which $M'(i,j) = 1$ is of the form $p^{(m)}$, where $p^{(m)} \in T$ and $p^{(m)} \in \mathit{Reducible}$ and the algorithm finds them correctly (the string is of the form **M1**). The only strings of length two, for which $M'(i,j) = 1$, are of the form $p^{(m)}q^{(n+1)}$, where $(p^{(m)}, q^{(n+1)}) \in \mathit{Pairs}$ (that is of the form **(M1)**), and the algorithm finds them correctly.

Let us now consider the substrings of the length $l > 2$ such that for all $l' < l$ the algorithm finds the substrings of the length l' correctly (we present some chosen cases).

Case **3.** $W^x_{[i,j]}$ is of the form **(M3)**. $W^x_i W^x_j$ takes part in the reduction to 1 in $W^x_{[i,j]}$.

First, let us assume $W^x_i W^x_{i'}$ reduce to 1 where $i' \neq j$. Then $W^x_{i'+1}$ can be:

- term. Then $W^x_{[i,i']}$ and $W^x_{[i'+1,j]}$ are of the form **(M1)** or **(M3)**, they are shorter and both $M'(i,i') = 1$ and $M'(i'+1,j) = 1$. Hence, by the induction hypothesis, these strings are found by the algorithm correctly, so $M'(i,j) = 1$ (case (**A1a**)).
- asterisk. Then $W^x_{[i,i']}$ is of the form **(M1)** or **(M3)**, it is shorter and $M'(i,i') = 1$. Hence, by the induction hypothesis, the string is found by the algorithm correctly. Let k be the position of the first right angle bracket following i'. $W^x_{[i,k]}$ is shorter than $W^x_{[i,j]}$ and it is of the form **(M2b)**. So, by the induction hypothesis $M'(i,k) = 1$ (case (**A3b**)). Similarly, the substring $W^x_{[k+1,j]}$ is of the form **(M2a)**, it is shorter than $W^x_{[i,j]}$. So, by the induction hypothesis $M'(k+1,j) = 1$ (case (**A3a**)). Hence, $M'(i,j) = 1$, by the induction hypothesis (case (**A1b**)).

Otherwise, let us assume $W_i^x \in \mathit{Reducible}$. Then W_{i+1}^x can be:

- term. Then $W_{[i+1,j]}^x$ is of the form **(M3)**, it is shorter, so by the induction hypothesis $M'(i+1,j)=1$. Moreover, $W_{[i,i]}^x$ is of the form **M1**, it is shorter and $M'(i,i)=1$. Hence, by the induction hypothesis, these strings are found by the algorithm correctly. Then $M'(i,j)=1$ (case (**A1a**)).
- asterisk. Let k be the position of the first right angle bracket following i. $W_{[i,k]}^x$ is shorter than $W_{[i,j]}^x$ and it is of the form **(M2b)**. So, by the induction hypothesis $M'(i,k)=1$. Similarly, the substring $W_{[k+1,j]}^x$ is of the form **(M2a)**, it is shorter than $W_{[i,j]}^x$. So, by the induction hypothesis $M'(k+1,j)=1$. Hence, $M'(i,j)=1$, by the induction hypothesis (case (**A1b**)).

Let us assume $i'=j$ so $W_i^x W_j^x$ reduces to 1. There are the following cases.

- W_{i+1}^x, W_{j-1}^x are terms. Then the substring $W_{[i+1,j-1]}^x$ is of the form **(M3)**. It is shorter than $W_{[i,j]}^x$, therefore $M'(i+1,j-1)=1$, as by the induction hypothesis, that string is found by the algorithm correctly. Then $M'(i,j)=1$ (case (**A2**)).
- $W_{i+1}^x = *, W_{j-1}^x \in T$. So $W_{[i,j]}^x$ is of the form $p^{(m)} * ... \rangle ... \langle ... q^{(n+1)}$. Let i' be the position of the first left angle bracket in $W_{[i,j]}^x$. There exists $i' < k \leqslant j-1$ such that $W_k^x W_{j-1}^x$ reduces to, or if $k=j-1$, then $W_{j-1}^x \in \mathit{Reducible}$. Hence, $W_{[i',j-1]}^x$ is of the form **(M2a)**. It is shorter than $W_{[i,j]}^x$, therefore $M'(i',j-1)=1$, as by the induction hypothesis, that string is found by the algorithm correctly. Then $M'(i,j)=1$ (case (**A4a**)).
- $W_{i+1}^x \in T, W_{j-1}^x = *$. It is proved symmetrically to the case $W_{i+1}^x = *, W_{j-1}^x \in T$.
- $W_{i+1}^x = *, W_{j-1}^x = *$. Then the string $W_{[i,j]}^x$ can be of the form $p^{(m)} * \mathbf{U} \rangle \langle \mathbf{V} * q^{(n+1)}$ and then $M'(i,j)=1$ by the initial case of the description of the algorithm. If $W_{[i,j]}^x$ contains more brackets, then there is a string $W_{[k,k']}^x$, where k is the index of the first left angle bracket in the string $W_{[i,j]}^x$ and k' is the index of the last right angle bracket in the string $W_{[i,j]}^x$. $W_{[k,k']}^x$ is of the form **(M4)**. It is shorter than $W_{[i,j]}^x$, therefore, by the induction hypothesis, $M'(k,k')=1$. So, $M'(i,j)=1$ by (**A4c**).

Obviously, the algorithm is not yet a real parsing algorithm since it answers only the question whether there exists any reduction to the designated type. However it can easily be modified to find such reductions, still in polynomial time. It can be done exactly as in the algorithm for pregroup grammars.

Each obtained reduction is described by the set of links involved in the reduction. If we want to obtain only one reduction, the complexity of the algorithm does not increase. The set of links $L(i,j)$ represents a reduction of some term to 1. Links are denoted by pairs of integers (k,l) such that $i \leqslant k < l \leqslant j$. We find the set of links by backtracking the indices of the function $M'(i,j)=1$, obviously starting with $M'(1,|W^x|)$. We also define an auxiliary function $\mathit{Prev}(i,j)$ to help us follow the backtracking (as the value of the function $M'(i,j)$ does

not say how it was obtained). The value of the function $Prev(i,j)$ is a sequence of three pairs $((l_1,l_2),(m1_1,m1_2),(m2_1,m2_2))$, where l_1,l_2 are indices of the link, $m1_1,m1_2,m2_1,m2_2$ are indices of function M on which the computation $M'(i,j)=1$ is based. If any of the values is not used, it is set to 0. Every time when the algorithm computes the value of the function $M'(i,j)=1$ we set the value of the function $Prev(i,j)$ in the following way. If the computation was executed by one of the cases:

- any initial case or (**A0**), then $Prev(i,j)=((i,j),(0,0),(0,0))$,
- (**A2**), (**A4a**), (**A4b**), (**A4c**), then $Prev(i,j)=((i,j),(k,l),(0,0))$, where (k,l) is the pair of indices for which the value of the function M was 1 in the current computation (that is e.g. in (**A2**) a pair $(k,l)=(i+1,j-1)$),
- (**A3a**), (**A3b**), (**A5**), then $Prev(i,j)=((0,0),(k,l),(0,0))$, where (k,l) is the pair of indices for which the value of the function M was 1 in the current computation,
- (**A1a**), (**A1a'**), (**A1b**), then $Prev(i,j)=((0,0),(i,k),(k+1,j))$,
- (**A4d**), then $Prev(i,j)=((i,i),(k,j),(0,0))$, where (k,j) is the pair of indices for which the value of the function M was 1 in the current computation,
- (**A4e**), then $Prev(i,j)=((j,j),(i,k),(0,0))$, where (i,k) is the pair of indices for which the value of the function M was 1 in the current computation,
- (**A6a**), then $Prev(i,j)=((i,i),(i+1,j),(0,0))$,
- (**A6b**), then $Prev(i,j)=((j,j),(i,j-1),(0,0))$.

Obviously, one can choose whether the algorithm should remember the first computed reduction or the last computed reduction. In the first case if $Prev(i,j)\neq((0,0),(0,0),(0,0))$, then it cannot be modified. In the latter, $Prev(i,j)$ is updated every time when the algorithms computes $M'(i,j)=1$. When the computation of the functions M and $Prev$ is finished, we easily compute the set $L(1,|W^x|)$. The definition of the function $L(i,j)$ is as follows:

- if $Prev(i,j)=((i,j),(0,0),(0,0))$, where $0<i<j$, then $L(i,j)=\{(i,j)\}$,
- if $Prev(i,j)=((i,j),(k,l),(0,0))$, where $0<i\leqslant k<l\leqslant j$, then $L(i,j)=L(k,l)\cup\{(i,j)\}$,
- if $Prev(i,j)=((0,0),(k,l),(0,0))$, where $0<i\leqslant k<l\leqslant j$, then $L(i,j)=L(k,l)$,
- if $Prev(i,j)=((0,0),(i,k),(k+1,j))$, where $0<i<k<j$, then $L(i,j)=L(i,k)\cup L(k+1,j)$.

The algorithm is polynomial and works in time proportional to n^3, where n is the length of the string W_x, assuming the set *Pairs* and *Reducible* are determined. The procedures for computing the set *Pairs* and the set *Reducible* are polynomial.

References

1. Buszkowski, W.: Lambek Grammars Based on Pregroups. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, pp. 95–109. Springer, Heidelberg (2001)
2. Buszkowski, W., Moroz, K.: Pregroup grammars and context-free grammars. In: Casadio, C., Lambek, J. (eds.) Computational Algebraic Approaches to Natural Language, Polimetrica, pp. 1–21 (2008)
3. Buszkowski, W., Lin, Z.: Pregroup Grammars with Letter Promotions. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 130–141. Springer, Heidelberg (2010)
4. Buszkowski, W., Lin, Z., Moroz, K.: Pregroup Grammars with Letter Promotions: Complexity and Context-Freeness. Journal of Computer and System Sciences 78(6), 1899–1909 (2012)
5. Lambek, J.: Type Grammars Revisited. In: Lecomte, A., Perrier, G., Lamarche, F. (eds.) LACL 1997. LNCS (LNAI), vol. 1582, pp. 1–27. Springer, Heidelberg (1999)
6. Lambek, J.: From Word to Sentence: a computational algebraic approach to grammar. Polimetrica (2008)
7. Mater, A.H., Fix, J.D.: Finite Presentations of Pregroups and the Identity Problem. In: Proceedings of FG-MoL 2005, pp. 63–72. CSLI (electronic) (2005)
8. Moroz, K.: A Savateev-style parsing algorithm for pregroup grammars. In: de Groote, P., Egg, M., Kallmeyer, L. (eds.) FG 2009. LNCS, vol. 5591, pp. 133–149. Springer, Heidelberg (2011)
9. Moroz, K.: Algorithmic questions for pregroup grammars, PhD thesis, Adam Mickiewicz University, Poznań (2010)
10. Pentus, M.: Lambek calculus is NP-complete. Theoretical Computer Science 357, 186–201 (2006)
11. Savateev, Y.: Unidirectional Lambek Grammars in Polynomial Time. Theory of Computing Systems 46, 662–672 (2010)

Towards an HPSG Analysis of Object Shift in Danish

Stefan Müller* and Bjarne Ørsnes

German Grammar, FU Berlin

Abstract. This paper develops an analysis of object shift in Danish. We suggest that object shift is best analyzed as an alternative mapping from the ARG-ST list to SPR and COMPS.

1 Introduction

Danish is an SVO language. This order is demonstrated by the following subordinated clause:

(1) at Jens har læst bogen
that Jens has read book.DEF
'that Jens has read the book'

Apart from being an SVO language, Danish is a verb second (V2) language, that is, any constituent can appear in front of the position of the finite verb in declarative main clauses. (2) shows an example in which the object is fronted:

(2) Bogen har Jens læst.
book.DEF has Jens read
'Jens has read the book.'

Adjuncts attach to the VP and are serialized either to the left or to the right. The negation obligatorily attaches to the left:

(3) at Jens ikke [$_{VP}$ læser bogen]
that Jens not reads book.DEF
'that Jens does not read the book'

In V2 sentences the finite verb is inverted, that is, placed to the left of the subject. One common analysis in GB/Minimalism and HPSG is to assume that the inverted verb is related to a verb trace in the VP. (4) shows the structure:

* We want to thank the audiences of the Third International Workshop on Germanic Languages (With Special Focus on Scandinavian) that was held 2012 at the Freie Universit[Pleaseinsert"PrerenderUnicode–"intopreamble]t Berlin and the participants of the HPSG workshop in Frankfurt, 2012 for discussion. Special thanks go to Sten Vikner for intense discussion of object shift. This work was supported by the Deutsche Forschungsgemeinschaft (DFG MU 2822/2-1).

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 69–89, 2013.

(4) Jens læser$_i$ ikke [$_{\text{VP}}$ _$_i$ bogen].
Jens reads not book.DEF

The negation can then be used as an indicator that marks the left periphery of the VP even in sentences with an inverted verb.

This paper deals with a certain order that is required for personal, reflexive, or locative pronouns in a non-subject function. These pronouns do not occur in the canonical position inside the VP (to the right of sentential adjuncts), but rather outside the VP to the left of sentential adjuncts. The examples in (5) show that a full NP *bogen* ('the book') must occur inside the VP to the right of sentential negation. The examples in (6) show that the unstressed pronoun *den* ('it') occurs outside the VP linearly preceding the sentential adverb *ikke* ('not').

(5) a. Jens læser ikke bogen.
Jens reads not book.DEF
'Jens is not reading the book.'

b. * Jens læser bogen ikke.
Jens reads book.DEF not

(6) a. Jens læser den ikke.
Jens reads it not
'Jens is not reading it.'

b. * Jens læser ikke den.
Jens reads not it

The paper will be structured as follows: in the next section we discuss the shifting in double object constructions and interactions of verb inversion, verb fronting, and object shift (facts that are known under the term Holmberg's Generalization). We then briefly discuss alternative proposals in Section 3 and provide our analysis in Section 4. Section 5 draws some conclusions.

2 The Phenomenon

Object shift applies in verb initial (V1) or V2 clauses, but is not possible in embedded clauses without verb inversion:[1]

(7) a. at Jens ikke giver dem bogen
that Jens not gives them book.DEF
'that Jens did not give the book to them'

b. * at Jens dem ikke giver bogen
that Jens them not gives book.DEF

c. Jens giver$_i$ dem$_j$ ikke [_$_i$ _$_j$ bogen]
Jens gives them not book.DEF

[1] Note, that the traces are used to mark the positions that full objects would take. While we are using traces for verb movement in our analysis, we do not assume a movement-based approach of object shift.

Object shift is strictly clause-bound (Vikner, 2006, p. 405). A pronoun can never shift into the matrix construction of an embedding verb. This situation would only obtain in a context with embedded V2, that is, when a verb selects a clause with verb fronting, since object shift is only observed in V1- and V2-clauses. In clauses with embedded V2, a pronoun cannot shift across its selecting head into the matrix clause. In the examples in (8), the complement clause is V2 with the object *forretten* ('the starter') in position before the finite verb, the so-called the prefield. As (8b) shows the reflexive pronoun *sig* ('REFL') cannot occur in the matrix clause.

(8) a. Jeg ved at forretten brød han [sig] ikke om.
I know that starter.DEF cared he REFL not about
'I know that he didn't like the starter.'

b. * Jeg ved [sig] at forretten brød han ikke om.
I know REFL that starter.DEF cared he not about

While Icelandic allows full NPs to shift, shifting is limited to weak pronouns in Danish: As Mikkelsen (2011, p. 252) shows, shifted elements have to be unstressed and they may not be phrase structurally complex.

It is possible to shift both objects of a ditransitive verb (9a), but the relative order of the objects has to be preserved, that is, the indirect object precedes the direct object as in sentences with full objects (9c).

(9) a. Han giver ham det ikke.
he is.giving him it not
'He is not giving it to him.'

b. * Han giver det ham ikke.
he is.giving it him not

c. Han giver ikke manden bogen.
he is.giving not man.DEF book.DEF
'He is not giving the man the book.'

It is not possible to shift the DO over the IO, as the contrast in (10) shows:

(10) a. Han skænkede ikke biblioteket bogen.
he donated not library.DEF book.DEF
'He did not donate the book to the library

b. ?* Han skænkede$_j$ den$_k$ ikke [_$_j$ biblioteket _$_k$].
he donated it not library.DEF
'He didn't donate it to the library.'

Interestingly though, if the IO is extracted, the DO can be shifted:

(11) Biblioteket$_i$ skænkede$_j$ han den$_k$ ikke [_$_j$ _$_i$ _$_k$].
library donated he it not
'He didn't donate it to the library.'

The only situation in which a DO can precede an IO is a configuration in which the DO is positioned to the left of the finite verb in the so-called prefield. (12) gives an example:

(12) Bogen$_k$ skænkede$_j$ han [_$_j$ biblioteket _$_k$].
book.DEF donated he library.DEF

Having discussed examples with transitive and ditransitive verbs we now turn to prepositional objects: Full PPs do not shift as (13) shows:

(13) a. Vi venter ikke på dig.
we wait not for you
'We are not waiting for you.'

b. * Vi venter på dig ikke.
we wait for you not

Shifting of a pronoun out of a PP is also impossible despite the general possibility of P stranding:

(14) a. * Vi venter dig ikke på.
we wait you not for
Intended: 'We are not waiting for you.'

b. Dig venter vi ikke på.
you wait we not for
'We are not waiting for you.'

The generalization about the data is that shifted elements have to be arguments of a verb. This generalization also captures valence-bound locatives.

V2 clauses (for instance, (2) and (14b)) are analyzed as nonlocal dependencies (that is, movement in GB/Minimalism and SLASHed categories in GPSG/HPSG), since the element before the finite verb may be a dependent of a deeply embedded head. The question now is whether reorderings of pronouns should be treated with the same mechanisms. There is evidence against analyses that treat shifting parallel to extractions of the prefield filling kind: For instance, Holmberg (1999, p. 18) and Vikner (2006) discussed shifted pronouns and argued that they do not license parasitic gaps. Extracted elements like *hvad for en bog* ('which book') in (15a) licence a second gap in an adjunct as for instance the phrase *uden at læse først* ('without reading first') (see Vikner 2006, p. 11 for a discussion of the examples in (15)). In example (15a) the fronted *wh*-constituent *hvad for en bog* ('which book') is co-indexed with a gap in the object position of the verb *stille* ('to put'). This gap, in turn, licenses the second gap (the object of *læse* ('to read')). If shifted pronouns would leave a trace inside the VP, we should expect them to be able to license parasitic gaps. However, in example (15b) the shifted object *den* ('it') is co-indexed with the first gap, and here the second gap (the object of *læse* ('to read')) is not licensed.

(15) a. [Hvad for en bog]$_i$ stillede alle _$_i$ hen på reolen uden at
which book put all onto bookcase.DEF without to
læse _$_i$ først?
read first
'Which book did everyone put on the shelf without reading first?'

b. * Alle stillede den$_i$ straks $\__i$ hen på reolen uden at
all put it immediately onto bookcase.DEF without to
læse $\__i$ først.
read first
'Everyone put it on the shelf without reading it first.'

This suggests that there is a fundamental difference between object shift and extraction to the prefield.

The examples in (7) showed that object shift does not occur in embedded clauses, that is, pronouns do not shift over finite verbs. The same observation can be made with regard to non-finite verbs:

(16) a. Jeg har ikke kysset hende.
I have not kissed her
'I hav not kissed her.'

b. * Jeg har hende ikke kysset.
I have her not kissed

But the shifting of pronouns becomes possible if the non-finite verb is extracted (Vikner, 2006, p. 407):

(17) a. Kysset$_i$ har$_j$ jeg hende$_k$ ikke [$\__j$ $\__i$ $\__k$], bare holdt hendes hånd.
kissed have I her not only held her hand
'I have not kissed her. I only held her hand.'

b. men helt [udelukke] kan man [det] da ikke eller hvad[2]
but wholly exclude can you it then not or what
'but you cannot wholly exclude it, can you?'

The facts about the necessity to invert the verb to V1/V2 order, to extract the non-finite verb, and to extract the IO if the DO is shifted have subsumed under Holmberg's Generalization (1999) in Transformational Grammar: Pronouns can shift only if they are the left-most overt element in the VP. This explains, why the finite and the non-finite verb has to be out of the way and why the DO cannot shift unless the IO is extracted. However, as will be shown in Section 3 a purely movement-based proposal runs into problems. A different way to describe the situation is to say that the verbs have to precede their arguments independent of shifting and that the IO has to precede the weak pronominal DO independent of extraction and shifting. (Note that weak pronouns cannot occupy the prefield, so the statement just made cannot be falsified by DO extractions to the prefield.)

3 Previous Analyses

3.1 Cliticisation

Erteschik-Shir (2005) suggested a cliticization approach to object shift. She assumes that weak pronouns cannot be pronounced on their own and hence must

[2] http://hope.pointblog.dk/svaert-at-vide-.html, 26.03.2012.

incorporate into a host. In her approach adverbials (by stipulation) cannot serve as hosts for prosodic incorporation and hence the pronoun must shift over adverbials. In an example like (18) the pronoun attaches to the subject and subject, pronoun, and negation form a prosodic unit.

(18) Læser Peter+den+ikke?
reads Peter+it+not
'Dosn't Peter read it?'

As Holmberg (1999, p. 28, Footnote 26) pointed out while discussing Hellan's analysis of object shift in Norwegian (1994), the analysis in Erteschik-Shir (2005) fails to explain why a weak pronoun also has to shift in the presence of a PP-adjunct. In the examples in (19) the sentential adjunct is syntactically a PP with a preposition and a DP object. Therefore, we should expect the pronoun to be able to incorporate into the DP *stor sandsynlighed* ('great probability'), given that DPs are possible hosts for phonological incorporation. But the pronoun does not incorporate into these constituents, instead it shifts.

(19) a. * Hun kender med stor sandsynlighed ham ikke.
she knows with big probability him not
Intended: 'It is most likely that she doesn't know him.'

b. Hun kender ham med stor sandsynlighed ikke.
she knows him with big probability not

Finally, Holmberg (1999, p. 27) pointed out another problem for the clitic analysis: it does not extend to object shift in Icelandic and Faroese that allow for complete NPs to undergo object shift.

3.2 Movement

As was shown in Section 2 the parasitic gap data is evidence against movement-based approaches. Furthermore there are problems, if one wants to capture Holmberg's generalization in a movement approach. To see this consider the analysis in Figure 1 on the facing page. If the DO can move only when the IO has moved already, the IO would have to move and to attach to the VP (or IP). The DO would be the next thing to move. However, the resulting order is ungrammatical and there is no way to get the correct order, if Holmberg's view has to play a role in the analysis.

3.3 Linearization-Based Analyses

Bjerre (2006) suggested a linearization-based approach in the framework of HPSG. In such approaches the dependents of a head are inserted into one flat list and linearization rules (LP rules) restrict the possible linearizations of elements in this list (Reape, 1994). Bjerre assumes traditional topological fields and sets up the LP statements accordingly.

Consider the analysis for the sentence in (20) which is given in Figure 2 on the next page:

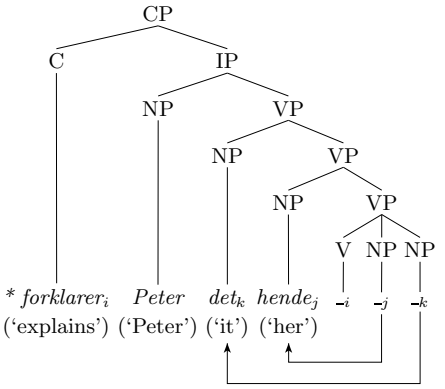


Fig. 1. Problems for movement-based approaches

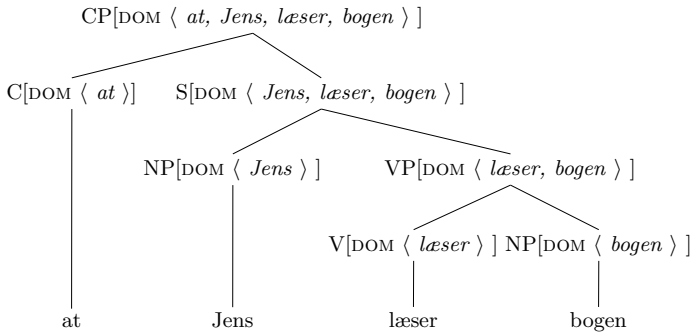


Fig. 2. Linearization-based analysis of Danish clauses

(20) at Jens læser bogen
that Jens reads book.DEF
'that Peter reads the book'

The domain objects are complex linguistic objects that are similar to the ones that we are using here. In the figure only the PHON values are given. Every lexical item comes with a domain object that represents its phonological, syntactic, and semantic properties. When a complex object is build, the domain objects of the daughters are inserted into the domain of the mother node. As Figure 2 shows, we end up with a flat representation of all constituents at the top-most node in the tree.

Bjerre (2006) suggests that syntactic functions are assigned to syntactic positions (by means of an appropriate type hierarchy) and that linear precedence is stated in terms of these syntactic positions. The type *verbal* is assigned to the position m (corresponding to the *fronted* position) and the position V (the *base* position within the VP). The syntactic function *object* is assigned to the

positions I and N, saying that an object can occur in position I (the position for shifted objects) or in the position N (the position of full NP objects within the VP). *n* is the field for the subject and *a1* the field for VP adjuncts. *F* is the field the corresponds to the prefield. The order of elements on the DOM list is constrained by precedence rules of the following (simplified) kind.

(21) $F < m < n < I < a1 < V < N$

Figure 3 shows our example augmented with the negation *ikke* and with the topological field assignment. The interesting case is now the analysis of object shift, which has the same structure as the non-shifted example but a different linearization. The analysis is given in Figure 4 on the next page.[3] The object pronoun is assigned to the field *I* rather than *N* and hence is linearized to the left of the adverb. Since the verb is assigned to the field *m* it precedes both the shifted pronoun and the adverb. It should be clear from the pictures that in linearization-based analyses the dominance structure is independent of the actual serialization of components. In particular discontinuous constituents are allowed in such models.

According to Bjerre the elements that are inserted into the prefield are inserted as a single domain object. With these basic assumptions it is unclear how the following example can be captured:

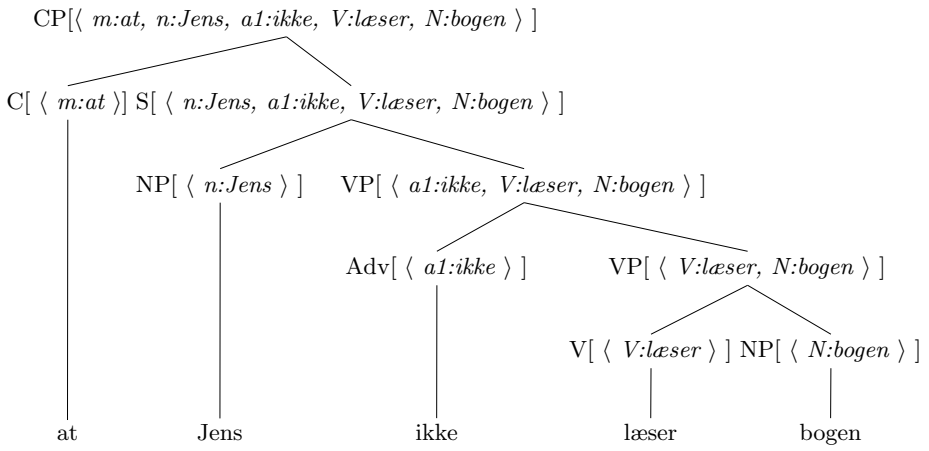


Fig. 3. Linearization-based analysis of Danish clauses with topological labels

[3] Bjerre assumes that objects that are positioned in the prefield are licenced there in head-filler structures. Probably he would apply this to subjects as well. Figure 4 would have to be augmented with a trace in the subject position and a Head-Filler combination at the top of the structure. However, this would not change the DOM values and assignment of topological fields, since traces are assumed to not contribute any domain objects.

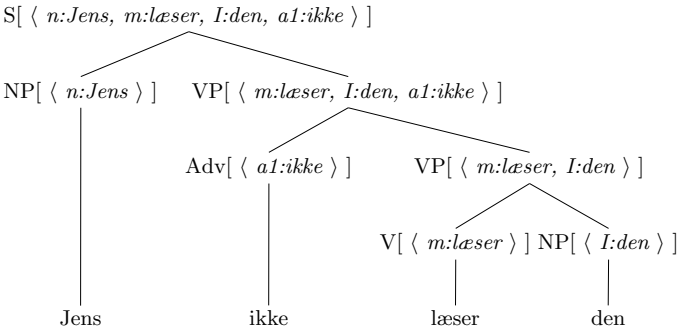


Fig. 4. Linearization-based analysis of object shift

(22) Læst har Jens den ikke.
read has Jens it not
'Jens did not read it.'

The problem is that the argument of the auxiliary verb *har* ('has') is a VP, but the VP is discontinuous in (22) since the object *den* ('it') appears in shifted position to the left of the negation. One could claim that auxiliaries form a single linearization domain with the verb that they embed and with the arguments of the embedded verb. This would be the clause union analysis that Reape suggested for the treatment of so-called coherent constructions in German, that is, for verbal complex formation. However, this would overgenerate, since this would allow sentences like (23):

(23) * Læst har Bjarne ikke bogen
read has Bjarne not book.DEF
'Bjarne did not read the book.'

The problem with (23) is that the object of *læst* ('read') is a full noun. It is not possible to front bare verbs in Danish, if the object is a full NP. If the object is a full NP it has to be fronted together with the verb as in (24):

(24) Læst bogen har Bjarne ikke
read book.DEF has Bjarne not
'Bjarne did not read the book.'

This is explained in approaches that assume that the argument of the auxiliary verb is a VP, that is, a verbal projection that includes all complements. Note that (23) cannot be ruled out by linearization constraints that refer to traditional topological fields since all constituents are in fields in which they can be linearized: The full NP is to the right of the negation as in (5a), the finite verb is in second position and the non-finite verb is in the prefield as in (22).

The problem could be solved by assuming partial compaction of domain objects à la Kathol and Pollard 1995. In such an approach one domain object would be inserted into the prefield if the object is a full NP and two objects would be

inserted into the matrix domain if the object in the extracted VP is a pronoun. Such an approach seems to be stipulative since it would have to formulate a complicated mechanism that applies only to the domain insertion of extracted VPs.

4 The Analysis

This section will provide an analysis in the framework of HPSG (Pollard and Sag, 1994). We will give the background assumtions in Section 4.1, develop the core of the analysis in Section 4.2. Section 4.3 deals with prepositional arguments and Section 4.4 describes the analysis of partial fronting and Section 4.5 explains how Holmberg's Generalization is captured in the analysis.

4.1 Background

Following Pollard and Sag (1994, Chapter 9) we assume that arguments of a head are represented in a list that is ordered according to the obliqueness hierarchy, that is, in the order Subj < IO < DO < Obliques (Pollard and Sag, 1992, p. 266, 280). In recent publications this list is called the argument structure list (ARG-ST). The elements from the ARG-ST list are mapped to the valence features SPR and COMPS. SPR stands for *specifier* and COMPS for *complements*. The SPR list may contain the determiner of an NP or the subject of a verb. (25) shows an example of a lexical item of a ditransitive verb with the arguments in the ARG-ST list linked to the semantic contribution of the verb and with the respective mapping to SPR and COMPS:

(25) *give*:

$$\begin{bmatrix} \textsc{spr} & \langle\, \mathrm{NP}_{\boxed{1}} \,\rangle \\ \textsc{comps} & \langle\, \mathrm{NP}_{\boxed{2}},\ \mathrm{NP}_{\boxed{3}} \,\rangle \\ \textsc{arg-st} & \langle\, \mathrm{NP}_{\boxed{1}},\ \mathrm{NP}_{\boxed{2}},\ \mathrm{NP}_{\boxed{3}} \,\rangle \\ \textsc{rels} & \left\langle \begin{bmatrix} \textsc{agent} & \boxed{1} \\ \textsc{goal} & \boxed{2} \\ \textsc{theme} & \boxed{3} \\ \textit{give} & \end{bmatrix} \right\rangle \end{bmatrix}$$

We follow Müller (To appear) in assuming binary branching structures and hence assume binary branching schemata for specifier head and head complement combinations. The analysis of an embedded clause is shown in Figure 5 on the facing page. The verb is combined with the IO first and the resulting object is combined with the DO and the result of this combination is a complete VP (the abbreviation VP stands for a linguistic object with the head category verb and an empty COMPS list). The VP is combined with the specifier to the left resulting in a complete clause (abbreviated as S). The complementizer selects for an S and the result of the combination of complementizer and S is a CP.

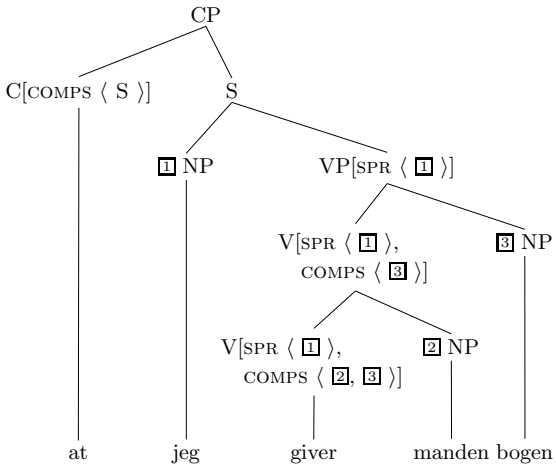


Fig. 5. Analysis of *at jeg giver manden bogen* ('that I give man.DEF book.DEF')

For the analysis of V1 and V2 sentences we follow proposals by Borsley (1989b) for English and Kiss and Wesche (1991); Kiss (1995); Meurers (2000); Müller (2007b) for German and assume a head movement analysis. The head movement analysis has the advantage that a uniform treatment of VP adjuncts is possible: the adjuncts attach to the left or to the right of VP and there is no difference between main and embedded clauses in this respect. The Danish verb inversion is similar to auxiliary inversion in English except that it applies to all finite verbs. While analyses involving empty elements should be avoided if possible it has been shown that so-called multiple frontings in German are best analyzed as combinations of the fronted elements with an empty verbal head. This empty verbal head is availible in verb movement analyses and the verb movement analysis blends nicely with the analysis of multiple frontings (Müller, 2005). So, we assume that it is justified to treat these closely related Germanic languages in a parallel way and hence assume a verb movement analysis for Danish as well.

The analysis is sketched in Figure 6 on the next page. The verb *læser* is mapped into a verb that selects for a saturated verbal projection (an S) that contains a verbal trace (represented as '//V'). The DSL feature that is used to represent information about the missing verb is a head feature and hence the information is percolated through the tree to the verb trace. In the verb trace the DSL value is shared with the LOCAL value of the trace and hence the verb trace has the same LOCAL value as the verb in initial position. In the case of our example this means that the verb trace selects for an NP via COMPS and for another one via SPR. The verb trace forms a VP with its complement. This VP is modified by *ikke* ('not') and afterwards combined with its subject in a *head-specifier-phrase*. The subject is a trace and the information about the missing

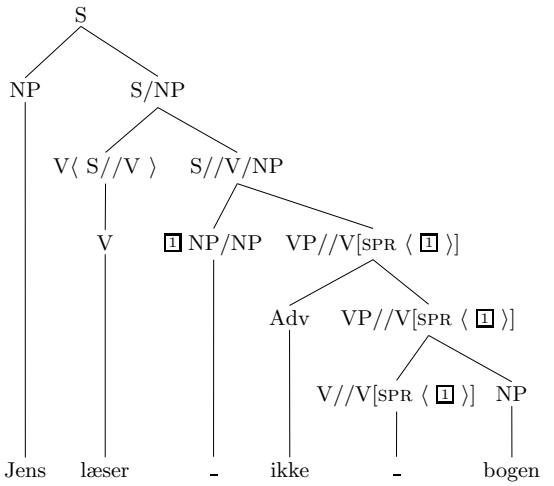


Fig. 6. Analysis of the V2 sentence *Jens læser ikke bogen.* ('Jens reads not book.DEF

constituent is percolated up to the mother nodes until it is finally bound off by the element in the prefield.

Danish is a language with rather fixed constituent order. The correct order of the objects in sentences with ditransitive verbs is guaranteed by combining the verb with its argument in order of their obliqueness, that is, the verb is combined with the IO first and the result of this combination is combined with the DO. However, there are languages with much freer constituent order. For instance, languages like German basically allow the arguments to be combined in any order with the verb. The Head Complement Schema of German will not be sensitive to the ordering of the complements on the COMPS list but allows the combination of a verb or verbal projection with any of the elements on its COMPS list that has not been saturated yet. While it is easy to enforce the relative order of a head with respect to its arguments and adjuncts in a system with binary branching structures, it is not obvious how LP constraints that order the complements relative to each other can be enforced. For instance there is a tendency for short elements (for instance pronouns) to precede heavy constituents. Of course one way of enforcing the order among coarguments is to licence all possible orderings in the lexicon (Uszkoreit, 1986) but this would result in a combinatorical explosion in the lexicon and spurious ambiguities that have to be excluded by ad hoc stipulations (Müller, 2004, p. 217–218). The alternative is to have a representation that corresponds to the linearization domain that one would have if one assumed flat structures. Therefore we assume a list in which all dependents of a head are inserted (the DOMAIN list of Reape (1994)). However, our approach is more restrictive than Reape's proposal in not allowing discontinuous constituents.

4.2 Object Shift as Alternative Mapping to Valence Features

The key feature of our analysis of object shift is an alternative mapping to the valence features. We assume that weak pronouns may be mapped to the SPR list rather than to the COMPS list, which would be their usual place. The following lexical item shows the CAT value for the verb *læser* ('to read'):

(26) CAT value for *læser* ('to read') with both arguments mapped to SPR:

$$\begin{bmatrix} \text{SPR} & \boxed{1} \oplus \boxed{2} \\ \text{COMPS} & \langle\rangle \\ \text{ARG-ST} & \boxed{1}\langle\ \text{NP}\ \rangle \oplus \boxed{2}\langle\ \text{NP}\ \rangle \end{bmatrix} \wedge \boxed{2} = \text{list of weak pronouns}$$

Rather than just mapping the first element of the ARG-ST list onto the SPR list, both arguments are mapped to the SPR list. Note that we do not claim that both arguments are subjects. Any properties that are specific to subjects and do not hold of all members of the SPR list have to be stated as constraints on the first element of the ARG-ST list in our approach. Note also that there are earlier proposals in the HPSG framework that suggested listing the subject in certain languages on the COMPS list (Borsley, 1989a for Welsh; Pollard, 1996 and following Pollard almost all researchers working on German) and similarly there are other analysis that map more than one argument to the valence list that usually contains a single subject (see Grover 1995 for an analysis of missing object constructions in English and Hahn 2012 for an analysis of so-called broad subjects in Arabic. We are also aware of the fact that predicative NPs require a determiner and a subject which they predicate over. The determiner and the subject are usually selected via different valence features. For predicative constructions, we follow Pollard (1996) and Kiss (1992) and assume a head feature SUBJ. While elements in SPR can be combined with their head in principle, this is never possible for elements in the SUBJ list, since there is no schema that refers to this head feature. See Müller 2009 for details on Predication.

The analysis of the example with a shifted pronoun is parallel to what we saw in Figure 6. The only difference is that the object is not realized as a complement but as a specifier. The respective analysis is shown in Figure 7. The fact that *læser* starts out as a VP may seem strange, but VP is just a shorthand for a verbal object with an empty COMPS list. As was shown in (26), *læser* has both arguments in the SPR list. The V1 rule licences a verbal item that selects for a fully saturated clausal projection with a verbal trace that has the properties of *læser*, that is, a verbal trace with two elements in the SPR list and an empty COMPS list. Since the information about the missing verb (the DSL value) is a head feature it is present at the verbal trace as well and since the DSL value of the verbal trace is identified with the LOCAL value of the trace, it is ensured that the verbal trace has the right properties. The adverb *ikke* selects for a VP and the combination of adverb and verbal trace can be combined with the two specifiers. The first specifier is the shifted object and the second specifier is a trace of the subject, which is bound off later in a head-filler structure.

It remains to be explained why the adverb cannot combine with a projection that consists of the VP and one specifier as in Figure 8. This structure is ruled out

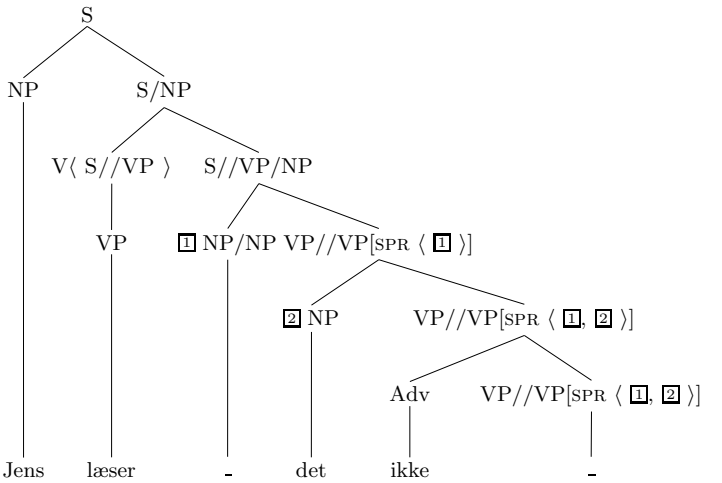


Fig. 7. Analysis of the sentence *Jens læser det ikke.* with object shift with a transitive verb.

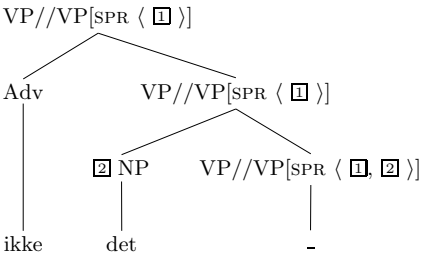


Fig. 8. A structure that is ruled out by a linearization constraint requiring the weak pronoun to precede the adverb

by a linearization constraint that requires shifted pronouns to precede adverbials. As was explained in Section 4.1, we assume that every head has a linearization domain into which the non-head sisters are inserted. Therefore the weak pronoun *det* ('it') and the adverbial *ikke* ('not') are in the same linearization domain and their relative order can be enforced by an LP constraint. Hence, the whole structure in Figure 8 is ruled out, while the one in Figure 7 does not violate the linearization constraint.

An alternative way of ruling out the structure in Figure 8 would be to require that the VP that the adverb combines with does not have any specifiers realized yet. This can be done easily in versions of HPSG that keep saturated arguments on the valence lists (Meurers, 1998; Przepiórkowski, 1999; Müller, 2008; Bender, 2008). Due to space limitations we did not introduce this concept here.

4.3 Shifting and Prepositional Objects

As was shown in Section 2, prepositional objects do not shift and neither do NPs inside of prepositional objects. This is explained by our analysis, since appart from the subject only light pronominals can be mapped to the SPR list.[4] So for the verb *arbejder* ('to work') there is only one mapping possible:

(27)
$$\begin{bmatrix} \textsc{spr} & \boxed{1}\langle\ \mathrm{NP}\ \rangle \\ \textsc{comps} & \boxed{2}\langle\ \mathrm{PP}[på]\ \rangle \\ \textsc{arg-st} & \boxed{1}\langle\ \mathrm{NP}\ \rangle \oplus \boxed{2}\langle\ \mathrm{PP}[på]\ \rangle \end{bmatrix}$$

Since complements licensed by the Head Complement Schema have to be realized to the right of the verb (or verb trace), it is clear that full PPs cannot precede the verb or the negation. This explains the ungrammaticality of (14a). For the same reasons sentences like (14b), repeated here as (28), are ruled out: There is no way for the NP object of the preposition to get into the SPR list of the verb and hence it cannot be realized to the left of the negation. The NP argument of the preposition can be extracted but then it has to be realized in a Head-Filler configuration in the prefield.

(28) * Vi venter [dig] ikke [på].
we wait you not for
Intended: 'We are not waiting for you.'

4.4 Partial Fronting

We assume passive and perfect auxiliaries to be raising verbs that just take over the SPR list of the verb that they embed. Auxiliaries are assigned the following argument structure:

(29) argument structure of the passive and perfect auxiliaries:
$$\begin{bmatrix} \textsc{arg-st} & \boxed{1} \oplus \langle\ \mathrm{VP}[\textsc{spr}\ \boxed{1}\ ]\ \rangle \end{bmatrix}$$

This argument structure is mapped to SPR and COMPS in the following way:

(30) argument structure and valence of the passive and perfect auxiliaries:
$$\begin{bmatrix} \textsc{spr} & \boxed{1} \\ \textsc{comps} & \langle\ \mathrm{VP}[\textsc{spr}\ \boxed{1}\ ]\ \rangle \\ \textsc{arg-st} & \boxed{1} \oplus \langle\ \mathrm{VP}[\textsc{spr}\ \boxed{1}\ ]\ \rangle \end{bmatrix}$$

[4] It remains an open question why PPs cannot shift in Icelandic. Icelandic does allow shifting of full NPs and therefore a constraint on weakness could not be assumed to rule out the shifting of PPs (Engels and Vikner, 2012, p. 19). Engels and Vikner (2012, p. 76) suggest an OT constraint StayBranchNoCase that says that branching constituents that do not get case must not be moved. This is basically a stipulation of the observable facts and of course we can stipulate an analogous constraint.

The example in (17), which is repeated as (31) for convenience, can then be analyzed as shown in Figure 9 on page 85.

(31) Kysset har jeg hende ikke.
kissed have I her not
'I have not kissed her.'

In (31) the object of *kysset* ('kissed') is shifted. This means that the analysis of (31) involves a lexical item for the participle that has an empty COMPS list and two elements on the SPR list. As far as the valence features are concerned, this is parallel to the lexical item for *læser* with a shifted object that was given in (26) on page 81. The difference between *læser* and *kysset* ist that the former is a finite verb and hence has the VFORM value *finite*, while the latter is a past participle and therefore has the VFORM value *perf*. The respective specification of *kysset* is provided in (32):

(32) CAT value for past participle *kysset* ('kissed') with both arguments mapped to SPR:

$$\left[\begin{array}{ll} \text{HEAD} & \left[\begin{array}{l}\text{VFORM } \textit{perf} \\ \textit{verb}\end{array}\right] \\ \text{SPR} & \boxed{1} \oplus \boxed{2} \\ \text{COMPS} & \langle\rangle \\ \text{ARG-ST} & \boxed{1}\langle\ \text{NP}\ \rangle \oplus \boxed{2}\langle\ \text{NP}\ \rangle \end{array}\right] \wedge \boxed{2} = \text{list of weak pronouns}$$

The perfect auxiliary *have* ('to have') selects for a VP with the VFORM value *perf*.

Since the lexical item for *kysset* has an empty COMPS list, it can function as the VP complement of the auxiliary. In the analysis of (31) the VP argument of the auxiliary is realized in the prefield. The VP in the prefield is connected to an extraction trace that functions as the complement of the verb trace. The verb trace has the same syntactic properties as the auxiliary in initial position, that is, it selects for a VP and attracts the SPR list from this VP in the way that was depicted in (30). The result of combining the verb trace and the VP trace is a VP that has two elements in its SPR list. This VP is combined with the negation and after this the two specifiers are realized.

There is a language particular fact about Danish that has not been mentioned so far. Partial fronting is possible with single verbs only. So either a full VP is fronted as in (33a) or a lexical verb as in (33b). In the latter case all objects of the verb have to be shifted.

(33) a. Foræret Anne bogen har Peter ikke.
given.as.a.present Anne book.DEF has Peter not
'Peter has not given Anne the book as a present.'

b. Foræret har Peter hende den ikke.
given.as.a.present has Peter her it not
'Peter has not given it as a present to her.'

Fronting verbs with some of their arguments in the fronted VP is ungrammatical:

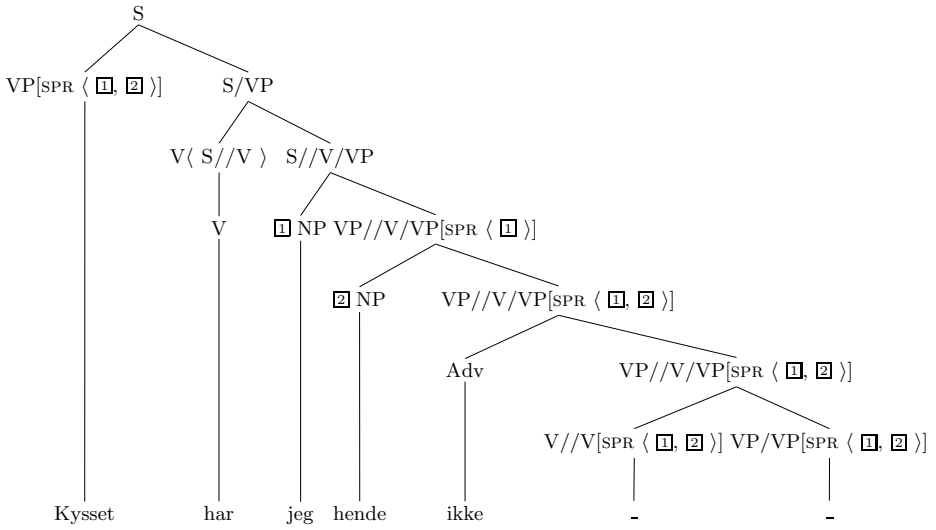


Fig. 9. Object shift with perfect tense and partial VP fronting

(34) * Foræret bogen har Peter hende ikke.
given.as.a.present book.DEF has Peter her not
Intended: 'Peter has not given her the book as a present.'

This can be captured by the following constraint on head filler phrases:[5]

$$\text{(35)}\quad \left[\begin{array}{l}\textsc{non-head-dtrs}\ \left\langle \left[\textsc{synsem|loc|cat}\ \left[\begin{array}{ll}\textsc{head} & \textit{verb}\\ \textsc{spr} & \langle\ [\ ]\ \rangle \oplus \textit{ne\_list}\end{array}\right]\right]\right\rangle \\ \textit{head-filler-phrase}\end{array}\right] \Rightarrow$$

$$\left[\textsc{non-head-dtrs}\ \left\langle \left[\textsc{synsem|lex}\ +\right]\right\rangle\right]$$

This constraint says: If the filler daughter has more than one element in the SPR list (that is, we have a case of object shift and hence a partial VP), the filler daughter has to be LEX+, that is, a lexical verb. We assume that the Head-Argument Schema specifys the LEX value of the mother node to be '–' and hence combinations of verbs with one ore more of their dependents would be ruled out by (35).

The constraint does not affect frontings of complete VPs, which are possible in Danish. A full VP has exactly one element in the SPR list and hence does not match the antecedent of the implication in (35).

[5] Engels and Vikner (2012, p. 109) discuss Danish data similar to Swedish data from Fox and Pesetzky 2005, which is discussed below in (37). These examples contain ditransitive verbs with one argument fronted and one left behind. We do not find the Danish examples acceptable. Speakers who admit such examples do not have the constraint in (35) in their grammars.

4.5 Holmberg's Generalization

The fact that object shift is only possible if the left-most element is shifted can be captured by two LP constraints, which may be surprising at first glance.[6] The constraints are given in (36):

(36) a. V < Pron[¬nom, weak+]
b. IO < DO

(36a) ensures that the pronoun may not be shifted over a verb and hence for shifting to be possible, the finite verbs have to be inverted and non-finite verbs have to be extracted. In a first attempt to capture the data, we formulated a constraint that required that heads with multiple specifiers that enter a head-specifier structure are required to be phonologically empty. While this is sufficient to rule out sentence like (7b), it does not capture that the fronting of non-finite verbs is required as well. In order to explain the ungrammaticality of sentences like (16b) we had to state a separate constraint to the effect that heads that select for more than one specifier and a VP require their VP argument to be extracted. That is, the fact that weak pronouns cannot be reordered across verbs was stated as a conjunction of two rather specific constraints that affected two different types of 'movement': head movement and non-head movement. Instead of the two complex constraints that we proposed earlier, we now use the linearization rule in (36a), which captures the phenomenon rather directly.

In addition to the LP statement in (36a), we assume the constraint in (36b), which requires the IO to shift or to extract for DO shift to be possible. In an earlier version of our analysis we assumed that a prefix of the ARG-ST list has to be mapped to SPR. If extracted elements are not mapped to the valence features, it follows that the DO cannot be mapped to the SPR list unless the IO is either mapped to SPR or extracted. While this is a rather nice approach for Danish, it does not extend to Swedish. As the following examples by Fox and Pesetzky (2005, p. 25) show, Swedish is more liberal than Danish in allowing partial frontings like (37):

(37) a. ? Get henne har jag den inte.
given her have I it not
b. * Get den har jag henne inte.
given it have I her not

The point about these examples is that an account that relies on mapping prefixes of ARG-ST onto SPR would predict the oposit judgement, that is, (37a) should be impossible and (37b) marked but possible. What is needed to account for (37a) is the possibility to map the DO to SPR and realize the IO in the fronted VP. So, the mapping from ARG-ST to SPR has to allow more than just prefixes

[6] See also Engdahl, Andréasson and Börjars 2004 for an analysis in Optimality Theory that relies on linearization constraints. While it is possible to introduce into HPSG ranked linearization constraints in the spirit of Uszkoreit 1987, Section 3.1, we do not assume factorial typology.

of ARG-ST to appear in the SPR list. For (37b) one would assume, as we do for Danish, that it is ruled out since the DO is serialized before the IO and hence violates the linearization rule in (36b).

Note, that the DO is part of the VP *get den*. In order for the linearization rule in (36b) to apply to IO and DO in (37b), the IO and DO have to be members of the same linearization domain. We assume that VP complements are domain unioned into the domain of the auxiliary (Reape, 1994). For (37b) this results in a linearization domain that contains the linguistic objects that correspond to the words of this sentence. In this respect our approach is rather similar to the one suggested by Bjarre, but it differs in not allowing for discontinuous constituents. Therefore the VP *get den* is always realized continuously and problems like those that were discussed above are avoided.

It is also important to note here that traces do not contribute anything to the linearization domains. If they would, sentences like (11) were ruled out since the DO *den* preceedes the trace of the IO $\__i$.

5 Conclusion

In this chapter we have presented an analysis of object shift in Danish without assuming any kind of movement or dislocation and without reducing object shift to a mere linearization phenomenon. We have suggested that lexical pronouns are members of the SPR list of their verbs. The analysis of auxiliaries and partial fronting involves argument attraction as in analyses of German partial verb phrase fronting (Müller, 1996, 1999, 2002; Meurers, 2000), but the arguments are attracted from the SPR list rather than from the COMPS list.

Linearization constraints account for the observations that have been summarized as Holmberg's Generalization.

The analysis has been partly implemented in the TRALE system (Meurers, Penn and Richter, 2002; Penn, 2004; Müller, 2007a) as part of a grammar fragment of Danish which uses a core grammar for German, Mandarin Chinese, Persian, Maltese, and Yiddish. See Müller 2013 on the CoreGram project. The respective grammars can be downloaded at http://hpsg.fu-berlin.de/Projects/CoreGram.html. The Danish grammar is described in Müller and Ørsnes In Preparation.

References

Bender, E.M.: Radical Non-Configurationality without Shuffle Operators: An Analysis of Wambaya. In: Müller, S. (ed.) Proc. Int. Conf. Head-Driven Phrase Struct. Grammar, pp. 6–24. CSLI Publications, Stanford (2008)

Bjerre, T.: Object Positions in a Topological Sentence Model for Danish – a Linearization-Based HPSG Approach. Presentation at Ph.D.-course at Sandbjerg, Denmark (2006)

Borsley, R.D.: An HPSG Approach to Welsh. J. of Linguistics 25, 333–354 (1989a)

Borsley, R.D.: Phrase-Structure Grammar and the Barriers Conception of Clause Structure. Linguistics 27, 843–863 (1989b)

Engdahl, E., Andréasson, M., Börjars, K.: Word Order in the Swedish Midfield – an OT Approach. In: Karlsson, F. (ed.) Proceedings of the 20th Scandinavian Conference of Linguistics, Helsinki, January 7-9, University of Helsinki, Department of General Linguistics, Helsinki (2004)

Engels, E., Vikner, S.: Scandinavian Object Shift and Optimality Theory. Ms, University of Aarhus (2012)

Erteschik-Shir, N.: Sound Patterns of Syntax: Object Shift. Theoretical Linguistics 31(1-2), 47–93 (2005)

Fox, D., Pesetzky, D.: Cyclic Linearization of Syntactic Structure. Theoretical Linguistics 31(1-2), 1–45 (2005)

Grover, C.: Rethinking Some Empty Categories: Missing Objects and Parasitic Gaps in HPSG. Ph. D.thesis, University of Essex (1995)

Hahn, M.: Arabic Relativization Patterns: A Unified HPSG Analysis. In: Müller, S. (ed.) Proceedings of the 19th International Conference on Head-Driven Phrase Structure Grammar, Chungnam National University Daejeon, pp. 144–164. CSLI Publications, Stanford (2012)

Hellan, L.: On Pronominal Clitics in Norwegian. In: Holmberg, A., Hedlund, C. (eds.) Proceedings of the XIVth Scandinavian Conference of Linguistics, University of Gothenberg, pp. 1–14 (1994)

Holmberg, A.: Remarks on Holmberg's Generalization. Studia Linguistica 53(1), 1–39 (1999)

Kathol, A., Pollard, C.J.: Extraposition via Complex Domain Formation. In: Proceedings of the 33rd Annual Meeting of the ACL, Boston (1995)

Kiss, T.: Variable Subkategorisierung. Linguistische Berichte 140, 256–293 (1992)

Kiss, T.: Infinite Komplementation. Neue Studien zum deutschen Verbum infinitum. Niemeyer, Tübingen (1995)

Kiss, T., Wesche, B.: Verb Order and Head Movement. In: Herzog, O., Rollinger, C.-R. (eds.) LILOG 1991. LNCS, vol. 546, pp. 216–242. Springer, Heidelberg (1991)

Meurers, D.: Lexical Generalizations in the Syntax of German Non-Finite Constructions. Arbeitspapiere des SFB 340 No. 145, Universität Tübingen (2000)

Meurers, W.D.: Raising Spirits and Assigning Them Case, vortrag auf dem Workshop Current Topics in Constraint-Based Theories of Germanic Syntax (1998)

Meurers, W.D., Penn, G., Richter, F.: A Web-Based Instructional Platform for Constraint-Based Grammar Formalisms and Parsing. In: Radev, D., Brew, C. (eds.) Effective Tools and Methodologies for Teaching NLP and CL, pp. 18–25 (2002)

Mikkelsen, L.: On Prosody and Focus in Object Shift. Syntax 14(3), 230–264 (2011)

Müller, S.: Yet another Paper about Partial Verb Phrase Fronting in German. In: Proceedings of Coling 1996, pp. 800–805. ACL, Copenhagen (1996)

Müller, S.: Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche. Niemeyer, Tübingen (1999)

Müller, S.: Complex Predicates: Verbal Complexes, Resultative Constructions, and Particle Verbs in German. CSLI Publications, Stanford (2002)

Müller, S.: Continuous or Discontinuous Constituents? A Comparison between Syntactic Analyses for Constituent Order and Their Processing Systems. Research on Language and Computation 2(2), 209–257 (2004)

Müller, S.: Zur Analyse der scheinbar mehrfachen Vorfeldbesetzung. Linguistische Berichte 203, 297–330 (2005)

Müller, S.: The Grammix CD Rom. A Software Collection for Developing Typed Feature Structure Grammars. In: King, T.H., Bender, E.M. (eds.) Grammar Engineering across Frameworks 2007. CSLI Publications, Stanford (2007a)

Müller, S.: Head-Driven Phrase Structure Grammar: Eine Einführung, 1st edn. Stauffenburg Verlag, Tübingen (2007b)
Müller, S.: Depictive Secondary Predicates in German and English. In: Hentschel, G., et al. (eds.) Secondary Predicates in Eastern European Languages and Beyond, pp. 255–273. BIS-Verlag, Oldenburg (2008)
Müller, S.: On Predication. In: Müller, S. (ed.) Proceedings of the 16th International Conference on Head-Driven Phrase Structure Grammar, pp. 213–233. CSLI Publications, Stanford (2009)
Müller, S.: The CoreGram Project: Theoretical Linguistics. Theory Development and Verification. Ms. Freie Universität, Berlin (2013)
Müller, S.: HPSG – A Synopsis. In: Alexiadou, A., Kiss, T. (eds.) Syntax – Ein internationales Handbuch zeitgenössischer Forschung, 2nd edn. Walter de Gruyter Verlag, Berlin (to appear)
Müller, S., Ørsnes, B.: Danish in Head-Driven Phrase Structure Grammar. Language Science Press, Berlin (in preparation)
Penn, G.: Balancing Clarity and Efficiency in Typed Feature Logic Through Delaying. In: Scott, D. (ed.) Proceedings of the 42nd Meeting of the Association for Computational Linguistics, Barcelona, Spain, pp. 239–246 (2004)
Pollard, C.J.: On Head Non-Movement. In: Bunt, H., van Horck, A. (eds.) Discontinuous Constituency, pp. 279–305. Mouton de Gruyter, Berlin (1996) (published version of a Ms. dated January 1990)
Pollard, C.J., Sag, I.A.: Anaphors in English and the Scope of Binding Theory. Linguistic Inquiry 23(2), 261–303 (1992)
Pollard, C.J., Sag, I.A.: Head-Driven Phrase Structure Grammar. University of Chicago Press, Chicago (1994)
Przepiórkowski, A.: On Case Assignment and "Adjuncts as Complements". In: Webelhuth, G., Koenig, J.-P., Kathol, A. (eds.) Lexical and Constructional Aspects of Linguistic Explanation, pp. 231–245. CSLI, Stanford (1999)
Reape, M.: Domain Union and Word Order Variation in German. In: Nerbonne, J., Netter, K., Pollard, C.J. (eds.) German in Head-Driven Phrase Structure Grammar, pp. 151–198. CSLI Publications, Stanford (1994)
Uszkoreit, H.: Linear Precedence in Discontinuous Constituents: Complex Fronting in German. Report No. CSLI-86-47. CSLI, Stanford (1986)
Uszkoreit, H.: Word Order and Constituent Structure in German. CSLI Publications, Stanford (1987)
Vikner, S.: Object Shift. In: Everaert, M., et al. (eds.) The Blackwell Companion to Syntax, pp. 392–436. Blackwell Publishing Ltd., Oxford (2006)

Cognitive and Sub-regular Complexity

James Rogers[1], Jeffrey Heinz[2], Margaret Fero[1], Jeremy Hurst[1], Dakotah Lambert[1], and Sean Wibel[1]

[1] Earlham College, Richmond IN 47374, USA
[2] University of Delaware, Newark DE 19716, USA

Abstract. We present a measure of cognitive complexity for subclasses of the regular languages that is based on model-theoretic complexity rather than on description length of particular classes of grammars or automata. Unlike description length approaches, this complexity measure is independent of the implementation details of the cognitive mechanism. Hence, it provides a basis for making inferences about cognitive mechanisms that are valid regardless of how those mechanisms are actually realized.

Keywords: Cognitive complexity, sub-regular hierarchy, descriptive complexity, phonological stress.

1 Introduction

Identifying the nature of the cognitive mechanisms employed by various species, and the evidence which helps determine this nature, are fundamental goals of cognitive science. The question of the relative degree of difficulty of distinguishing (proto-) linguistic patterns has received a considerable amount of attention in recent Artificial Grammar Learning (AGL) research [1,2], as well as in current research in phonology [3,4]. In the AGL research, as in the phonological research, the complexity of the learning task has been central. This in no small part depends on the complexity of the patterns being learned.

This paper studies the pattern complexity of subclasses of the class of regular stringsets[1] from a model-theoretic perspective, which has its roots in the seminal work of McNaughton and Papert [5] (and, ultimately, Büchi [6] and Elgot [7]). An important aspect of this analysis is that it is *independent* of any particular representation. We argue that descriptive complexity of this model-theoretic sort provides a more consistent measure of complexity than typical complexity measures based on minimum description length. More importantly, we show how this notion of cognitive complexity can provide concrete evidence about the capabilities of the recognition mechanism that is valid for all mechanisms, regardless of their implementation.

[1] To minimize confusion between natural and formal languages will generally use the term "stringset" to denote a set of strings rather than the more traditional "language", except that we will use the original terminology when referring by name to concepts defined elsewhere in the literature.

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 90–108, 2013.

This complexity analysis is exemplified with stress patterns in the world's languages. Stress patterns are rules that govern which syllables are emphasized, or stressed, in words. The reason we use stress patterns to illustrate the complexity hierarchy is because the cross-linguistic typology of stress patterns has been well-studied [8,9] and because finite-state representations of these patterns already exist [10,11].

In the next section, we explain why approaches based on minimum description length fail to provide an adequate notion of cognitive complexity in these domains. We then (Section 3) develop a model-theoretic foundation for building hierarchies of descriptive complexity that do provide a consistent and useful notion of cognitive complexity. In Section 4 we develop such a hierarchy based on adjacency. This is the Local hierarchy of McNaughton and Papert, although our presentation is more abstract and provides a basis for the generalizations that follow. Section 4.1 treats the Strictly-Local sets. We do this in greater detail than we do in the subsequent sections, providing the general framework and allowing us to focus on specific variations at the higher levels of the hierarchy. Sections 4.2, 4.3 and 4.4 treat the Locally Testable, Locally Threshold Testable and Star-Free sets, respectively.

In Section 5 we repeat this structure for a hierarchy based on precedence rather than adjacency. The Piecewise Testable level (Section 5.2) of this hierarchy is well known, but the Strictly Piecewise level (Section 5.1) has only been studied relatively recently. The two hierarchies converge at the level of the Star-Free sets.

We conclude with a brief description of our current work applying these results to the phonology of stress patterns.

While most of the language-theoretic details we present here are not new, we present them within a more general framework that provides better insight into the common characteristics of and parameters of variation between the classes. Our main contribution, however, is the use of these descriptive hierarchies as the basis of a measure of cognitive complexity capable of providing clear and reliable insights about obscure cognitive mechanisms.

2 Cognitive Complexity of Simple Patterns

The formal foundation for comparisons of the complexity of patterns has primarily been the information theoretic notion of minimum description length. This compares the total number of bits needed to encode a model of computation—for our purposes, a general recognition algorithm—plus the number of bits required to specify the pattern with respect to that model.

Our focus, here, is on patterns that can be described as regular stringsets. While there are many computational models that we might choose, we will focus on a few standard ones: Regular Grammars, Deterministic Finite State Automata (DFAs) and Regular Expressions [12]. All of these computational models are equivalent in their formal power and there is no significant difference in the

size of the encodings of the computational models themselves[2] (the recognition algorithms), so there is no *a priori* reason to prefer one over another. The question is how well the relative size of the descriptions of patterns with respect to a given computational model reflects pre-theoretic notions of the relative complexity of processing the patterns. So we will concentrate on comparing complexity within a given computational model. This allows us to ignore the encoding of the computational model itself.

One does not have to look far to find examples of pairs of stringsets in which these three computational models disagree with each other about the relative complexity of the stringsets. Moreover each get the apparent relative complexity wrong on one or another of these examples. Figure 1 compares minimal descriptions, with respect to each of these computational models, of the set of strings of 'A's and 'B's that end with 'B', which we will refer to as EndB, and minimal descriptions of the set of strings of 'A's and 'B's in which there is an odd number of occurrences of 'B', which we will refer to as OddB.[3] Thinking simply in terms of what a mechanism has to distinguish about a string to determine whether it meets the pattern or not, what properties of strings distinguish those that fit the pattern from those that do not, EndB is clearly less complex than OddB. In the first case, the mechanism can ignore everything about the string except the last (or most recent) symbol; the pattern is 1-Definite, i.e., fully determined by the last symbol in the string. In the second, it needs to make its decision based on the number of occurrences of a particular symbol modulo two; it is properly regular in the sense that it is regular but not star-free (see Section 4.4).

The Regular Grammars get this intuition right, as do the regular expressions. The DFAs, on the other hand, differ only in the label of two transitions. There is no obvious attribute of the DFAs, themselves, that distinguishes the two.

Figure 2 compares minimal descriptions of the set of strings of 'A's and 'B's in which there is at least one occurrence of 'B's (SomeB) with minimal descriptions of strings in which there is exactly one occurrence of 'B' (OneB). Here, there can be no question of the relative complexity of the two stringsets: in order to recognize that exactly one 'B' occurs, one must be able to recognize that at least one 'B' occurs; in order to generate a string in which exactly one 'B' occurs, one must be able to generate a string with at least one 'B'. But for both the Regular Grammars and the Regular Expressions the size of the description of SomeB is greater than that of OneB. If we insist that DFAs be total, in the sense of having a total transition function—an out edge from each state for each symbol of the alphabet—then the minimal DFA for OneB is larger than that for SomeB. But if we trim the DFAs, deleting states that cannot fall on paths from the start state to an accepting state, the DFAs are identical except that OneB actually requires one fewer transition.

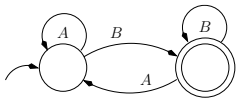
[2] Note that what is in question here is the encoding of the model, a representation of, say, a Turing machine program to process the descriptions, *not* the descriptions themselves. The encodings of the models vary in size by at most a constant.

[3] In the case of the DFAs, the minimality is easy to verify. For the other computational models minimality could be verified by enumeration, although this seems excessive.

Sequences of 'A's and 'B's which end in 'B' (EndB)

Regular Grammar: $S_0 \longrightarrow AS_0,\ S_0 \longrightarrow BS_0,\ S_0 \longrightarrow B$

DFA:

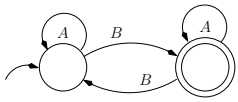


Regular Expression: $(A + B)^*B$

Sequences of 'A's and 'B's which contain an odd number of 'B's (OddB)

Regular Grammar: $S_0 \longrightarrow AS_0,\ S_0 \longrightarrow BS_1,$
$S_1 \longrightarrow AS_1,\ S_1 \longrightarrow BS_0,\ S_1 \longrightarrow \varepsilon$

DFA:



Regular Expression: $(A^*BA^*BA^*)^*A^*BA^*$

Fig. 1. Minimal descriptions: strings which end in 'B' vs. strings with an odd number of 'B's

The point of these comparisons is that, even with just these four extremely simple patterns, all of these computational models disagree with each other about relative complexity and each of them gets some of the relative complexities wrong. Relative information theoretic complexity, at this level, depends on the choice of computational model and none of these computational models consistently reflects the actual pre-theoretic relative complexity of distinguishing the patterns.

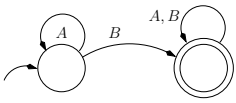
There are many ways to describe regular stringsets beyond the ones considered here [13] so the above is not a deductive proof that no such computational model exists. While searching for an appropriate computational model is one line of research, this program faces a fundamental limitation. Encoding complexity with respect to a particular computational model severely limits the validity of conclusions we might draw about actual cognitive mechanisms from relative complexity results. In the domain of language, the structure of the cognitive mechanisms that an organism uses to recognize a pattern is hotly debated. If a complexity measure is going to provide useful insights into the characteristics of the cognitive mechanisms that can distinguish a pattern, it is an advantage if it is agnostic about the operational details of the mechanisms themselves.

The alternative to searching for a computational model is to develop an abstract measure of complexity. This measure should be invariant across all possible cognitive mechanisms and depend only on properties that are necessarily common to all computational models that can distinguish a pattern. Such a measure

Sequences of 'A's and 'B's which contain at least one 'B' (SomeB)

Regular Grammar: $S_0 \longrightarrow AS_0,\ S_0 \longrightarrow BS_1,$
$S_1 \longrightarrow AS_1,\ S_1 \longrightarrow BS_1,\ S_1 \longrightarrow \varepsilon$

DFA:

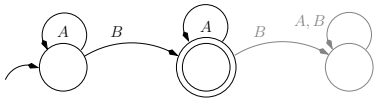


Regular Expression $A^*B(A+B)^*$

Sequences of 'A's and 'B's which contain exactly one 'B' (OneB)

Regular Grammar: $S_0 \longrightarrow AS_0,\ S_0 \longrightarrow BS_1,$
$S_1 \longrightarrow AS_1,\ S_1 \longrightarrow \varepsilon$

DFA:



Regular Expression: A^*BA^*

Fig. 2. Minimal descriptions: strings that contain at least one 'B' vs. strings that contain exactly one 'B'

has to be based on intrinsic properties of the (generally infinite) set of stimuli that match a pattern. We will take as the basis of the measure the properties of the stimuli that distinguish those that satisfy a pattern from those that do not. These are the things to which a cognitive mechanism needs to be sensitive—the properties of strings it must be able to detect—in order to correctly classify a stimulus with respect to a pattern.

3 Cognitive Complexity from First Principles

At the most fundamental level, we need to decide what kind of objects (entities, things) we are reasoning about and what relationships between them we are reasoning with. Since we are focusing on linguistic-like behaviors, we will assume that the cognitive mechanisms of interest perceive (process, generate) linear sequences of events.[4]

These we can model as strings, linear sequences of abstract symbols, which we will take to consist of a finite discrete linear order (isomorphic to an initial

[4] Historically, the term "event" has referred to the entire sequence. But, in general the overall pattern may be hierarchically structured, i.e., sequences of subsequences each of which would, itself, be an event. So the distinction, here, seems to be spurious and we will refer to the elements of any sequence as an event.

segment of the natural numbers) that is labeled with an alphabet of events. The labeling partitions the domain of the linear order into subsets, each the set of positions at which some event occurs. Representing these as ordinary relational structures [14], we get word models of Figure 3, in which we use the symbol '$\lhd$' to denote successor (adjacency) and '$\lhd^+$' to denote less-than (precedence). Concatenation with respect to these models is just the ordered sum of the linear orders.[5] We take these models simply to *be* strings; we use no other formalization.

We will distinguish three classes of models: (+1)—models which include only successor (restricted to be successor with respect to some linear order), (<)—models which include only less-than, and models which include both (word models in general).

4 Adjacency—Substrings

The first hierarchy of complexity classes we will consider is based on reasoning about adjacency, in general about substrings, i.e., blocks of consecutive symbols within a string. This gives us a well known sequence of stringset classes, based on generalizations of the *Strictly Local Languages* [5]. We formalize these classes here in a way that will support generalization to stringset classes that are based on other ways of reasoning about strings.

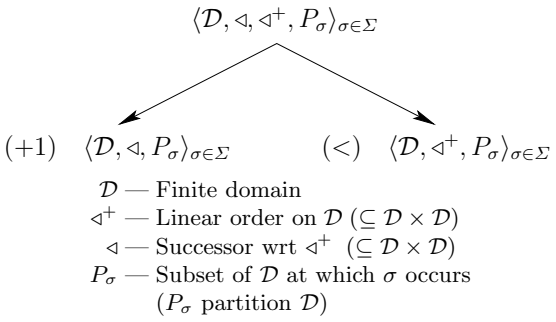


Fig. 3. Word models. (+1) models are the restriction of the general word models to $\lhd$; (<) models are the restriction to $\leq_2$.

All stringsets within these classes are defined in terms of their constituent substrings with the variation between the classes corresponding to how descriptions can be built from those substrings. Traditionally, the substrings that occur within a string are referred to as its *factors* (with respect to concatenation).

[5] That is to say, the concatenation of two word models is the disjoint union of their domains and of their interpretations of the relation symbols extended so that the minimum point of the domain of the right word is the successor of the maximum point of the domain of the left. Note that the empty string is represented by a model with an empty domain, which is usually avoided, but this presents no problems for our applications.

Definition 1 (k-Factor).
v is a factor *of w if $w = uvx$ for some $u, v \in \Sigma^*$.*
*v is a k-*factor *of w if it is a factor of w and $|v| = k$.*
The set of k-factors of a string w is:

$$F_k(w) \stackrel{def}{=} \begin{cases} \{v \in \Sigma^k \mid (\exists u, x \in \Sigma^*)[w = uvx]\} & \text{if } |w| \geq k, \\ \{w\} & \text{otherwise.} \end{cases}$$

This lifts to sets in the standard way $F_k(L) \stackrel{def}{=} \{F_k(w) \mid w \in L\}$.

k-factors are essentially k-grams without probabilities. Note that the set of k-factors of a word w which is shorter than k includes just w, itself. The set of all k-factors over an alphabet Σ is $F_k(\Sigma^*) = \{w \in \Sigma^* \mid |w| \leq k\}$, where $|w|$ denotes the length of w. $F_k(\Sigma^*)$ properly includes $F_{k-i}(\Sigma^*)$ for all $i < k$.

4.1 Strictly Local Sets

Definition 2 (Strictly Local Sets). *A strictly k-Local definition $\mathcal{G}$, over some alphabet Σ, is a set of k-factors over $\Sigma \cup \{\rtimes, \ltimes\}$, where '$\rtimes$' and '$\ltimes$' are new symbols: initial and final markers, respectively.*

$$\mathcal{G} \subseteq F_k(\{\rtimes\} \cdot \Sigma^* \cdot \{\ltimes\})$$

A string w satisfies *$\mathcal{G}$ ($w \models \mathcal{G}$) iff the set of k-factors of $\rtimes \cdot w \cdot \ltimes$ is a subset of $\mathcal{G}$.*

$$w \models \mathcal{G} \stackrel{def}{\iff} F_k(\rtimes \cdot w \cdot \ltimes) \subseteq \mathcal{G}$$

The stringset licensed by a description $\mathcal{G}$ is the set of words that satisfy it.

$$L(\mathcal{G}) \stackrel{def}{=} \{w \mid w \models \mathcal{G}\}$$

A set of strings is Strictly k-local *(*SL_k*) iff it is $L(\mathcal{G})$ for some strictly k-local definition $\mathcal{G}$. It is* Strictly Local *(*SL*) iff it is* SL_k *for some k.*

The expression $\rtimes \cdot w \cdot \ltimes$ denotes w augmented with explicit initial and final markers. A strictly k-local description is the set of k-factors that are licensed to occur in the augmented string. Again, $F_k(\{\rtimes\} \cdot \Sigma^* \cdot \{\ltimes\})$ contains factors of length less than k, but in this case they all begin with '$\rtimes$' and end with '$\ltimes$'. Hence, they license only words of length less than $k - 1$.

The characteristic property of Strictly k-local sets is that they are closed under substitution of suffixes that start with the same $(k-1)$-factor.

Theorem 1 (Suffix Substitution Closure). *A stringset L is strictly k-local iff whenever there is a string x of length $k-1$ and strings w, y, v, and z, such that*

$$w \cdot \overbrace{x}^{k-1} \cdot y \in L \text{ and } v \cdot \overbrace{x}^{k-1} \cdot z \in L \Rightarrow w \cdot \overbrace{x}^{k-1} \cdot z \in L$$

(Sketch of proof:) Closure of SL_k stringsets under substitution of suffixes in this way is nearly immediate. If $w \cdot x \cdot y$ and $v \cdot x \cdot z \in L(\mathcal{G})$ for some SL_k definition $\mathcal{G}$, and $|x| = k - 1$ then

$$F_k(\rtimes \cdot w \cdot x \cdot z \cdot \ltimes) \subseteq F_k(\rtimes \cdot w \cdot x \cdot y \cdot \ltimes) \cup F_k(\rtimes \cdot w \cdot x \cdot y \cdot \ltimes) \subseteq \mathcal{G}$$

For the other direction, suppose that a stringset L is closed under substitution of suffixes that start with the same $(k-1)$-factor. Let $\mathcal{G}_L = F_k(\{\rtimes\} \cdot L \cdot \{\ltimes\})$. That $L \subseteq L(\mathcal{G}_L)$ is immediate by construction. One can show that $L(\mathcal{G}_L) \subseteq L$ by constructing an arbitrary $w \in L(\mathcal{G}_L)$ in stages from strings in L that share successively long prefixes of w, extending the prefix, at each stage, by substitution of suffixes.

Note that this is a *characterization*: every SL_k stringset is closed under substitution of suffixes in this way and every stringset that is SSC-closed for some k can be defined by a SL_k definition.

SL_k and SL, as a whole, are closed under intersection but not union, complement, concatenation or Kleene-$*$ [5].

Example 1. Stress in the language Alawa is governed by two (actually three) phonological rules [11]:

- In words of all sizes, primary stress falls on the penultimate syllable.
- In words of all sizes, there is no secondary stress.

The third rule is implicit in all stress patterns

- Every word has exactly one syllable that receives primary stress.

This pattern is not SL_2 as witnessed by:

$$\rtimes\sigma\,\acute{\sigma}\,\sigma\ltimes \in L_{\text{Alawa}},\ \rtimes\,\acute{\sigma}\,\ltimes \in L_{\text{Alawa}},\ \text{but}\ \rtimes\sigma\,\acute{\sigma}\,\ltimes \notin L_{\text{Alawa}}.$$

On the other hand, we can capture L_{Alawa} with the constraints:

1. Do not permit 3-factors with multiple primary stress.
2. Do not permit unstressed penultimate syllables.
3. Do not permit primary stress to be followed by more than one syllable.
4. Do not permit unstressed monosyllables.
5. Do not permit empty words.

$$\begin{aligned} L_{\text{Alawa}} = L(\ & F_3(\rtimes \cdot \Sigma^+ \cdot \ltimes) \\ & - \{\ \rtimes\acute{\sigma}\acute{\sigma},\ \acute{\sigma}\acute{\sigma}\ltimes,\ \sigma\acute{\sigma}\acute{\sigma},\ \acute{\sigma}\sigma\acute{\sigma},\ \acute{\sigma}\acute{\sigma}\sigma,\ \acute{\sigma}\acute{\sigma}\acute{\sigma}, && (1) \\ & \quad \sigma\acute{\sigma}\ltimes,\ \sigma\sigma\ltimes, && (2) \\ & \quad \acute{\sigma}\sigma\sigma, && (3) \\ & \quad \rtimes\sigma\ltimes, && (4) \\ & \quad \rtimes\ltimes\}\ \ ) && (5) \\ = L(\ & \{\rtimes\sigma\sigma,\ \rtimes\sigma\acute{\sigma},\ \rtimes\acute{\sigma}\sigma,\ \sigma\sigma\sigma,\ \sigma\sigma\acute{\sigma},\ \sigma\acute{\sigma}\sigma, \acute{\sigma}\sigma\ltimes,\ \rtimes\acute{\sigma}\ltimes\}\ ) \end{aligned}$$

Hence $L_{\text{Alawa}} \in \mathrm{SL}_3 - \mathrm{SL}_2$.

The strictly local classes form a proper hierarchy in k.

Theorem 2 (SL-Hierarchy).

$$\mathrm{SL}_1 \subsetneq \mathrm{SL}_2 \subsetneq \mathrm{SL}_3 \subsetneq \cdots \subsetneq \mathrm{SL}_i \subsetneq \mathrm{SL}_{i+1} \subsetneq \cdots \subsetneq \mathrm{SL}$$

(Sketch of proof) The inclusions follow nearly immediately from the definition of an SL_k definition. The separations are easy to obtain using generalizations of the proof that the Alawa stress pattern is not SL_2.

The proper inclusions reflect the intuition that distinguishing a pattern that requires attending to a larger block of symbols is likely to be cognitively more difficult than distinguishing one that can be recognized by attending to smaller blocks.

While every finite stringset is SL_k for some k, there is no k for which SL_k includes all finite stringsets. Note, also, that given fixed k and Σ, there are only finitely many SL_k stringsets. SL_k is learnable in the limit from positive data in the sense of Gold [15]; SL as a whole is not [16].

Example 2. Edlefsen, et al. [17] have categorized the 109 patterns in Heinz's Stress Pattern Database [18]:

9 are SL_2	Abun West, Afrikans, ... Cambodian, ... Maranungku
44 are SL_3	Alawa, Arabic (Bani-Hassan), ...
24 are SL_4	Arabic (Cairene),[6]...
3 are SL_5	Asheninca, Bhojpuri, Hindi (Fairbanks)
1 is SL_6	Icua Tupi
28 are not SL	Amele, Bhojpuri (Shukla Tiwari), Arabic Classical, Hindi (Keldar), Yidin,...

72% are SL, all $k \leq 6$. 49% are SL_3.

This suggests that the majority of stress patterns in natural languages are cognitively very simple, and perhaps even learnable in the limit.

Cognitive Interpretation of SL. It is important to note that the definition of the class SL and its characterization by suffix substition closure make no reference to any computational model of any sort. They are stated purely in terms of the structure of the stringset itself. Members of an SL_k stringset are distinguished from non-members purely on the basis of their k-factors. This assumes nothing about *how* those distinctions might be made by a particular computational mechanism. Any mechanism that can distinguish members of an SL_k stringset from non-members *must* be able, at least, to distinguish strings in this way. Any capabilities they may have beyond that are, in a sense, wasted, at least with respect to that stringset.

This gives us a general characterization of cognitive mechanisms that are capable of recognizing SL_k stringsets.

[6] The formalization of Arabic (Cairene) is controversial. Thomas Graf formalizes this in a way that is properly regular [19].

- Any cognitive mechanism that can distinguish member strings from non-members of a (properly) SL_k stringset must be sensitive, at least, to the length k blocks of consecutive events that occur in the presentation of the string.
- If the strings are presented as sequences of events in time, then this corresponds to being sensitive, at each point in the string, to the immediately prior sequence of $k-1$ events.

Note, again, that the cognitive mechanism is not required to analyze strings in terms of blocks of consecutive events, even if they are presented as sequences of events in time. It just needs to be able to make judgements, at each point in the presentation of the string, that depend on the sequences of $k-1$ events which occur prior to that point. Every regular stringset can be generated by a context-free grammar that is not a regular grammar, that does not analyze the string in terms of contiguous blocks of symbols. Nevertheless, if the stringset is strictly local, it will still need to get the judgements right; the set of all strings that it generates will be closed under substitution of suffixes; it will differ from the set it does not generate only in the blocks of consecutive symbols that occur in the strings.

4.2 Locally Testable Languages

The standard phonological assumption that in every word there is some syllable that receives primary stress [20] is problematic for SL. Letting $\Sigma = \{\sigma, \acute{\sigma}, \grave{\sigma}\}$ (representing unstressed syllables, those with primary stress and those with secondary stress, respectively), this assumption can be described with the following regular expression: $\Sigma^* \acute{\sigma} \Sigma^*$. Note that we are not (yet) ruling out the possiblity that more than one syllable receives primary stress. To see that this is not SL, suppose, for contradiction, that it was. Then it would necessarily be SL_k for some particular k. But then

$$\rtimes \overbrace{\sigma \cdots \sigma}^{k-1} \acute{\sigma} \ltimes,\ \rtimes \acute{\sigma} \overbrace{\sigma \cdots \sigma}^{k-1} \ltimes \in L_{\mathrm{Some}\acute{\sigma}} \quad \text{but } \rtimes \overbrace{\sigma \cdots \sigma}^{k-1} \ltimes \notin L_{\mathrm{Some}\acute{\sigma}}$$

SL patterns cannot, in general, require a factor to occur; they can, at most, forbid factors from occurring. Hence, they cannot enforce the requirement that primary stress occurs unless either the stress is required to fall within a fixed radius of one end of the word (as in the case of Alawa) or the factors preceding the stress can be distinguished in some other way from those following it.

The next level of the Local Hierarchy, the class of *Locally Testable* (LT) languages is the closure of SL under Boolean operations. Since this includes complement, it allows one to require the occurrence of specific factors. Rather than taking LT descriptions to be Boolean combinations of SL descriptions, we use a simple propositional calculus to describe LT sets. This provides a foundation for extending the descriptions to First Order descriptions.

Definition 3 (Local k-expressions). *The logic of Local k-expressions is based on the smallest set including the following forms, with the intended semantics as indicated.*

$$f \in F_k(\rtimes \cdot \Sigma^* \cdot \ltimes) \qquad w \models f \overset{def}{\iff} f \in F_k(\rtimes \cdot w \cdot \ltimes)$$
$$\varphi \wedge \psi \qquad w \models \varphi \wedge \psi \overset{def}{\iff} w \models \varphi \text{ and } w \models \psi$$
$$\neg\varphi \qquad w \models \neg\varphi \overset{def}{\iff} w \not\models \varphi$$

The k-factors serve as our atomic propositions. While these are not devoid of internal structure, they are either a factor of a string or not. Hence, strings can be seen as valuations of the factors in the ordinary propositional sense. The calculus of k-expressions is just an idiosyncratic propositional calculus.

Definition 4 (Locally Testable Sets). *A stringset L over Σ is* k-Locally Testable *iff (by definition) there is some local k-expression φ over Σ (for some k) such that L is the set of all strings that satisfy φ:*

$$L = L(\varphi) \overset{def}{=} \{w \in \Sigma^* \mid w \models \varphi\}$$

A stringset is LT *iff it is* LT_k *for some k.*

Note that SL_k descriptions can be interpreted by local k-expressions:

$$L(\mathcal{G}) = L(\bigwedge_{f_i \notin \mathcal{G}} [\neg f_i])$$

Thus $\mathrm{SL}_k \subsetneq \mathrm{LT}_k$. In particular, SL stringsets are exactly those LT stringsets that can be expressed as conjunctions of negative constraints.

Since strings are, in effect, propositional valuations, the Locally Testable stringsets can be characterized by the fact that they must be the union of finitely many classes of strings that are equivalent with respect to the k-factors they comprise.

Theorem 3 (Local Test Invariance). *A stringset L is Locally Testable iff there is some k such that, for all strings x and y, if $\rtimes \cdot x \cdot \ltimes$ and $\rtimes \cdot y \cdot \ltimes$ have exactly the same set of k-factors then either both x and y are members of L or neither is.*

In other words, if

$$w \equiv_k^L v \overset{\text{def}}{\iff} F_k(\rtimes w \ltimes) = F_k(\rtimes v \ltimes)$$

the LT_k stringsets cannot break the equivalence classes of Σ^* with respect to $\equiv_k^L$. (The superscript L here refers to the fact that this is an equivalence with respect to Local criteria.)

LT_k and LT as a whole are closed under Boolean operations, by definition, but are not closed under concatenation or Kleene-$*$ [5].

Since there are finitely many $\equiv_k^L$ equivalence classes, for fixed k, there are finitely many LT_k stringsets over a given alphabet. LT_k is learnable in the limit, although LT is not [21].

Example 3. Stress in the Mongolic language Buriat is governed by two explicit constraints:

- Primary stress falls on the right-most non-final heavy syllable, else on the final syllable if it is heavy, else on the initial syllable.
- Secondary stress falls on the initial syllable and on heavy syllables.

The second constraint forbids any unstressed H. One of the consequences of these constraints is that if a word ends with $\acute{H}$ then there is no non-final $\grave{H}$. This is LT_2 as witnessed by the 2-expression:

$$\neg(\acute{H} \ltimes \wedge \grave{H}\sigma)$$

It is not, on the other hand, SL since

$$\rtimes \grave{H} \overbrace{L \cdots L}^{k-1} \acute{H} L \ltimes,\ \rtimes \grave{L} \overbrace{L \cdots L}^{k-1} \acute{H} \ltimes \in L_{\text{Buriat}} \quad \text{but} \quad \rtimes \grave{H} \overbrace{L \cdots L}^{k-1} \acute{H} \ltimes \notin L_{\text{Buriat}}$$

Furthermore, it is not LT_1, either:

$$\rtimes \grave{H}\acute{H}\grave{H} \ltimes \equiv_1^L \rtimes \grave{H}\grave{H}\acute{H} \ltimes$$

Theorem 4 (LT-Hierarchy).

$$\mathrm{LT}_1 \subsetneq \mathrm{LT}_2 \subsetneq \mathrm{LT}_3 \subsetneq \cdots \subsetneq \mathrm{LT}_i \subsetneq \mathrm{LT}_{i+1} \subsetneq \cdots \subsetneq \mathrm{LT}$$

Again, the hierarchy reflects the intuition that attention to larger blocks is likely to be cognitively more difficult than attention to smaller blocks (because, for one thing, it requires more memory).

Local Test Invariance implies that, in order to distinguish strings that satisfy an LT_k pattern from those that do not, a mechanism has to be sensitive to the *set* of k-factors that occur in a string, not just whether specific k-factors occur or not.

Cognitive Interpretation of LT

- Any cognitive mechanism that can distinguish member strings from non-members of a (properly) LT_k stringset must be sensitive, at least, to the *set* of length k contiguous blocks of events that occur in the presentation of the string—both those that do occur and those that do not.
- If the strings are presented as sequences of events in time, then this corresponds to being sensitive, at each point in the string, to the set of length k blocks of events that occurred at any prior point.

Here again, this interpretation is fully independent of the way that the mechanism actually parses the strings. However it may do that, it must be able to distinguish strings purely on the basis of their sets of k-factors.

4.3 FO(+1)—Locally Threshold Testable Languages

LT constraints can require primary stress, but they cannot rule out multiple occurrences of primary stress. To see this, let $L_{\text{One}\acute{\sigma}}$ be the set of strings over $\{\sigma, \acute{\sigma}, \grave{\sigma}\}$ in which *exactly* one $\acute{\sigma}$ occurs. Suppose, for contradiction, that it is LT, hence LT_k for some k. Then

$$\rtimes \overbrace{\sigma \cdots \sigma}^{k-1} \acute{\sigma} \overbrace{\sigma \cdots \sigma}^{k-1} \ltimes \equiv_k^L \rtimes \overbrace{\sigma \cdots \sigma}^{k-1} \acute{\sigma} \overbrace{\sigma \cdots \sigma}^{k-1} \acute{\sigma} \overbrace{\sigma \cdots \sigma}^{k-1} \ltimes$$

but the former is in $L_{\text{One}\acute{\sigma}}$ while the latter is not. The problem is that LT automata cannot count. Or, more precisely, they can count only to 1.

In order to distinguish strings in which some $\acute{\sigma}$ occurs from those in which more than one occurs, we will need to be able to distinguish one instance of $\acute{\sigma}$ from another. We need to state our constraints in terms of specific positions within a string. We do this with a standard First Order language for our (+1) word models, strings with successor but not less-than.

Definition 5. FO(+1) *is the standard First Order logical system over the models* $\langle \mathcal{D}, \lhd, P_\sigma \rangle_{\sigma \in \Sigma}$ *with equality on the domain:*

$$\begin{array}{rc}
x \lhd y & w, [x \mapsto i, y \mapsto j] \models x \lhd y \overset{def}{\iff} j = i + 1 \\
x \approx y & w, [x \mapsto i, y \mapsto j] \models x \approx y \overset{def}{\iff} j = i \\
P_\sigma(x) & w, [x \mapsto i] \models P_\sigma(x) \overset{def}{\iff} i \in P_\sigma \\
\varphi \wedge \psi & \vdots \\
\neg\varphi & \vdots \\
(\exists x)[\varphi(x)] & w, s \models (\exists x)[\varphi(x)] \overset{def}{\iff} w, s[x \mapsto i] \models \varphi(x) \\
 & \qquad \textit{for some } i \in \mathcal{D}
\end{array}$$

where $w, s[x \mapsto i] \models \varphi(x)$ *says that the string* w *satisfies the formula* $\varphi(x)$*, in which the variable* x *possibly occurs, with the position* i *taken to be the value of* x*. (So* i *witnesses that there is some position which satisfies the formula.)*

We take FO(+1) to denote the class of FO(+1)-definable stringsets, as well. A stringset L *is in the class FO(+1) iff it is FO(+1)-definable:*

$$L = L(\varphi) \overset{def}{=} \{w \mid w \models \varphi\}.$$

Note that local k-expressions can be captured in FO(+1) by Boolean combinations of existential formulae with k variables. (The same k variables can be reused in each existential subformula.) Hence $\text{LT} \subsetneq \text{FO}(+1)$.

Example 4. $L_{\text{Some}\acute{\sigma}}$ is FO(+1) as witnessed by the formula $(\exists x)[\acute{\sigma}(x)]$.

$L_{\text{At-Most-One}\acute{\sigma}}$ is FO(+1) as witnessed by $(\forall x, y)[\neg(\acute{\sigma}(x) \wedge \acute{\sigma}(y) \wedge x \not\approx y)]$

Consequently, $L_{\text{One}\acute{\sigma}}$ is also FO(+1).

Thomas [22] characterizes FO(+1) in terms of *Local Threshold Testability*, equivalence in terms of the multiplicity of k-factors up to some fixed finite threshold t.

Definition 6 (Locally Threshold Testable). *A set L is* Locally Threshold Testable *(LTT) iff there is some k and t such that, for all $w, v \in \Sigma^*$:*
if for all $f \in F_k(\rtimes \cdot w \cdot \ltimes) \cup F_k(\rtimes \cdot v \cdot \ltimes)$
either $|w|_f = |v|_f$ or both $|w|_f \geq t$ and $|v|_f \geq t$,
then $w \in L \iff v \in L$.

So a stringset is $\text{LTT}_{k,t}$ iff it does not distinguish between strings that, for any k-factor w, either have the same number of occurrences of w or have at least t occurrences; a stringset is LTT iff it is $\text{LTT}_{k,t}$ for some k and t.

Theorem 5 ([22]). *A set of strings is First-order definable over $\langle \mathcal{D}, \lhd, P_\sigma \rangle_{\sigma \in \Sigma}$ iff it is* Locally Threshold Testable.

Once again, there are only finitely many stringsets over a given alphabet if k and t are fixed. So $\text{LTT}_{k,t}$ is learnable in the limit, although LTT, and FO(+1), are not.

Cognitive Interpretation of FO(+1)

- Any cognitive mechanism that can distinguish member strings from non-members of a (properly) FO(+1) stringset must be sensitive, at least, to the multiplicity of the length k blocks of events, for some fixed k, that occur in the presentation of the string, distinguishing multiplicities only up to some fixed threshold t.
- If the strings are presented as sequences of events in time, then this corresponds to being able to count up to some fixed threshold.

4.4 FO(<)—Star Free Languages

While FO(+1) formulae can distinguish strings on the multiplicity of their k-factors, they cannot distinguish the order in which those factors occur.

Example 5. A second primitive constraint on stress in Buriat is that no syllable with primary stress can properly precede a non-final heavy syllable (which, necessarily would have secondary stress). But this is not a constraint that is FO(+1) definable since

$$\rtimes \grave{L} \overbrace{L \cdots L}^{k-1} \grave{H} \overbrace{L \cdots L}^{k-1} \acute{H} \overbrace{L \cdots L}^{k-1} \ltimes$$

and

$$\rtimes \grave{L} \underbrace{L \cdots L}_{k-1} \acute{H} \underbrace{L \cdots L}_{k-1} \grave{H} \underbrace{L \cdots L}_{k-1} \ltimes$$

have the same number of each k-factor.

This is a constraint that can be enforced in terms of less-than:

$$\neg(\exists x, y)[\acute{\sigma}(x) \wedge \grave{H}(y) \wedge x < y]$$

Note that less-than is not FO definable from successor (as witnessed by the example) although successor is FO definable from less-than. Hence FO(+1) $\subsetneq$ FO(<).

The characterization of FO(<) is the primary result of McNaughton and Papert [5].

Definition 7 (Local Testability with Order). *The class of stringsets that are Locally Testable with Order (*LTO*) is the closure of* LT *under concatenation and Boolean operations.*

Note that threshold testability is not required, since it can be reduced to concatenation. Any fixed number of occurrences of a factor can be captured as the concatenation of a fixed number of single occurrences [5].

Definition 8 (Star-Free Languages). *The class of star-free stringsets is the closure of the class of finite stringsets under union, concatenation and complement with respect to* Σ^*.

This is the class of stringsets that are the denotation of regular expressions extended with a complement operator but without Kleene star.

Theorem 6 ([5]). *For any stringset L, the following are equivalent*

- *L is First-order definable over* $\langle \mathcal{D}, \lhd^+, P_\sigma \rangle_{\sigma \in \Sigma}$
- *L is* LTO
- *L is* Star-Free.

This class of languages is not learnable in the limit because it contains every finite language and at least one infinite language [15].

Cognitive Interpretation of SF **(FO(<))**

- Any cognitive mechanism that can distinguish member strings from non-members of a (properly) SF stringset must be sensitive, at least, to both the order and the multiplicity of the length k blocks of events, for some fixed k, that occur in the presentation of the string, distinguishing multiplicities only up to some fixed threshold t.
- If the strings are presented as sequences of events in time, then this corresponds to being able not only to count events up to some threshold but also to track the sequence in which those events occur.

5 Precedence—Subsequences

The Buriat constraint of Example 5 is a simple negative constraint on the order of syllables. If we specify our constraints in terms of precedence rather than adjacency this can be captured at the level corresponding to SL. We can do this simply by interpreting our atomic formulae as subsequences rather than factors. Let:

$$v \sqsubseteq w \overset{\text{def}}{\iff} v = \sigma_1 \cdots \sigma_n \text{ and } w \in \Sigma^* \cdot \sigma_1 \cdot \Sigma^* \cdots \Sigma^* \cdot \sigma_n \cdot \Sigma^*$$

So $v \sqsubseteq w$ iff v is a subsequence of w.

The logic of *Piecewise k-expressions* is identical in form and meaning to that of local k-expressions except that the atomic formulae are now strings of length less than or equal to k over Σ (with no end markers) which are interpreted as subsequences.

$$s \in \Sigma^{\leq k} \qquad w \models s \overset{\text{def}}{\iff} s \sqsubseteq w$$

To emphasize that we are working with subsequences rather than substrings, we will generally write the subsequence $\sigma_1\sigma_2$ as $\sigma_1 \,..\, \sigma_2$.

5.1 Strictly Piecewise Testable Sets

Strictly k-Piecewise Testable sets are those sets that are definable as conjunctions of negative atomic piecewise k-expressions. As with SL these are closed under intersection but not union, complement, concatenation or Kleene-$*$. And they form a proper hierarchy in k.

The class of SP constraints was characterized by Rogers, et al. [23]. Strikingly these are exactly the sets of strings that are closed under subsequence. The parameter k is the length of the longest string that is not included although all of its subsequences are. One immediate consequence of this is that no set of SP constraints will suffice to define the stress pattern of a human language since SP constraints cannot require some primary stress to occur.

Example 6. While SP constraints cannot require primary stress to occur, they can prohibit more than one primary stress, as witnessed by the piecewise 2-expression $\neg\acute{\sigma} \,..\, \acute{\sigma}$.

The Buriat constraint of Example 5 is SP_3, as witnessed by $\neg\acute{\sigma} \,..\, \grave{H} \,..\, \sigma$.

Since the sequence $\acute{\sigma}\grave{H}\sigma$ must be excluded but none of its subsequences may be (on the basis of this constraint) it is not an SP_2 definable constraint.

Note that while SP constraints can neither require strings to start or end with a particular symbol nor require them not to start or end with a particular symbol, they can forbid particular symbols from occurring anywhere except at the beginning or end of a string.

As with SL_k, SP_k is learnable in the limit if k is fixed. SP in general is not.

Cognitive Interpretation of SP

- Any cognitive mechanism that can distinguish member strings from non-members of a (properly) SP_k stringset must be sensitive, at least, to the length k (not necessarily consecutive) sequences of events that occur in the presentation of the string.
- If the strings are presented as sequences of events in time, then this corresponds to being sensitive, at each point in the string, to up to $k-1$ events distributed arbitrarily among the prior events.

5.2 Piecewise Testable Sets

Continuing to follow the pattern of the local hierarchy, the *Piecewise Testable* sets are those which are definable by arbitrary piecewise k-expressions. These are well studied, having been introduced in Simon [24]. As with the LT sets, they can be characterized as the union of finitely many equivalence classes, but with strings being equivalent if they share the same set of subsequences, rather than the same set of factors.

Theorem 7 (k-Subsequence Invariance). *A stringset L is Piecewise Testable iff there is some k such that, for all strings x and y, if x and y have exactly the same set of subsequences of length less than or equal to k then either both x and y are members of L or neither is.*

Example 7. PT constraints can require primary stress to occur on exactly one syllable: $\acute{\sigma} \wedge \neg \acute{\sigma} \mathinner{..} \acute{\sigma}$.

In general, they cannot require a syllable to occur initially or finally unless that syllable cannot occur more than once. Hence the Buriat constraint of Example 3 can be captured in PT_2 with the expression $(\acute{H} \wedge \neg \acute{H} \mathinner{..} \sigma) \rightarrow \neg \grave{H}$. This asserts that if there is some $\acute{H}$ but no non-final $\acute{H}$ (thus there is a final $\acute{H}$) then there are no $\grave{H}$.

In parallel with LT_k, PT_k for fixed k is learnable in the limit, but PT in general is not [21].

Cognitive Interpretation of PT

- Any cognitive mechanism that can distinguish member strings from non-members of a (properly) PT_k stringset must be sensitive, at least, to the set of length k subsequences of events that occur in the presentation of the string—both those that do occur and those that do not.
- If the strings are presented as sequences of events in time, then this corresponds to being sensitive, at each point in the string, to the set of all length k subsequences of the sequence of prior events.

5.3 First Order

Still following the pattern of the local hierarchy, the next step is to move to a First Order language over $(<)$ models. But successor is FO definable from less-than, so at the FO level $(<)$ models are equivalent to models with both successor and less-than. It is at this point, the Star-Free sets, that the local and piecewise hierarchies meet.

6 Further Work

From a practical point of view, one of the most important characteristics of these hierarchies is that all of the classes are closed under intersection. This means that

complicated patterns can be factored into the co-occurrence of primitive patterns of one type or the other (local or piecewise), as we have done here with Alawa and (partially) Buriat, with the overall complexity being the supremum of the complexities of the primitive constraints.

We are currently factoring all of the stress patterns in Heinz's database into primitive constraints for which we have determined the complexities with respect to the local and piecewise hierarchy. The results, though preliminary, are exciting. With the possible (controversial) exception of Cairene Arabic, all of the patterns are at worst Star-Free. Moreover, while there are patterns that are properly Star-Free from either the local or piecewise perspective (Buriat is an example), all of the patterns we have factored (nearly all of the patterns in the database) are co-occurrences of either LT and SP constraints or SL and PT constraints. This suggests that stress in every human language can be factored into co-occurrence of simple constraints, all of which are potentially learnable in the limit at least in principle if an upper bound on the length of the (sub)sequence is established.

These preliminary results are made possible by the abstract complexity measures introduced here. It is unclear how an approach based on the minimal description length of a particular model could have obtained this result. Furthermore, this complexity analysis, while introduced with examples from phonology, is much more far-reaching than that. They can be, and are being, applied to syntactic structures [25]. One reason this is possible is because these measures are sufficiently and appropriately abstract. They are agnostic about the operational details of models themselves, and therefore they provide a basis for making inferences about cognitive mechanisms that are valid, regardless of how those mechanisms are actually realized.

References

1. Folia, V., Uddén, J., de Vries, M., Forkstam, C., Petersson, K.M.: Artificial language learning in adults and children. Language Learning 60, 188–220 (2010)
2. Hauser, M.D., Chomsky, N., Fitch, W.T.: The faculty of language: What is it, who has it, and how did it evolve? Science 298(5598), 1569–1579 (2002)
3. Heinz, J.: Learning long-distance phonotactics. Linguistic Inquiry 41(4), 623–661 (2010)
4. Heinz, J., Idsardi, W.: Sentence and word complexity. Science 333(6040), 295–297 (2011)
5. McNaughton, R., Papert, S.: Counter-Free Automata. MIT Press (1971)
6. Büchi, J.R.: Weak second-order arithmetic and finite automata. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 6, 66–92 (1960)
7. Elgot, C.C.: Decision problems of finite automata and related arithmetics. Transactions of the American Mathematical Society 98, 21–51 (1961)
8. Hayes, B.: Metrical Stress Theory. Chicago University Press (1995)
9. van der Hulst, H., Goedemans, R., van Zanten, E. (eds.): A survey of word accentual patterns in the languages of the world. Mouton de Gruyter, Berlin (2010)
10. Heinz, J.: The Inductive Learning of Phonotactic Patterns. PhD thesis, University of California, Los Angeles (2007)

11. Heinz, J.: On the role of locality in learning stress patterns. Phonology 26(2), 303–351 (2009)
12. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (2001)
13. Kracht, M.: The Mathematics of Language. Mouton de Gruyter (2003)
14. Enderton, H.B.: A Mathematical Introduction to Logic. Academic Press (1972)
15. Gold, E.: Language identification in the limit. Information and Control 10, 447–474 (1967)
16. Garcia, P., Vidal, E., Oncina, J.: Learning locally testable languages in the strict sense. In: Proceedings of the Workshop on Algorithmic Learning Theory, pp. 325–338 (1990)
17. Edlefsen, M., Leeman, D., Myers, N., Smith, N., Visscher, M., Wellcome, D.: Deciding strictly local (SL) languages. In: Breitenbucher, J. (ed.) Proceedings of the Midstates Conference for Undergraduate Research in Computer Science and Mathematics, pp. 66–73 (2008)
18. Heinz, J.: UD phonology lab stress pattern database (March 2012), `http://phonology.cogsci.udel.edu/dbs/stress/`
19. Graf, T.: Comparing incomparable frameworks: A model theoretic approach to phonology. University of Pennsylvania Working Papers in Linguistics 16(2), Article 10 (2010), `http://repository.upenn.edu/pwpl/vol16/iss1/10`
20. Hyman, L.M.: How (not) to do phonological typology: the case of pitch-accent. Language Sciences 31(2-3), 213–238 (2009); Data and Theory: Papers in Phonology in Celebration of Charles W. Kisseberth
21. García, P., Ruiz, J.: Learning k-testable and k-piecewise testable languages from positive data. Grammars 7, 125–140 (2004)
22. Thomas, W.: Classifying regular events in symbolic logic. Journal of Computer and Systems Sciences 25, 360–376 (1982)
23. Rogers, J., Heinz, J., Bailey, G., Edlefsen, M., Visscher, M., Wellcome, D., Wibel, S.: On languages piecewise testable in the strict sense. In: Ebert, C., Jäger, G., Michaelis, J. (eds.) MOL 10. LNCS (LNAI), vol. 6149, pp. 255–265. Springer, Heidelberg (2010)
24. Simon, I.: Piecewise testable events. In: Brakhage, H. (ed.) GI-Fachtagung 1975. LNCS, vol. 33, pp. 214–222. Springer, Heidelberg (1975)
25. Graf, T.: Locality and the complexity of minimalist derivation tree languages. In: de Groote, P., Nederhof, M.-J. (eds.) Formal Grammar 2010/2011. LNCS, vol. 7395, pp. 208–227. Springer, Heidelberg (2012)

Is Malay Grammar Uniform? A Constraint-Based Analysis

Sharifah Raihan Syed Jaafar

National University of Malaysia
s_raihan@ukm.my

Abstract. This paper presents phonological systems of Malay grammar. Focusing on prefixation, which includes single and multiple prefixation in Malay, this study claims that the grammar of Malay is not completely uniform. The occurrence of nasal and voiceless obstruent clusters is not always resolved by nasal substitution, as claimed by previous Malay scholars regarding the clusters. Based on evidence from one million words obtained from the DBP-UKM corpus database, I further claim that Malay has co-existent grammars, one of which allows nasal and voiceless obstruent clusters, while another does not. This paper proposes that the co-existent grammars in Malay can satisfactorily be explained by adopting a constraint-based analysis named Optimality theory (Prince and Smolensky, 1993).

Keywords: Malay, prefixation, nasal substitution, Optimality theory.

1 Introduction

It has long been observed that the phonological patterns of a language are not completely uniform (e.g. Inkelas and Zoll 2007). This means that the grammar of a language can possibly have more than one phonological pattern. As stated in Inkelas and Zoll (2007), a language can vary systematically like in social register, lexical stratum (native vs. non-native), and morphological category (e.g. stem vs. affix, reduplicant vs. base).

According to what Inkelas and Zoll have claimed above, this paper intends to discuss the phonological system of Malay. By focusing on the issue of nasal and voiceless obstruent clusters in Malay prefixation, I am in agreement with the statement made by Inkelas and Zoll. Transformation and innovation happen continuously in languages. Malay is a member of the Malayic sub-group of the Malayo-Polynesian branch of the Austronesian language family. It is widely used in a number of countries including Malaysia, Indonesian, Brunei, Singapore and surrounding areas. As stated in Act 152 of the Federal Constitution of Malaysia, Malay is the national and the official language of Malaysia. As the national and official language of Malaysia, the Malay language or *Bahasa Malaysia* has undergone a long process of development towards its function as the national and official language. The language therefore has undergone much transformation and innovation

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 109–125, 2013.

which have affected the language systems. One of the language systems that was affected is the phonological system of the language. In this paper, I am going to focus on nasal and voiceless obstruent clusters in Malay prefixation.

2 Previous Studies on Malay

It has been widely claimed by previous Malay scholars (e.g. Hassan, 1974; Omar, 1986; Koh, 1981; Othman, 1983; Ahmad, 1993; Karim et al., 1989, 1994; Karim, 1995; and many others) that a nasal segment is always homorganic to the following consonant. In the case of prefixation, when nasal final prefixes are attached to roots, the nasal segments in the prefixes must be homorganic to the initial consonants of the bases. Besides the homorganic nasal, previous Malay scholars have also claimed that bases beginning with voiceless obstruents following nasal segments undergo deletion. Voiced obstruents after nasal segments however are retained – no deletion occurs. This is because the language disallows that a sequence of nasal and voiceless obstruents would emerge in the surface representation.

It has long been observed that the obstruent voiceless consonants, /p, t, k and s/, in Malay affixation are deleted when the consonants are concatenated with nasal final prefixes /pəN-/ and /məN-/. At the same time, the phonological behaviour of the nasal segments in the prefixes is always homorganic to the following consonant of the root. Let us consider some relevant examples below, as cited in Karim et al. (1994).

Nasal final prefixes in Malay (from Karim et al., 1994: 147)

a) /məŋ-pukul/	[məmukul]
ACT.PRF-scold ‘to scold’	
b) /məŋ-tari/	[mənari]
ACT.PRF-dance ‘to dance’	
c) /məŋ-karaŋ/	[məŋaraŋ]
ACT.PRF-compose ‘to compose’	
d) /məŋ-sinar/	[məŋinar]
ACT.PRF-ray ‘to ray’	

In rule-based analyses, two rules have been postulated to ensure the phonological restriction mentioned above is obeyed. The two rules are: (1) Nasal Assimilation, and (2) Voiceless Obstruent Deletion. These two rules have to be applied in order. I show below how these two rules apply:

Nasal substitution in rule-based analysis

Input	/məŋ-təmu-i/
1) Nasal assimilation	mən-təmu-wi
2) Voiceless obstruent deletion	mən-əmu-wi
Output	[mənəmuwi]

From the ordering of rules above, correct output is obtained whereby nasal and voiceless obstruent clusters do not emerge on the surface. In this study, I will argue

that the analysis proposed by scholars for this group does not work for some prefixed words. As observed in the DBP-UKM (The Institute of Language and Literature, National University of Malaysia) corpus, there are counter-examples where the clusters emerge in the surface representations, as shown below. This poses a question, as the language does not allow clusters to emerge in the surface representation, yet there are counter-examples showing the presence of clusters on the surface.

(i) /məŋ-tadbir/ [mən-tadbe]
ACT.PRF-administrative 'to administer'
(ii) /məŋ-protes/ [məm-pɣotes]
ACT.PRF-protest 'to protest'

The voiceless obstruents [t] and [p] in the above examples remain undeleted after the assimilated nasal. The rules: nasal assimilation and voiceless obstruent deletion, as postulated in a rule-based approach, fail to account for the actual process of prefixation in Malay, whereby voiceless obstruents following nasal segments in some prefixes do not undergo the deletion process. As a result, nasal and voiceless obstruent clusters emerge in the surface representation. This disobeys absolutely the grammar of the language whereby nasal and voiceless obstruent clusters are not permitted to surface.

Besides the aforementioned examples, there is another case where we can find the occurrence of nasal and voiceless obstruent clusters in the language. It occurs in multiple prefixation in Malay, i.e. when two prefixes are attached to a root. To the best of my knowledge, only two rule-based analyses concerning nasal and voiceless obstruent clusters in multiple prefixation have been performed by scholars. These are by Omar (1986) and Karim et al. (1989). I shall now demonstrate how the analyses postulated by these scholars pose a problem when accounting for nasal and voiceless obstruent clusters in multiple prefixation.

Nasal and voiceless obstruent clusters in multiple prefixation (from the DBP-UKM corpus).

i) **/pəŋ-pər**-kaja-an/ [**pə.mər**.ka.ja.an]
NOM.PRF-VERBL.PRF-rich-NOM.SUF
'enrichment'
ii) **/məŋ-pər**-luas-kan/ [**məm.pər**.lu.was.kan]
VERBL.PRF-NOM.PRF-strength-CAUS.SUF
'to cause to broaden'

When the two rules, nasal assimilation and voiceless obstruent deletion, are applied to the words, the outputs are:

(i)	Input	**/pəŋ-pər**-kaja-an/
	1) Nasal assimilation	pəm-pər-kaja-an
	2) Voiceless obstruent deletion	pəm-ər-kaja-an
	Output	[**pə.mər**.ka.ja.an]
(ii)	Input	/məŋ-pər-luwas-kan/
	1) Nasal assimilation	məm-pər-luwas-kan
	2) Voiceless obstruent deletion	məm-ər-luwas-kan
	Output	*[**mə.mər**.lu.was.kan]

As we can see in the above examples, the rule ordering, nasal assimilation and voiceless obstruent deletion, as postulated in rule-based analysis to account for nasal and voiceless obstruent clusters, only works for the data in (i). These rules, however, fail to account for the data in (ii), as *[mə.mər.lu.as.kan] is not the right output, although the cluster has been successfully eliminated. This clearly shows that the proposed solution to avoid nasal and voiceless obstruent clusters does not always work to explain the occurrence of the clusters in multiple prefixation.

Although some of the examples given fulfil the descriptive rules, they may not be able to explain the real process of prefixation in Malay, since there is evidence that some voiceless obstruent consonants are not deleted when the combining process occurs. This phenomenon of undeleted voiceless obstruents, as claimed by scholars in many cases, has been retained. Most of them resort to the same solution, which is to treat the phenomenon as somehow exceptional.

I shall discuss how the rule-based analysis poses a problem when accounting for Malay prefixation, particularly nasal final prefixes. We will then see that the problem can be accounted for by constraint-based analysis i.e. Optimality theory.

3 Data and Methodology

In order to investigate the actual process of prefixation in Malay, corpus data from the DBP-UKM corpus database were collected. As many as one million words were collected for this study. Corpus data were chosen to prove the existence of the peculiar phonological behaviour of nasal and voiceless obstruent clusters in Malay grammar, i.e. in its process of prefixation. The data are essentially needed to verify what previous studies have claimed regarding the clusters. Furthermore, corpus data were chosen because the data comprise examples of real usage of the language. As was claimed by previous Malay scholars, nasal substitution is the regular phonological process applied to break up nasal and voiceless obstruent clusters in Malay prefixation. They further claim that the occurrence of the clusters in some Malay prefixed words as listed above are exceptional cases in the language. This paper argues against this claim. By adopting a constraint-based analysis, i.e. OT, this paper claims that the cases are due to different lexical strata, i.e. native vs. non-native. The claim then reveals that Malay has co-existent grammars.

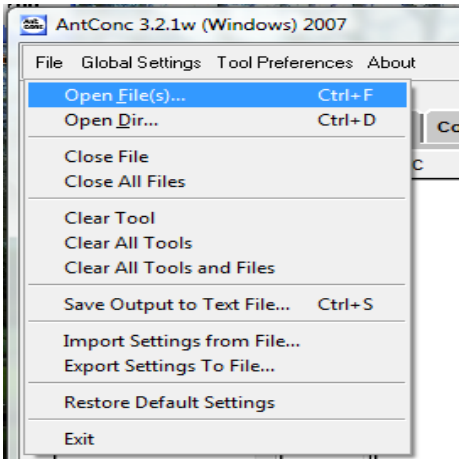
Since the data accessed from the DBP-UKM corpus are raw data, they need to be categorised according to the type of prefixes they belong to. It would be difficult to categorise one million data manually. I have therefore used corpus software named ‘AntConc’ to do the categorisation.

For single prefixation, I grouped the data according to the initial obstruent consonants of the bases: voiced and voiceless obstruents. These are two different sorts of data in which voiceless obstruents form the initial consonant of the root. There are voiceless obstruents with and without nasal substitution. The ones without nasal substitution are the type of data which violate the phonological requirements of the language since the voiceless obstruents remain undeleted. This type of data therefore violates *NC$_o$, the markedness constraint. I now explain how those groups, i.e. voiced

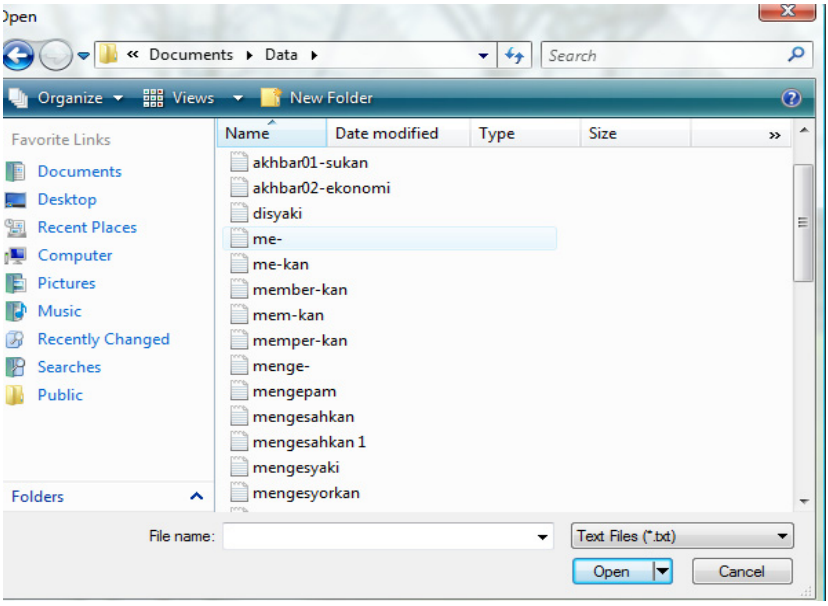
and voiceless obstruents (with or without nasal substitution), are categorised using AntConc software.

(1) Voiced Obstruents

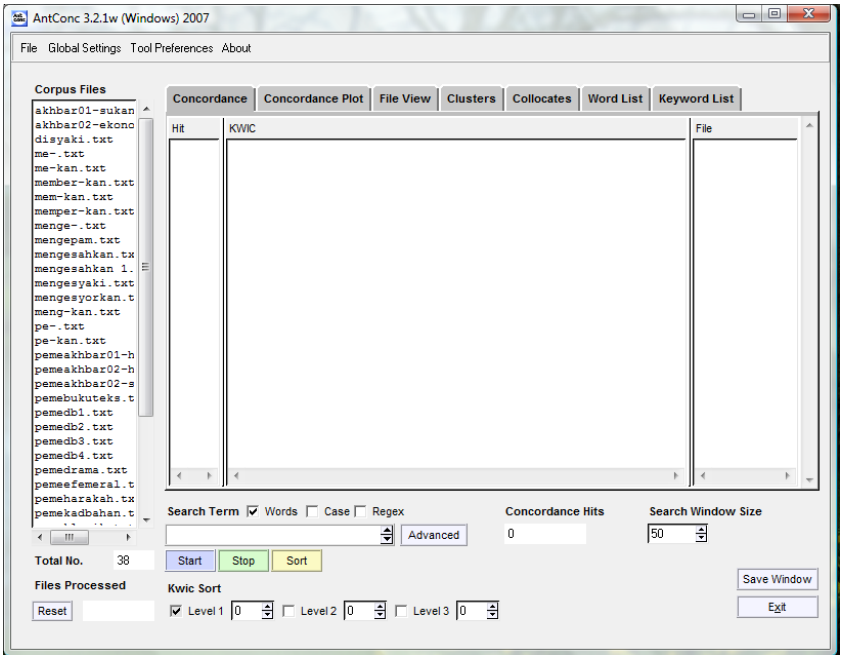
Before the relevant data for this group can be generated, we must choose a text file(s) where the data are stored by clicking on the **file** which is located at the top left of the software page and then select **open file(s)**. It looks like this:



A standard file-open will then appear. We can double click on the text folder which contains the data, select the text files by clicking on them, and then click on the **open** button on the bottom – as the following screen shows:

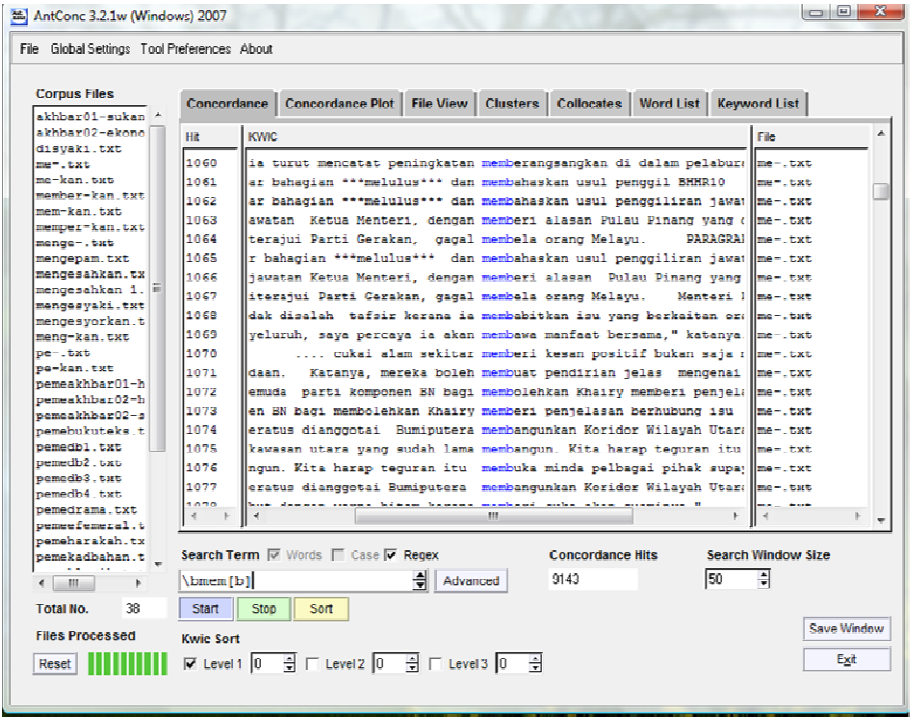


After all the text files have been selected, the data are now listed on the main page of AntConc, as shown below:



Now, we start searching for nasal and voiced obstruent clusters in the text files. To do that, we have to use some regular expressions (Regex) to search for the pattern we are looking for – nasal plus voiced obstruent. As the position of the nasal segment of the prefix is determined by the following initial consonant of the base, i.e. voiced obstruent, we cannot search for whole allomorphs of /məŋ+/ i.e. [məm], [mən] and [məŋ] at the same time. Searching for each of the allomorphs must be done one by one.

In what follows, I show how to search for the allomorph [məm]. Please bear in mind that the initial consonant following [məm] is a bilabial voiced obstruent, i.e. [b]. As just mentioned, some regular expressions must be used to search for the relevant words where the clusters are situated. Thus, a right character must be set for this. To search for [məm] followed by a voiced obstruent [b], the regular expression **\bmem[b]** is used. Observe that **\b** is added before **mem** in the regular expression **\bmem[b]** to indicate a word boundary. This regular expression, **\bmem[b]**, must be typed in the search term box on the main page of AntConc. Make sure to tick the **regex** box on the concordance screen. A concordance list will appear in the central area of the main page with all the occurrences of [mem+b]. All the steps mentioned above are shown in the screenshot below:



As we can see in this AntConc screenshot, the concordance list is for [mem+b]. To know how many examples of the word were found, just refer to the box of **Concordance Hits**. If we look at **Concordance Hits**, there are 9,143 words for [mem] + b initial base. Our search to find [mem] + b initial base examples of words in the corpus files is now done. To search for other voiced obstruents, such as [d] and [g], the steps discussed above are repeated.

(2) Voiceless Obstruents

To search for nasal plus voiceless obstruent clusters in the corpus is not as easy as searching for nasal plus voiced obstruent clusters. This is because nasal segments before voiceless obstruents undergo assimilation (for voiceless obstruents with nasal substitution). As we know, when a nasal combines with a voiceless obstruent, the nasal undergoes assimilation, while the voiceless obstruent is deleted. Therefore we get such outputs as /pəŋ-potoŋ/ → [pə-motoŋ] and /məŋ-tarik/ → [mə-nareʔ].

There are two possible ways to search for nasal and voiceless obstruent clusters in the corpus. First, if we set **\bpem** as the regular expression to search for /pəŋ+/ plus /p/ initial base, the concordance words that will appear can be: (1) the right words whereby the root actually begins with /p/ combined with a single nasal final prefix; (2) the wrong words where the root does not actually begin with /p/ combined with a single nasal final prefix but is a sonorant consonant instead, such as [pəminat] where the underlying form is /pəŋ+minat/; (3) nominal multiple prefixes /pəŋ+pər/ →

[pəmər], as in /pəŋ+pər+badan+an] → [pə.mər.ba.da.nan]. Second, if we only write **\bpe** as the regular expression, the software will generate all the words starting with [pə+]. Examples of the words that appear are as follows. For convenience, the words that start with [pə+] are underlined.

79 "Yang penting pemegang jawatan persatuan tidak bol BHES7
21 sendirian kerana pelumba di BHC34
424 Zahid yang juga pemenang pingat perak Kejohanan Lumb BHKS99
623 Daripada penelitian dan pemerhatian yang dibuat, didapati BHBC15

The examples of concordance words listed above are generated when the regular expression **\bpe** is used. None of the concordances listed above are words that we are looking for except concordance (79), which is the correct form of /pəŋ/ + /p/ initial base where the underlying form is /pəŋ+pəgaŋ/ → [pəməgaŋ]. Concordance (21) is wrong since the initial consonant of the root is not a voiceless obstruent, i.e. [lumba]. The word [perak] in concordance (424) is not a prefixed word but a root word. Since the word starts with [pe] it also appears in the concordance list. The other form we get from the search is that of multiple prefixed words, such as in concordance (623), /pəŋ+pər+hati+an/ → [pə.mər.hati.jan].

To search for nasal and voiceless obstruent clusters in the corpus, I use a second way, i.e. **\bpe,** as the regular expression to find any initial voiceless roots that combine with prefix /pəŋ+/. Since the examples of words that appear in the concordance list contain more than one phonological character, the results can be categorised into five groups: (1) /pəŋ+/ combines with a voiceless obstruent initial root (with or without nasal substitution; (2) /pəŋ+/ with a sonorant initial root; (3) nominal prefixes /pəŋ+ mər/ → [pə-mər]; (4) /pəŋ+/ with a monosyllabic root; and (5) /pəŋ+/ with a voiced obstruent initial root. Thus the categorisation has to be done manually whereby all the examples are categorised according to their phonological character. This means that we have five types of data, two of which are only useful for our analysis, i.e. (1) and (3). I briefly lay out some examples from the concordance list to represent those groups:

(1) /pəŋ+/ with voiceless obstruent initial root.

(i) With nasal substitution

921 206 mengesan penipuan apabila pemeriksaan pengesahan BHA198
941 bahawa kemunculan tanda pemesongan bearis bukanlah alasan BHA176
980 Mengenai aduan ke atas pengilang atau pengimport yang disyaki
89177 Mengenai ekonomi pula, penubuhan Zon Pemprosesan Eksport adalah

(ii) Without nasal substitution.

2446 untuk memudah dan mempercepatkan pemprosesan permohonan
3748 selain pengalaman meluas dalam pentadbiran di kementerian
9643 berisi air dan memasukkan tiub pensterilan ke dalamnya untuk
89242 terus diberikan kepada kegiatan pengkomersialan keluaran

(2) Nominal prefixes /pəŋ+mer/ → [pə-mər]

902 BSKL) membingungkan pemerhatian apabila terus mencatat BHDE81
903 Deutsche itu kerana pemerhati berpendapat ia mungkin BHDE26
937 teknologi pemerolehan minyak di tempat pengeluar BHFE61

The same situation occurs for the prefix /məŋ+/ plus initial voiceless obstruent base. All the groups mentioned above appear in the concordance list except for the third group. When **\bme** as the regular expression is entered into the **Search Term** box, we do not find any examples of words for the nominal prefixes /pəŋ+mər/ as we found before for the prefix /pəŋ+/. Verbal prefixes, i.e. /məŋ+per/ → [məmpər], are found instead. Here are examples of words for those groups:

(1) /məŋ+/ with voiceless obstruent initial root

(i) With nasal substitution

20 iaitu membuat pemecahan secara mengejut. Dia yang BHLS54
33 kerana dikatakan tidak muntuk memikul tugas sebagai BHBC16
35 itu, cukup Itali itu, pernah menewaskan pemecut handalan BHj99
94 dwitahunan 570 ini boleh memisahkan antara pemenang dan BHES60

(ii) Without nasal substitution

1409 Menjadi harapannya, lirik yang dihasilkan tidak mengkhayalkan
4038 untuk menjadikannya lebih bijak dari segi memproses dan mengawal

6061 Penduduk Palestin sebelum ini pernah memfailkan saman terhadap
6691 di luar bangunan muzium. Muzium itu turut mempamerkan

(2) Verbal prefixes /məŋ+per/ → [məm-pər]

209 mereka ke Itali untuk berla 539 paksa mempercepatkan tarikh BHC43
185 berdiri di pentas pemenang, barulah McRae memperlihatkan BHES65
462 zi yang mengenal pasti mereka yang disyaki selain memperincikan
526 Majlis Usahawan di Peringkat Daerah (MPUD) dan memperkukuhkan

4 Malay Co-existent Grammars: Constraint-Based Analysis

Observations from the DBP-UKM corpus show that the claim regarding nasal substitution postulated by previous Malay scholars on prefixation does not hold for the whole dataset. The generalization postulated by previous studies can only explain some of the output derived from the process of prefixation. This shows that the proposed rule-based analysis does not adequately explain the real process of prefixation in Malay. I am going to discuss the two patterns that occur in Malay i.e. (1) outputs with nasal and voiceless obstruent clusters, and (2) outputs without nasal and voiceless obstruent clusters. Why do these two patterns occur in the language? Supposedly, outputs with nasal and voiceless obstruent clusters should not emerge in the surface representation as the language precludes such clusters.

The occurrence of the two patterns in single prefixation is analysed in terms of different strata of Malay words, according to their etymology: native or non-native (Itô and Mester 1999). Based on the corpus data, I thus postulate the following lexical strata for Malay:

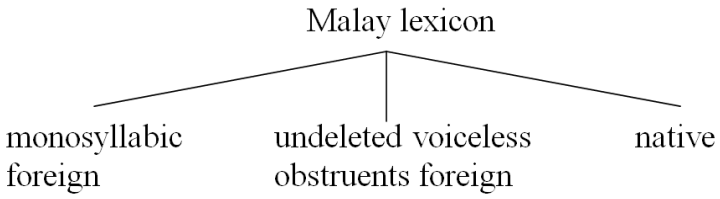


Fig. 1. The three strata of Malay lexicon (Syed Jaafar 2010)

Constraints are ranked differently in each lexical stratum according to the role played by the crucial constraints in the ranking, i.e. the markedness constraint *NC$_o$,

which bans a sequence of nasal and voiceless obstruents in the surface and the faithfulness constraint, which requires the output to be as faithful as possible to the input, i.e. UNIFORMITY.

In what follows, I am going to present how the three lexical strata proposed for the Malay lexicon are analysed. As we see below, the phenomenon of inconsistency of the occurrence of nasal and voiceless obstruent clusters is analysed by the same set of constraints but they are ranked differently. Each lexical stratum has its own constraint ranking. As was mentioned, Malay has co-existent grammars, where one does not allow nasal and voiceless obstruent clusters, while the other one does. As we shall see in the following tableau analyses, nasal substitution as the regular strategy to eliminate the clusters, as claimed by previous Malay scholars, only applies when the roots are Malay native words. This means that *NC$_{o}$, a constraint which bans a sequence of nasal and voiceless obstruents on the surface, is obeyed for Malay native words but is violated for foreign words.

> ***NC$_{o}$**
> No nasal/ voiceless obstruent sequences.

As we will see, *NC$_{o}$ is violated by foreign words as nasal substitution is not the way to resolve the clusters. Nasal and voiceless obstruent clusters in monosyllabic foreign and undeleted voiceless obstruent foreign are resolved by vowel epenthesis and nasal assimilation, respectively. The relevant constraints that play an important role to explain these phonological processes are DEP-IO and NASAL ASSIMILATION. When the clusters undergo nasal assimilation, the two segments, i.e. the nasal segment and the initial voiceless obstruent, are preserved. The preservation segments can be explained by a faithfulness constraint named UNIFORMITY. All the three constraints are defined below:

> **DEP-IO**
> Every segment in the input must have a correspondent in the output.
>
> **NAS ASS** (cf.: Jun, 1995; Padgett, 1995; Boersma, 1998; Pater, 2001)
> A nasal must share place features with a following consonant.
>
> **UNIFORMITY** ('No Coalescence') (McCarthy and Prince, 1999: 296)
> No element of S_2 has multiple correspondents in S_1.

For the monosyllabic foreign lexicon, a nasal and voiceless obstruent occurring in the input representation is resolved by vowel epenthesis. Nasal substitution, which is claimed to be the regular strategy to eliminate the clusters in the language, is not applied however. I briefly exemplify some of the relevant data from the corpus:

a) məŋ-ə-**cam**
ACT.PRF-STEMEX-recognise 'to recognise'
b) məŋ-ə-**cap**
ACT.PRF-STEMEX-stamp 'to stamp'
c) məŋ-ə-**sah**
ACT.PRF-STEMEX-validate 'to validate'
d) məŋ-ə-**kod**
ACT.PRF-STEMEX-code 'to code'

Bringing together all the constraints introduced thus far, I establish the following tableau to account for the monosyllabic foreign words. The relevant constraint ranking is: NASAL ASSIMILATION >> *NC̥ >> UNIFORMITY >> DEP-IO.

/məŋ$_1$+p$_2$am/	NAS ASS	*NC̥	UNI	DEP-IO
a. məm$_{12}$am			*!	
b. məm$_1$p$_2$am		*!		
c. məŋ$_1$p$_2$am	*!			
d.☞mə.ŋ$_1$ə.p$_2$am				*

We now see how words in the group of undeleted voiceless obstruent foreign words are analysed. Before I establish a tableau analysis for this group, let us observe first some of the relevant data below:

Nasal final prefixes (from the DBP-UKM corpus)

i) /məŋ-kritik/ [məŋ-kritik]
ACT.PRF-critic 'to criticise'

ii) /pəŋ-struktur-an/ [pən-struktu-ran]
NOM.PRF-structure-NOM.SUF
'structure'

iii) məŋ-xatan/ [məŋ-xatan]
ACT.PRF-circumcision 'to circumcise'

iv) məŋ-fasakh/ [məm-fasakh]
ACT.PRF-divorce 'to annul a marriage'

With the same set of constraints in the monosyllabic foreign lexical strata, I establish the following tableau for undeleted voiceless obstruent foreign words. Observe that the constraints are ranked differently from monosyllabic foreign words. The *NC̥ constraint which bans the clusters to emerge in the surface is ranked lower as this group allows nasal and voiceless obstruent clusters.

/məŋ$_1$+p$_2$roses/	NAS ASS	DEP-IO	UNIFORMITY	*NC̥
a. məm$_{12}$ro.ses			*!	
b. ☞məm$_1$p$_2$roses				*
c. məŋ$_1$p$_2$roses	*!			
d. məŋ$_1$əp$_2$roses		*!		
/məŋ$_1$+t$_2$auhid/				
e. mən$_{12}$auhid			*!	
f. ☞mən$_1$t$_2$auhid				*
g. məŋ$_1$t$_2$auhid	*!			
h. məŋ$_1$ət$_2$auhid		*!		

The third group is that of native words. Before I start the analysis, let us first consider some relevant examples of this group:

(i) /məŋ-potoŋ/ [mə-motoŋ]
ACT.PRF-cut 'to cut'

(ii) /məŋ-kuat-kan/ [mə-ŋuwat-kan]
ACT.PRF-strong-CAUS.SUF 'to cause to strengthen for'

(iii) /pəŋ-pindah-an/ [pə-mindah-an]
NOM.PRF-migrate-NOM.SUF 'migration'

(iv) /məŋ-kunjuŋ-i/ [mə-ŋunʤung-i]
ACT.PRF-visit-LOC.SUF 'to cause to visit'

The tableau analysis for this group is shown below:

/məŋ$_1$+p$_2$otoŋ/	NAS ASS	*NC$_o$	DEP-IO	UNIFORMITY
a. ☞ məm$_{12}$otoŋ				*
b. məm$_1$p$_2$otoŋ		*!		
c. məŋ$_1$p$_2$otoŋ	*!			
d. məŋ$_1$əp$_2$otoŋ			*!	

In multiple prefixation, the co-existent grammars occur at prefix-prefix boundaries when two prefixes end with nasal segments attached to voiceless obstruent initial roots. At this morphological boundary, the clusters emerge in the surface representation in verbal multiple prefixes /məŋ+pər/. The clusters however undergo nasal substitution in nominal multiple prefixes, /pəŋ+pər/. I exemplify some of the data taken from the corpus:

a) Verbal prefixes

i) **məm.pər**.kuwat.kan
VERBL.PRF-NOM.PRF-strength-CAUS.SUF
'to cause to strengthen for'

ii) **məm-pər**-luas-kan
VERBL.PRF-NOM.PRF-strength-CAUS.SUF
'to cause to broaden for'

iii) **mən-tər**-taʧam-kan
VERBL.PRF-VERBL.PRF-sharp- CAUS.SUF
'to cause to sharpen for'

b) Nominal prefixes

i) **pə-mər**-kaja-an
NOM.PRF-VERBL.PRF-rich-NOM.SUF
'enrichment'

ii) **pə-mər**-badan-an
NOM.PRF-VERBL.PRF-body-NOM.SUF
'organisation'
iii) **pə-məl**-bagai-an
NOM.PRF-VERBL.PRF-various-NOM.SUF
'variety'

A generalisation from the above examples can be summarised as: Nasal substitution occurs when the multiple prefixes produce a nominal prefixed word, as shown in (a). On the other hand, when the multiple prefixes form a verbal word, as in (b), nasal substitution is blocked. In this analysis, I will claim that nasal and voiceless obstruent clusters occurring in /məŋ+pər/ are due to the morphological boundary prefix-prefix where the clusters exist.

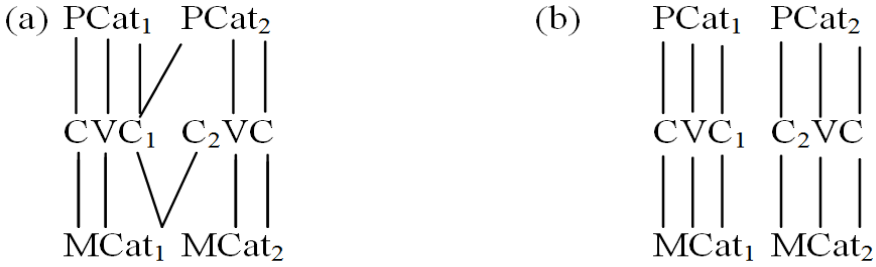
As already noted, the language does not allow nasal and voiceless obstruent clusters in the surface representation. Therefore, voiceless obstruents following nasals regularly undergo nasal substitution, as claimed by previous scholars. One question that can be asked here is: Is it obligatory for a sequence of nasal and voiceless obstruents to undergo nasal substitution? Or to put it in another way: Must nasal substitution be applied whenever there is a nasal and voiceless obstruent cluster since the phonetic requirements are already met? To answer this question in the context of multiple prefixation, I suggest that another factor, as well as the phonetic environment, i.e. the morphological environment, is worthy of consideration. Considering both factors, I claim that the process of multiple prefixation should differentiate between the verbal and the nominal prefixes. As nasal substitution is blocked in the verbal prefixes, the EDGE INTEGRITY constraint thus plays a crucial role to account for the blocking process of nasal substitution.

> **EDGE INTEGRITY** (McCarthy and Prince, 1995)
> Edge segments in the input preserve their segments at the edge of the corresponding prosodic structure.

As defined, EDGE INTEGRITY requires that the morphological unit preserves its edge segments in the input by keeping them at the edge of a corresponding prosodic structure. There is a strict faithfulness constraint on the segments at the edges so that every segment at the edge of a morphological unit is protected and is immune to phonological processes like epenthesis (Kang, 2002).

In the following diagram, we see the structure in (a) violates EDGE INTEGRITY since the final segment C_1 of $MCat_1$ is linked to $MCat_2$ and is not affiliated with $PCat_1$. Recall that the process of nasal substitution causes the two segments in the input to merge into a single segment in the output, due to the process of nasal substitution. Therefore, we see that the final segment C_1 of $MCat_1$ is also linked to the initial segment C_2 of $MCat_2$. The structure in (b) does not violate EDGE-INTEGRITY at all, since the two segments C_1 and C_2 are at the edges of their prosodic constituents.

/CVC + CVC/ ('+' stands for a morphological boundary) (from Kang, 2002).



I now establish the following constraint ranking for verbal multiple prefixes: EDGE-INTEGRITY >> NASAL ASSIMILATION >> *NC$_o$ >> DEP-IO >> UNIFORMITY.

a) Verbal prefixes

/məŋ$_1$+p$_2$ər+bəsar/	EDGE INTEG	NAS ASS	*NC$_o$	DEP-IO	UNI
a. məm$_{12}$ərbəsar	*!				*
b.☞məm$_1$p$_2$ərbəsar			*		
c. məŋ$_1$p$_2$ərbəsar		*!			

The tableau above shows that the faithfulness constraint EDGE-INTEGRITY dominates the markedness constraint *NC$_o$. Because of that, candidate (a), with nasal substitution, loses due to a fatal violation of the faithfulness constraint. In contrast, candidate (b) violates the markedness constraint *NC$_o$, as the candidate does not undergo nasal substitution. Since the markedness constraint *NC$_o$ is ranked beneath the faithfulness constraint, EDGE INTEGRITY, the least unmarked output is preferable to the unmarked ones. Therefore [məm$_1$p$_2$ərbəsar] emerges as the winner, not *[məm$_{12}$ərbəsar]. This ranking, EDGE INTEGRITY >> *NC$_o$, can thus account straightforwardly for why nasal substitution does not occur in the environment of the prefix-prefix juncture.

b) Nominal prefixes

/pəŋ$_1$+p$_2$ər+kasa/	NAS ASS	*NC$_o$	EDGE INTEG	DEP-IO	UNI
a. ☞pəm$_{12}$ərkasa			*		*
b. pəm$_1$p$_2$ərkasa		*!			
c. pəŋ$_1$p$_2$ərkasa	*!				

With a rule-based analysis, two rules, (1) nasal assimilation and (2) voiceless obstruent deletion, would be applied to account for nasal and voiceless obstruent clusters. These two rules have to be applied in order, in that the nasal assimilation rule

must precede the voiceless obstruent deletion rule. It is assumed that the same rules have also been applied to explain nasal and voiceless obstruent clusters in multiple prefixation, since there is a nasal and voiceless obstruent cluster. I illustrate how the rules apply:

	Input:	/pəŋ+pər+badan+an/
(1)	Nasal Assimilation:	pəm+pər+badan+an
(2)	Voiceless Obstruent Deletion:	pəm+ər+badan+an
	Output:	[pə.mər.bada.nan]

The above derivation shows that by applying the same rules, in order, to nominal multiple prefixes /pəŋ+pər/, the correct output is obtained. However, if this method of analysis were to be applied to another type of data, as we have in 145(b) for verbal prefixes /məŋ+pər/, we would instead get an incorrect output, as the following derivation shows:

	Input:	/məŋ+pər+kuat+kan/
(1)	Nasal Assimilation:	məm+pər+kuwat+kan
(2)	Voiceless Obstruent Deletion:	məm+ər+kuwat+kan
	Output:	*[mə.mər.kuwat.kan]

The above derivation clearly shows that the two rules, taken in order, fail to account for /məŋ+pər/. From the above derivation we derive an output with nasal substitution. This is incorrect since /məŋ+pər/ does not undergo nasal substitution.

5 Conclusions

The above discussion has presented some important points about the grammar of Malay. As we saw, nasal and voiceless obstruent clusters are not entirely prohibited in Malay. Nasal and voiceless obstruent clusters are disfavoured in the language. This can be seen in the analysis of single prefixation, where words in the native group obey *NC̥ – the constraint which bans the clusters from occurring – while in the remaining groups, monosyllabic foreign and undeleted voiceless plosive in loanwords do not.

On the other hand, nasal and voiceless obstruent clusters at the prefix-prefix juncture are not resolved by nasal substitution. The clusters at this morphological boundary are permitted to emerge in surface representation as the edges of a morphological word are preserved by the faithfulness constraint called EDGE-INTEGRITY. However, as we saw, nasal substitution applies to nominal prefixes. In the above analysis, it is clearly shown that OT offers a much better solution to handling all the problems in both single and multiple prefixes, as opposed to any other model.

Those phonological processes occurring in Malay prefixation clearly show that Malay has co-existent grammars. As a result of transformation and innovation, Malay allows nasal and voiceless obstruent clusters in the language, as occurring in foreign words, next to an absence of nasal and voiceless obstruent clusters in native words.

References

1. Ahmad, Z.: Fonologi generatif: teori dan penerapan. Institute of Language and Literature, Kuala Lumpur (1993)
2. Boersma, P.: Typology and acquisition in functional and arbitrary phonology. Ms. University of Amsterdam (1998)
3. Hassan, A.: The morphology of Malay. Institute of Language and Literature, Kuala Lumpur (1974)
4. Inkelas, S., Zoll, C.: Is grammar dependence real? A comparison between cophonological and indexed constraint approaches to morphologically conditioned phonology. Linguistics 45, 133–171 (2007)
5. Ito, J., Mester, A.: The phonological lexicon. In: Tsujimura, N. (ed.) The Handbook of Japanese Linguistics. Oxford Blackwell (1999)
6. Jun, J.: Perceptual and articulator factors in place assimilation: an Optimality theoretic approach. Doctoral dissertation. UCLA, Los Angelas (1995)
7. Kang, E.: Edge integrity and the syllable structure in Korea. In: The Proceedings of the 16th Pacific-Asia of Language. Informations and Computation. The Society of Korean Information Society, Seoul (2002)
8. Karim, N.S., Onn, F., Musa, H., Mahmood, A.H.: Tatabahasa dewan, 2nd edn. perkataan. Institute of Language and Literature, Kuala Lumpur (1989)
9. Karim, N.S., Onn, F., Musa, H., Mahmood, A.H.: Tatabahasa dewan. Institute of Language and Literature, Kuala Lumpur (1994)
10. Karim, N.S.: Malay grammar for academics and professionals. Institute of Language and Litrature, Kuala Lumpur (1995)
11. Koh, B.B.: Pengajaran bahasa Malaysia. Utusan Publications & Distributors Sdn. Bhd, Kuala Lumpur (1981)
12. McCarthy, J.J., Prince, A.S.: Faithfulness and Identity in Prosodic Morphology. In: Kager, R., van der Hulst, H., Zonneveld, W. (eds.), pp. 218–384 (1999)
13. Omar, A.: Nahu Melayu mutakhir, 2nd edn. Institute of Language and Literature, Kuala Lumpur (1986)
14. Othman, A.: Imbuhan me- dan pentingnya dalam pengajaran tatabahasa bahasa Melayu. Institute of Language and Literature (1983)
15. Padgett, J.: Partial class behaviour and nasal place assimilation. In: Proceedings of the Arizona Phonology Conference: Workshop on Features in Optimality Theory. Coyote Working Papers. University of Arizona Department of Linguistics, Tuscon (1995)
16. Pater, J.: Austronesian nasal substitution revisited. In: Lombardi, L. (ed.) Segmental Phonology in Optimality Theory: Constraints and Representations, pp. 159–182. Cambridge University Press (2001)
17. Prince, A., Smolensky, P.: Optimality theory: constraint interaction in generative grammar. Available on Rutgers Optimality Archive, ROA-537 (1993)
18. Jaafar, S., Raihan, S.: Newcastle Working Papers in Linguistics, Newcastle University, vol. 16 (2010)

Completeness of Full Lambek Calculus for Syntactic Concept Lattices

Christian Wurm

Fakultät für Linguistik und Literaturwissenschaften,
CITEC Universität Bielefeld
cwurm@uni-bielefeld.de

Abstract. Syntactic concept lattices are residuated structures which arise from the distributional analysis of a language. We show that these structures form a complete class of models with respect to the logic $\mathbf{FL}_\bot$; furthermore, its reducts are complete with respect to **FL** and $L1$.

1 Introduction

Syntactic concept lattices arise from the distributional structure of languages. Their main advantage is that they can be constructed on distributional relations which are weaker than strict equivalence. [3] has shown how these lattices can be enriched with a monoid structure to form residuated lattices. This makes it natural to ask whether they are an appropriate model for some substructural logics. Natural candidates are $L1$, the well-studied Lambek calculus (introduced in [11]) with the extension that it allows conclusions from empty premises, and its well-known conservative extensions **FL** and $\mathbf{FL}_\bot$ (see [8]).

We give a proof of completeness of $\mathbf{FL}_\bot$ for the class of residuated concept lattices for any language. Our proof will be constructed on top of well-known completeness results for the class of residuated lattices. We will show that any residuated lattice can be embedded into a syntactic concept lattice. So if an inequation fails in a residuated lattice, it also fails in the image. As corollaries, we get the completeness of syntactic concept lattice reducts for $L1$ and **FL**.

2 Residuated Syntactic Concept Lattices

2.1 Equivalences and Concepts

Syntactic concept lattices originally arose in the structuralist approach to syntax, back when syntacticians tried to capture syntactic structures purely in terms of distributions of strings[1] (see, e.g. [9]). An obvious way to do so is by partitioning strings/substrings into *equivalence classes*: we say that two strings w, v are equivalent in a language $L \subseteq \Sigma^*$, in symbols

[1] Or words, respectively, depending on whether we think of our language as a set of words or a set of strings of words.

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 126–141, 2013.

(1) $w \sim_L^0 v$ iff for all $x \in \Sigma^*$, $wx \in L \Leftrightarrow vx \in L$.

This defines the well-known Nerode-equivalence. We can use a richer equivalence relation, by considering not only left contexts, but also right contexts:

(2) $w \sim_L^1 v$ iff for all $x, y \in \Sigma^*$, $xwy \in L \Leftrightarrow xvy \in L$.

Of course, this can be arbitrarily iterated for tuples of strings. The problem with equivalence classes is that they are too restrictive for many purposes: assume we want to induce our grammar on the basis of a given dataset; then it is quite improbable that we get the equivalence classes we would usually desire as linguists. And even if we have an unlimited supply of examples, it seems unrealistic to describe our grammar on the basis of equivalence classes only: there might be constructions, collocations, idioms which ruin equivalences which we would intuitively consider to be adequate.

Syntactic concepts provide a somewhat less rigid notion of equivalence, which can be conceived of as equivalence restricted to a given set of contexts. This at least partly resolves the difficulties we have described above.

2.2 Syntactic Concepts: Definitions

For a general introduction to lattices, see [6]; for background on residuated lattices, see [8]. Syntactic concept lattices form a particular case of what is well-known as formal concept lattice (or formal concept analysis) in computer science. In linguistics, they have been introduced in [16]. They were brought back to attention and enriched with residuation in [3], [4], as they turn out to be useful representations for language learning. In this section, we follow the presentation given in [3].

Given a language $L \subseteq \Sigma^*$, we define two maps: a map $\triangleright : \wp(\Sigma^*) \to \wp(\Sigma^* \times \Sigma^*)$, and $\triangleleft : \wp(\Sigma^* \times \Sigma^*) \to \wp(\Sigma^*)$, which are defined as follows:

(3) for $M \subseteq \Sigma^*$, $M^{\triangleright} := \{(x, y) : \forall w \in M, xwy \in L\}$;

and dually

(4) for $C \subseteq \Sigma^* \times \Sigma^*$, $C^{\triangleleft} := \{x : \forall (v, w) \in C, vxw \in L\}$.

That is, a set of strings is mapped to the set of contexts, in which all of its elements can occur. The dual function maps a set of contexts to the set of strings, which can occur in all of them. Obviously, $\triangleleft$ and $\triangleright$ are only defined with respect to a given language L, otherwise they are meaningless. As long as it is clear of which language (if any concrete language) we are speaking, we will omit however any reference to it. For a set of contexts C, $C^{\triangleleft}$ can be thought of as an equivalence class with respect to the contexts in C; but not in general: there might be elements in $C^{\triangleleft}$ which can occur in a context $(v, w) \notin C$ (and conversely).

The two compositions of the maps, $\triangleleft\triangleright$ and $\triangleright\triangleleft$, form a closure operator on subsets of $\Sigma^* \times \Sigma^*$ and Σ^*, respectively, that is:

1. $M \subseteq M^{\triangleright\triangleleft}$,
2. $M^{\triangleright\triangleleft} = M^{\triangleright\triangleleft\triangleright\triangleleft}$,
3. $M \subseteq N \Rightarrow M^{\triangleright\triangleleft} \subseteq N^{\triangleright\triangleleft}$,

for $M, N \subseteq \Sigma^*$. The same holds for contexts, where we simply exchange the order of the mappings, and use subsets of $\Sigma^* \times \Sigma^*$. We say a set M is **closed** if $M^{\triangleright\triangleleft} = M$. The closure operator $\triangleright\triangleleft$ gives rise to a lattice $\mathfrak{L}_S := \langle \mathfrak{B}_S, \leq \rangle$, where the elements of $\mathfrak{B}_S$ are the closed sets, and $\leq$ is interpreted as $\subseteq$. The same can be done with the set of closed contexts. Given these two lattices, $\triangleright$ and $\triangleleft$ make up a Galois connection between the two:

1. $M \leq N \Leftrightarrow M^{\triangleleft} \geq N^{\triangleleft}$, and
2. $C \leq D \Leftrightarrow C^{\triangleright} \geq D^{\triangleright}$.

Furthermore, for $\mathfrak{L}_S$ the lattice of closed subsets of strings, $\mathfrak{L}_C$ the lattice of contexts, it is easy to show that $\mathfrak{L}_S \cong \mathfrak{L}_C^{\partial}$, where by $[-]^{\partial}$ we denote the **dual** of a lattice, that is, the same lattice with its order relation inverted; and by $\cong$ we denote that there is an isomorphism between two structures. Therefore, any statement on the one lattice is by duality a statement on the other. Consequently, we can directly conceive of the two as a single lattice, whose elements are **syntactic concepts**:

Definition 1. *A syntactic concept A is an (ordered) pair, consisting of a closed set of strings, and a closed set of contexts, written $A = \langle S, C \rangle$, such that $S^{\triangleright} = C$ and $C^{\triangleleft} = S$.*

Note also that for any set of strings S and contexts C, $S^{\triangleright} = S^{\triangleright\triangleleft\triangleright}$ and $C^{\triangleleft} = C^{\triangleleft\triangleright\triangleleft}$. Therefore, any set M of strings gives rise to a concept $\langle M^{\triangleright\triangleleft}, M^{\triangleright} \rangle$, and any set of C contexts to a concept $\langle C^{\triangleleft}, C^{\triangleleft\triangleright} \rangle$. Therefore, we denote the concept which is induced by a set M, regardless of whether it is a set of strings or contexts, by $\mathcal{C}(M)$. We speak of the *extent* of a concept A as the set of strings it contains, which we denote by S_A; the *intent* of A is the set of contexts it contains, denoted by C_A. For example, given a language L, we have $S_{\mathcal{C}((\epsilon,\epsilon))} = L$, as all and only the strings in L can occur in L in the context (ϵ, ϵ).

We define the partial order $\leq$ on concepts by

$$\langle S_1, C_1 \rangle \leq \langle S_2, C_2 \rangle \qquad \Longleftrightarrow \qquad S_1 \subseteq S_2; \tag{5}$$

which gives rise to the syntactic concept lattice $\mathfrak{L}$:

Definition 2. *The lattice of concepts of a language L, $\mathfrak{L}(L) = \langle \mathfrak{B}, \wedge, \vee \rangle$, with the partial order $\subseteq$, is called the **syntactic concept lattice**, where $\top = \mathcal{C}(\Sigma^*)$, $\bot = \mathcal{C}(\Sigma^* \times \Sigma^*)$, and for $\langle S_i, C_i \rangle, \langle S_j, C_j \rangle \in \mathfrak{B}$, $\langle S_i, C_i \rangle \wedge \langle S_j, C_j \rangle = \langle S_i \cap S_j, (C_i \cup C_j)^{\triangleleft\triangleright} \rangle$, and $\vee$ as $\langle (S_i \cup S_j)^{\triangleright\triangleleft}, C_i \cap C_j \rangle$.*

It is easy to verify that this forms a complete lattice. Note the close connection between intersection of stringsets and union of context sets, and vice versa. Obviously, $\mathfrak{L} \cong \mathfrak{L}_S$, which we defined before.

2.3 Monoid Structure and Residuation

As we have seen, the set of concepts of a language forms a lattice. In addition, we can also give it the structure of a monoid: for concepts $\langle S_1, C_1\rangle, \langle S_2, C_2\rangle$, we define:

$$\langle S_1, C_1\rangle \circ \langle S_2, C_2\rangle = \langle (S_1S_2)^{\triangleright\triangleleft}, (S_1S_2)^{\triangleright}\rangle, \tag{6}$$

where $S_1S_2 = \{xy : x \in S_1, y \in S_2\}$. Obviously, the result is a concept. $'\circ'$ is associative on concepts:

$$\text{for } X, Y, Z \in \mathfrak{B},\ X \circ (Y \circ Z) = (X \circ Y) \circ Z. \tag{7}$$

This follows from the fact that $[-]^{\triangleright\triangleleft}$ is a *nucleus*, that is, it is a closure operator and in addition it satisfies

$$S^{\triangleright\triangleleft}T^{\triangleright\triangleleft} \subseteq (ST)^{\triangleright\triangleleft}. \tag{8}$$

Using this property and the associativity of string concatenation, the result easily follows. Furthermore, it is easy to see that the neutral element of the monoid is $\mathcal{C}(\epsilon)$. This monoid structure respects the partial order of the lattice, that is:

Lemma 3. *For concepts $X, Y, Z, W \in \mathfrak{B}$, if $X \leq Y$, then $W \circ X \circ Z \leq W \circ Y \circ Z$.*

We can extend the operation $\circ$ to the contexts of concepts:

$$(x, y) \circ (w, z) = (xw, zy). \tag{9}$$

This way, we still have $f \circ (g \circ h) = (f \circ g) \circ h$ for singleton contexts f, g, h. The operation can be extended to sets in the natural way, preserving associativity. For example, $C \circ (\epsilon, S) = \{(x, ay) : (x, y) \in C, a \in S\}$. We will use this as follows:

Definition 4. *Let $X = \langle S_X, C_X\rangle, Y = \langle S_Y, C_Y\rangle$ be concepts. We define the right residual $X/Y := \mathcal{C}(C_1 \circ (\epsilon, S_2))$, and the left residual $Y\backslash X := \mathcal{C}(C_1 \circ (S_2, \epsilon))$.*

For the closed sets of strings S, T, define $S/T := \{w : \text{for all } v \in T, wv \in S\}$. We then have $S_X/S_Y = S_{X/Y}$. So residuals are unique and satisfy the following lemma:

Lemma 5. *For $X, Y, Z \in \mathfrak{B}$, we have $Y \leq X\backslash Z$ iff $X \circ Y \leq Z$ iff $X \leq Z/Y$.*

For a proof, see [3]. This shows that the syntactic concept lattice can be enriched to a residuated lattice (see the definition below, or check the references). Note that every language, whether computable or not, has a syntactic concept lattice. An important question is whether it is finite or not. This question can be answered in the following way.

Proposition 6. *The syntactic concept lattice for a language L is finite if and only if L is regular.*

This is a rather immediate consequence of the Myhill-Nerode theorem. A short note is in order about finite/infinite alphabets. Later on, we will embed residuated monoids/lattices into syntactic concept lattices. If these algebras have an infinite domain, we will need an infinite alphabet for the construction of the corresponding language; and so we will in general need languages over infinite alphabets. In case the domain of the algebra is countable, we can encode its letters with (finite) strings over a finite alphabet; if it is uncountable, however, this does not work anymore. So in the sequel, by a *language* we mean a set of finite strings over some alphabet, regardless of whether it is finite, countable or uncountable.

This is of course unsatisfactory, as for us the notion "language" implies the finiteness of the alphabet. We can yield completeness results for languages over *finite* alphabets in two ways: 1. we use the Lindenbaum-Tarski construction to construct the counter-model, which then results in a countable model, whose domain we can encode in a finite alphabet. 2. there is an even simpler solution using the finite model property of $L1$, **FL** and $\mathbf{FL}_\bot$, which says that for each underivable sequent of the logic, there is a finite model in which it does not hold. We will use this second solution.[2]

In the sequel, we will denote by *SCL* the class of all syntactic concept lattices, that is, the class of all lattices of the form $\mathfrak{L}(L)$ for some language L, without any further requirement regarding L itself except the ones stated above.

2.4 The Linguistic Order

Syntactic concepts are related to an order, which will have some importance in the sequel. Given a language $L \subseteq \Sigma^*$, we write $w \leq_L v$ iff $xvy \in L \to xwy \in L$. We call $\leq_L$ the *linguistic order*. Note that this is a pre-order, as from $w \leq_L v$ and $v \leq_L w$ follows $w \sim_L v$, where $\sim_L$ is substitutional equivalence (we denoted this above as $\sim_L^1$), but not equality. We can however think of $\leq_L$ as a partial order if we define it over $[\Sigma^*]_{\sim_L}$, that is, the set of L-equivalence classes rather than the set of strings. As is easy to see, either way $\leq_L$ respects concatenation of strings. This way, a language $L \subseteq \Sigma^*$ defines a preordered monoid $(\Sigma^*, \leq_L, \cdot, \epsilon)$.

We say a set (of strings) W is **downward closed** (with respect to $\leq_L$) if from $w \in W$ and $v \leq_L w$ it follows that $v \in W$.

Lemma 7. *Given a language $L \subseteq \Sigma^*$ and a set of strings $W \subseteq \Sigma^*$, if $W = W^{\triangleright\triangleleft}$, then W is downward closed with respect to $\leq_L$.*

Proof. Assume $W = W^{\triangleright\triangleleft}$, $v \leq_L w$, $w \in W$. We know that for all $(a, b) \in W^{\triangleright}$, $awb \in L$. By $v \leq_L w$ it follows that also $avb \in L$ if $awb \in L$. Consequently, $v \in W^{\triangleright\triangleleft}$, and so $W^{\triangleright\triangleleft}$ is downward closed. □

The converse implication does not hold, that is: not every downward closed set is closed under $[-]^{\triangleright\triangleleft}$. This is because the $[-]^{\triangleright\triangleleft}$-closure considers only the L-contexts which are common to *all* strings in W.

[2] Thanks to an anonymous reviewer for pointing this out to me.

3 Lambek Calculus and Extensions

3.1 The Logics L, $L1$, FL and $\mathbf{FL}_\bot$

The Lambek calculus L was introduced in [11]. $L1$ is a proper extension of L, and $\mathbf{FL}, \mathbf{FL}_\bot$ are each conservative extensions of $L1$ and the preceding one. Let Pr be a set, the set of **primitive types**, and C be a set of **constructors**, which is, depending on the logics we use, $C_L := \{/, \backslash, \bullet\}$, or $C_{\mathbf{FL}} := \{/, \backslash, \bullet, \vee, \wedge\}$. By $Tp_C(Pr)$ we denote the set of types over Pr, which is defined as the smallest set such that:

1. $Pr \subseteq Tp_C(Pr)$.
2. if $\alpha, \beta \in Tp_C(Pr)$, $\star \in C$, then $\alpha \star \beta \in Tp_C(Pr)$.

If there is no danger of confusion regarding the primitive types and constructors, we also simply write Tp for $Tp_C(Pr)$. We now present the inference rules corresponding to these constructors. We call an inference of the form $\Gamma \vdash \alpha$ a **sequent**, for $\Gamma \in Tp^*$, $\alpha \in Tp$, where by Tp^* we denote the set of all (possibly empty) *sequences* over Tp, which are concatenated by $'$,$'$ (keep in mind the difference between *sequents*, which have the form $\Gamma \vdash \alpha$, and *sequences* like Γ, which are in Tp^*).

With one exception, rules of inference in our logics are not given in the form of sequents $\Gamma \vdash \alpha$, but rather as rules to derive new sequents from given ones. In general, uppercase Greek letters range as variables over sequences of types. In the inference rules for L, premises of $' \vdash '$ (that is, left hand sides of sequents) must be non-empty; in $L1$ they can be empty as well; everything else is equal. In **FL** and $\mathbf{FL}_\bot$ we also allow for empty sequents. Lowercase Greek letters range over single types. Below, we present the standard rules of the Lambek calculus L / $L1$, with one axiom schema and several (meta-)rules to derive new sequents from given ones.

$$(ax) \qquad \alpha \vdash \alpha$$

$$(\mathbf{I}-/) \quad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \beta/\alpha} \qquad (\mathbf{I}-\backslash) \quad \frac{\alpha, \Gamma \vdash \beta}{\Gamma \vdash \alpha\backslash\beta}$$

$$(/-\mathbf{I}) \quad \frac{\Delta, \beta, \Theta \vdash \gamma \quad \Gamma \vdash \alpha}{\Delta, \beta/\alpha, \Gamma, \Theta \vdash \gamma} \qquad (\backslash-\mathbf{I}) \quad \frac{\Delta, \beta, \Theta \vdash \gamma \quad \Gamma \vdash \alpha}{\Delta, \Gamma, \alpha\backslash\beta, \Theta \vdash \gamma}$$

$$(\bullet-\mathbf{I}) \quad \frac{\Delta, \alpha, \beta, \Gamma \vdash \gamma}{\Delta, \alpha \bullet \beta, \Gamma \vdash \gamma} \qquad (\mathbf{I}-\bullet) \quad \frac{\Delta \vdash \alpha \quad \Gamma \vdash \beta}{\Delta, \Gamma \vdash \alpha \bullet \beta}$$

These are the standard rules of L / $L1$ (roughly as in [11]). We have rules to introduce either slash and $'\bullet'$ both on the right hand side of $\vdash$ and on the

left hand side of $\vdash$. We will now add two additional connectives, which are well-known from structural logics, namely $\vee$ and $\wedge$. These are not present in $L/L1$, have however been considered as extensions as early as in [12], and have been subsequently studied by [10].

$$(\wedge - \mathbf{I}\ 1) \quad \frac{\Gamma, \alpha, \Delta \vdash \gamma}{\Gamma, \alpha \wedge \beta, \Delta \vdash \gamma} \qquad (\wedge - \mathbf{I}\ 2) \quad \frac{\Gamma, \beta, \Delta \vdash \gamma}{\Gamma, \alpha \wedge \beta, \Delta \vdash \gamma}$$

$$(\mathbf{I} - \wedge) \quad \frac{\Gamma \vdash \alpha \quad \Gamma \vdash \beta}{\Gamma \vdash \alpha \wedge \beta}$$

$$(\vee - \mathbf{I}) \quad \frac{\Gamma, \alpha, \Delta \vdash \gamma \quad \Gamma, \beta, \Delta \vdash \gamma}{\Gamma, \alpha \vee \beta, \Delta \vdash \gamma}$$

$$(\mathbf{I} - \vee\ 1) \quad \frac{\Gamma \vdash \alpha}{\Gamma \vdash \alpha \vee \beta} \qquad (\mathbf{I} - \vee\ 2) \quad \frac{\Gamma \vdash \beta}{\Gamma \vdash \alpha \vee \beta}$$

$$(1 - \mathbf{I}) \quad \frac{\Gamma, \Delta \vdash \alpha}{\Gamma, 1, \Delta \vdash \alpha} \qquad (\mathbf{I} - 1) \quad \vdash 1$$

This gives us the logic **FL**. Note that this slightly deviates from standard terminology, because usually, **FL** has an additional constant 0. In our formulation, 0 and 1 coincide. In order to have logical counterparts of the bounded lattice elements $\top$ and $\bot$, we introduce two logical constants, which are denoted by the same symbol.[3]

$$(\bot - \mathbf{I}) \quad \Gamma, \bot\ \Delta \vdash \alpha \qquad (\mathbf{I} - \top) \quad \Gamma \vdash \top$$

This gives us the calculus $\mathbf{FL}_\bot$. From a logical point of view, all these extensions of L are quite well-behaved: they are conservative, and also allow us to preserve the important result of [11], namely admissibility of the cut-rule in L:

$$(cut) \quad \frac{\Delta, \beta, \Theta \vdash \alpha \quad \Gamma \vdash \beta}{\Delta, \Gamma, \Theta \vdash \alpha}$$

We say that a sequent $\Gamma \vdash \alpha$ is derivable in a calculus L or an extension, if it can be derived by the axiom and the rules of inference; we then write $\Vdash_L \Gamma \vdash \alpha$, $\Vdash_{L1} \Gamma \vdash \alpha$, $\Vdash_{\mathbf{FL}} \Gamma \vdash \alpha$, etc., depending on which calculus we use.

[3] Whereas L and $L1$ are equally powerful in the sense of languages which are recognizable, [10] shows that **FL** is considerably more powerful than L: whereas L only recognizes context-free languages by the classical result of [15], **FL** can recognize any finite intersection of context-free languages. We only briefly mention this, because we have no space to make precise what it means for a calculus to recognize a class of languages.

3.2 The Semantics of $L1$, FL and $\mathbf{FL}_\bot$

The standard model for L is the class of residuated semigroups. We will not consider L in the sequel, as there are some additional problems if we want to interpret L in syntactic concept lattices: there is nothing in L corresponding to the unit element, but we cannot just do away with ϵ in syntactic concept lattices without some complications. The standard model for $L1$ is the class of residuated monoids. These are structures $(M, \cdot, 1, \backslash, /, \leq)$, where $(M, \cdot, 1)$ is a monoid, $(M, \leq)$ is a partial order, and $\cdot, /, \backslash$ satisfy the law of residuation: for $m, n, o \in M$,

$$(10) \qquad m \leq o/n \Leftrightarrow m \cdot n \leq o \Leftrightarrow n \leq m\backslash o.$$

Note that this implies that $\cdot$ respects the order $\leq$. The standard model for **FL** is the class of residuated lattices, and for $\mathbf{FL}_\bot$, the class of bounded residuated lattices. A residuated lattice is an algebraic structure $\langle M, \cdot, \vee, \wedge, \backslash, /, 1\rangle$, where in addition to the previous requirements, $(M, \vee, \wedge)$ is a lattice; the lattice order $\leq$ need not be stated, as it can be induced by $\vee$ or $\wedge$: for $a, b \in M$, $a \leq b$ is a shorthand for $a \vee b = b$. A bounded residuated lattice is a structure $\langle M, \cdot, \vee, \wedge, \backslash, /, 1, \top, \bot\rangle$, where $\langle M, \cdot, \vee, \wedge, \backslash, /, 1\rangle$ is a residuated lattice, $\top$ is the maximal element of the lattice order $\leq$ and $\bot$ is its minimal element.

For a general introduction see [8]. We will give definitions only once for each operator; we can do so because each definition for a given connector is valid for all classes in which it is present.

We call the class of residuated semigroups RS, the class of residuated monoids RM, the class of residuated lattices RL; the class of bounded residuated lattices $RL_\bot$. We now give a semantics for the calculi above. We start with an interpretation σ which interprets elements in Pr, and extend σ to $\overline{\sigma}$ by defining it inductively over our type constructors, which is for now the set $C := \{/, \backslash, \bullet, \vee, \wedge\}$. Assignment goes as follows, for $\alpha, \beta \in Tp_C(Pr)$:

1. $\overline{\sigma}(\alpha) = \sigma(\alpha) \in M$, if $\alpha \in Pr$
2. $\overline{\sigma}(\top) = \top$
3. $\overline{\sigma}(\bot) = \bot$
4. $\overline{\sigma}(1) = 1$
5. $\overline{\sigma}(\alpha \bullet \beta) := \overline{\sigma}(\alpha) \cdot \overline{\sigma}(\beta)$
6. $\overline{\sigma}(\alpha/\beta) := \overline{\sigma}(\alpha)/\overline{\sigma}(\beta)$
7. $\overline{\sigma}(\alpha\backslash\beta) := \overline{\sigma}(\alpha)\backslash\overline{\sigma}(\beta)$
8. $\overline{\sigma}(\alpha \vee \beta) := \overline{\sigma}(\alpha) \vee \overline{\sigma}(\beta)$
9. $\overline{\sigma}(\alpha \wedge \beta) := \overline{\sigma}(\alpha) \wedge \overline{\sigma}(\beta)$

Note that the constructors on the left-hand side and on the right-hand side of the definition look identical (with the exception of $\bullet$ and $\cdot$), but they are not: on the left-hand side, they are type constructors, on the right hand side, they are operators of a residuated lattice. The same holds for the constants $\top, \bot, 1$.

This is how we interpret the types of our logic. What we want to interpret next are the *sequents* of the form $\Gamma \vdash \alpha$. We say that a sequent $R = \gamma_1, ..., \gamma_i \vdash \alpha$

is true in a model $\mathcal{M}$ under assignment σ, in symbols: $(\mathcal{M}, \sigma) \models \gamma_1, ..., \gamma_i \vdash \alpha$, if and only if $\overline{\sigma}(\gamma_1 \bullet ... \bullet \gamma_i) \leq \overline{\sigma}(\alpha)$ holds in $\mathcal{M}$. That is, we interpret the $'$,$'$, which denotes concatenation in sequents, as $\cdot$ in the model, and $\vdash$ as $\leq$. In the sequel, for Γ a sequence of types, we will often write $\sigma(\Gamma)$ as an abbreviation, where we leave the former translation implicit. For the case of theorems, that is, derivable sequents with no antecedent, we have the following convention: $(\mathcal{M}, \sigma) \models \vdash \alpha$ iff $1 \leq \overline{\sigma}(\alpha)$ in $\mathcal{M}$, where 1 is the unit element of $\mathcal{M}$. Note that this case does not arise in L.

More generally, for a given class of (bounded) residuated lattices (monoids, semigroups) $\mathfrak{C}$, we say that a sequent is *valid* in $\mathfrak{C}$, in symbols, $\mathfrak{C} \models \gamma_1, ..., \gamma_i \vdash \alpha$, if for all $\mathcal{M} \in \mathfrak{C}$ and all assignments σ, $(\mathcal{M}, \sigma) \models \gamma_1, ...\gamma_i \vdash \alpha$.

4 Completeness: Preliminaries

There are a number of completeness results for the logics we have considered here. We will consider the most general ones, which will be important in the sequel.

Theorem 8. *(Buszkowski) For the class of residuated semigroups RS, $RS \models \Gamma \vdash \alpha$ if and only if $\Vdash_L \Gamma \vdash \alpha$. For the class of residuated monoids RM, $RM \models \Gamma \vdash \alpha$ if and only if $\Vdash_{L1} \Gamma \vdash \alpha$.*

Theorem 9. *For the class RL of residuated lattices, $RL \models \Gamma \vdash \alpha$ if and only if $\Vdash_{\boldsymbol{FL}} \Gamma \vdash \alpha$. For the class $RL_\bot$ of bounded residuated lattices, $RL_\bot \models \Gamma \vdash \alpha$ if and only if $\Vdash_{\boldsymbol{FL}_\bot} \Gamma \vdash \alpha$.*

For reference on theorem 8, see [1], [2]. For theorem 9, see [8]. The proofs for the above completeness theorems usually proceed via the Lindenbaum-Tarski construction: we interpret primitive types as atomic terms modulo mutual derivability, and define $\sigma(\alpha) \leq \sigma(\beta)$ iff $\alpha \vdash \beta$. Then we can perform an induction over constructors to get the same for arbitrary formulas/terms. So there are quite simple completeness proofs for the general case.

What is much harder to obtain is completeness in the finite case, usually referred to as **finite model property**. A logic $\mathcal{L}$ has finite model property if from the fact that a sequent $\Gamma \vdash \alpha$ is not provable in $\mathcal{L}$, it follows that there is a *finite* model $\mathcal{M}$ and an assignment σ such that $(\mathcal{M}, \sigma) \not\models \Gamma \vdash \alpha$.

Theorem 10. *1. L1 has finite model property.*
*2. **FL** has finite model property.*
3. $\boldsymbol{FL}_\bot$ has finite model property.

For the first claim, consider [7]; the second and third has been established by [14]. We want to establish soundness and completeness of the calculi with respect to the class of syntactic concept lattices and their reducts. The latter results are crucial to show that completeness holds also if we restrict ourselves to languages over finite alphabets. First we see that our calculus is sound with respect to the model:

Theorem 11. *(Soundness) If* $\Vdash_{\boldsymbol{FL}_\bot} \Gamma \vdash \alpha$*, then for the class of syntactic concept lattices SCL, we have* $SCL \models \Gamma \vdash \alpha$*.*

This actually follows from soundness direction of theorem 9, because *SCL* is just a particular class of bounded residuated lattices. As $L, L1, \mathbf{FL}$ are fragments of $\mathbf{FL}_\bot$, we get the same result for $L, L1$ and $\mathbf{FL}$, regarding the sequents which contain only the operators which have a counterpart in the logic. What is much harder to obtain is completeness.

Theorem 12. *(Completeness) If* $SCL \models \Gamma \vdash \alpha$*, then* $\Vdash_{\boldsymbol{FL}_\bot} \Gamma \vdash \alpha$*.*

Proof idea: our proof of completeness for the class of syntactic concept lattices is based on the completeness result for the class $RL_\bot$. The idea of the proof is quite simple: starting from the completeness of $\mathbf{FL}_\bot$ for the class of bounded residuated lattices, we show that each residuated lattice can be isomorphically embedded into a syntactic concept lattice. So let $\mathbf{B}$ be a residuated lattice and $\mathfrak{L}$ a syntactic concept lattice, then from the fact that $h : B \to \mathfrak{L}$ is an isomorphic embedding, we can conclude that if $(\mathbf{B}, \sigma) \not\models \Gamma \vdash \alpha$, then $(\mathfrak{L}, h \circ \sigma) \not\models \Gamma \vdash \alpha$. This in turn means that if there is a residuated lattice, where a certain inequation does not hold, then there is a syntactic concept lattice for some language where it does not hold either. This allows us to extend the completeness result from the general class of bounded residuated lattices, which is obtained by contraposition, to the class of residuated concept lattices. The finite model property of the calculi allows us to assume that the countermodels are finite; so we can conclude that $\mathbf{FL}_\bot$ is complete with respect to the class of syntactic concept lattices of languages over finite alphabets.

The next section will be devoted to presenting the embedding and to show why it does the job as required.

5 Proof of the Main Theorem

5.1 Isomorphic Embedding in Syntactic Concept Lattices

Let $\mathbf{B} = (B, \vee, \wedge, /, \backslash, \cdot, 1, \top, \bot)$ be a bounded residuated lattice. We denote the partial order of $\mathbf{B}$ by $\leq_B$. Recall that different terms over B can denote the same element of B. To avoid confusion, we denote this by the equality $=_B$. That means, for terms s, t over B, $s =_{\mathbf{B}} t$ states that s, t denote the same element of B.

Define $\Sigma := \{b, \underline{b} : b \in B\}$. We define a language $L_B \subseteq \Sigma^*$ as the set of strings $L_B := \{b_1 b_2 ... b_n \underline{b} : b_1 \cdot b_2 \cdot ... \cdot b_n \leq_{\mathbf{B}} b\}$. For a string $w = b_1 ... b_n \in B^*$, by $w^\bullet$ we denote the term $b_1 \cdot ... \cdot b_n$.

Note that we now have an ambiguity as to whether a certain $b \in B$ is a letter of Σ or an element of the lattice. We could have generally avoided this ambiguity at the price of complicating notation; but we rather try to avoid the ambiguity in all

particular statements, by using $\leq_B$ or $\leq_L$, etc. Importantly, the non-atomic terms over the lattice $\mathbf{B}$ are *not* part of the alphabet Σ, only the elements of B which these terms denote are in Σ. So we have to take care to not read the terms as syntactic objects of L_B: whereas a term t will not occur in L_B unless it is an atomic term, the element it denotes does occur in L_B for any term t.

We define a map $\gamma : \wp(B^*) \to \wp(\Sigma^*)$ by $\gamma(X) = (X)^{\triangleright\triangleleft}$, where $[-]^{\triangleright\triangleleft}$ is the syntactic concept closure with respect to L_B. As is easy to see, γ is a closure operator, as $[-]^{\triangleright\triangleleft}$ is a closure, and we have $\gamma(b_1)\gamma(b_2) \leq \gamma(b_1 b_2)$, where by the concatenation of two sets we simply mean the concatenation of their elements, that is, $VW := \{ab : a \in V, b \in W\}$.[4] As we have said, a map which has these properties is called *nuclear*.

A nuclear map gives rise to what is called the *nuclear image* of $\wp(\mathbf{B}^*)$, the lattice $\langle \gamma[\wp(B^*)], \cap, \cup_\gamma, \circ_\gamma, /, \backslash, \gamma(\top), \gamma(\bot)\rangle$, which we will call $Q(\mathbf{B})$. From the fact that γ is nuclear, it follows that $Q(\mathbf{B})$ is a complete residuated lattice (see [8],p.174), where $X \cup_\gamma Y := \gamma(X \cup Y)$, $X \circ_\gamma Y := \gamma(X \cdot Y)$. Furthermore, $Q(\mathbf{B})$ is bounded by $\gamma(\top) = B^*$[5] and $\gamma(\bot) = \{\bot\}$, as $\bot$ is contained in any γ-closed set.

It is easy to see that $Q(\mathbf{B})$ can be isomorphically embedded into the syntactic concept lattice of L_B, which we call $\mathfrak{L}(L_B)$: in fact, $Q(\mathbf{B})$ is isomorphic to the fragment of $\mathfrak{L}(L_B)$ which consists of all concepts from strings in B^*. It is easy to see that in $\mathfrak{L}(L_B)$ these are closed under meet, join and concatenation; but note that this does not follow from general considerations, and is rather a consequence of the particular distributional structure of L_B. Consequently, we have an isomorphic embedding $\mathcal{C} : Q(\mathbf{B}) \to \mathfrak{L}(L_B)$ (as we defined it above), which simply maps closed sets of strings onto their concepts.

What we still need is an appropriate embedding from $\mathbf{B}$ into $Q(\mathbf{B})$. The next lemma will be very helpful to understand what is to follow:

Lemma 13. *Let $\leq_L$ be the linguistic order of L_B, $a, b \in B$. Then $a \leq_B b \iff a \leq_L b$.*

Proof. $\Rightarrow$ Assume $a \leq_B b$. Then if $xby \in L_B$ (we know that $y \neq \epsilon$), then it is easy to see that $xay \in L_B$, by the definition of L_B: each word of L_B corresponds to an inequation which holds in $\mathbf{B}$, and the inequation remains valid under the substitution of a for b. Therefore, $a \leq_L b$.

$\Leftarrow$ Assume $a \leq_L b$. As we have $b \leq_B b$, we have $b\underline{b} \in L_B$. By assumption, we then have $a\underline{b} \in L_B$. This can only be the case if $a \leq_B b$. □

We define a map $h : B \to Q(\mathbf{B})$ (equivalently, $h : B \to \gamma[\wp(B^*)]$), where $h(b) = \{w \in \Sigma^* : w\underline{b} \in L_B\}$. This is clearly a γ-closed set (or put differently: extent of a syntactic concept of L_B), as it equals the closed set $(\epsilon, \underline{b})^\triangleleft$.

In the sequel, we will often use h with terms instead of atoms. Of course, here the same applies as before: h is *not* defined over terms, it only maps the elements denoted by the terms. A crucial lemma is the following:

[4] The latter inequation follows from our above considerations on syntactic concepts.
[5] We assume that $\top \in B$!

Lemma 14. *For $w \in \Sigma^*, b \in B$, the following three are equivalent:*

1. $w \in h(b)$
2. $w^\bullet \leq_B b$
3. $w \leq_L b$

Proof. 2. $\Rightarrow$ 1.: if $w^\bullet \leq_B b$, then $w\underline{b} \in L_B$, and so $w \in h(b)$.

1. $\Rightarrow$ 2.: assume $w \in h(b)$. Then we have $w\underline{b} \in L_B$. But $w\underline{b} \in L_B$ only if $w^\bullet \leq_B b$, so the implication follows.

The biimplication 2. $\Leftrightarrow$ 3. is only a slight generalization of the preceding lemma. We write $w \sim_L v$ as a shorthand for $w \leq_L v$ & $v \leq_L w$ (recall that $\leq_L$ is a pre-order rather than a partial order). So assume that for $a, b, c \in B$, $a \cdot b =_B c$. From this it follows that $x \cdot (a \cdot b) \cdot y \leq_B z$ if and only if $x \cdot c \cdot y \leq_B z$. Consequently, $ab \sim_L c$. This can be extended to arbitrary terms over B and $\cdot$: for any $w \in B^*$, we have $w \sim_L w^\bullet$. Consequently, we have $w^\bullet \leq_B b$ if and only if $w^\bullet \leq_L b$ (lemma 13) if and only if $w \leq_L b$. □

We will now show that h defines a proper isomorphic embedding of $\mathbf{B}$ into the nuclear image $Q(\mathbf{B})$, which forms a fragment of the syntactic concept lattice of L_B.

Theorem 15. *For each $\star \in \{\wedge, \vee, \cdot, /, \backslash\}$, we have $h(a) \star_\gamma h(b) = h(a \star b)$, where $a \star b$ denotes the unique element of B denoted by the term, and $\star_\gamma$ denotes the interpretation of $\star$ in the γ-image of $\wp(B^*)$. This means that h is an isomorphic embedding.*

Proof. We proceed by cases:

Case 1: $\star = \wedge$.

a) $h(a) \cap h(b) \supseteq h(a \wedge b)$. Assume that $c \in h(a \wedge b)$. Then by lemma 14, $c \leq_B (a \wedge b)$. Therefore, $c \leq_B a, b$, and consequently $c \leq_L a, b$. Therefore, $c \in h(a), c \in h(b)$, and thus $c \in h(a) \cap h(b)$.

b) $h(a) \cap h(b) \subseteq h(a \wedge b)$. Assume that $c \in h(a) \cap h(b)$. Then $c \in h(a)$, $c \in h(b)$; consequently, $c \leq_B a, b$ (by lemma 14); consequently, $c \leq a \wedge b$, and therefore $c \in h(a \wedge b)$.

Case 2: $\star = \vee$

a) $h(a) \cup_\gamma h(b) \supseteq h(a \vee b)$: Assume that $c \in h(a \vee b)$. Then $c \leq_L a \vee b$. We now show that $a \vee b \in h(a) \cup_\gamma h(b)$: whenever $wav \in L_B$, $wbv \in L_B$, then $w(a \vee b)v \in L_B$. Consequently, $a \vee b \in \gamma(\{a, b\}) \subseteq h(a) \cup_\gamma h(b)$. As γ-closed sets are downward closed with respect to the linguistic order, we also have $c \in h(a) \cup_\gamma h(b)$.

b) $h(a) \cup_\gamma h(b) \subseteq h(a \vee b)$: By lemma 14, we know that for all $x \in h(a)$, $x \leq_L a$, and for all $y \in h(b)$, $y \leq_L b$. Consequently, by lemma 14 we have $z \leq_L a \vee b$ for all $z \in h(a) \cup_\gamma h(b)$, and so $h(a) \cup h(b) \subseteq h(a \vee b)$. As $h(a \vee b)$ is γ-closed, we must also have $h(a) \cup_\gamma h(b) \subseteq h(a \vee b)$ by order preservation of γ.

Case 3: $\star = \cdot$.

Recall that we use $a \sim_L b$ as a shorthand for $a \leq_L b$ & $b \leq_L a$, and that for $a, b, c \in B$, from $a \cdot b =_B c$ it follows that $ab \sim_L c$. This means that we may interchange the two arbitrarily in L_B, and likewise when we talk about the map h or any γ-closed sets.

a) $h(a) \circ_\gamma h(b) \supseteq h(a \cdot b)$. Assume that $c \in h(a \cdot b)$. Then $c \leq_L a \cdot b$, and so $c \leq_L ab$. As $h(a) \circ_\gamma h(b) = \gamma(h(a)h(b))$, we have $ab \in h(a) \circ_\gamma h(b)$, and as γ closed sets are downward closed with respect to $\leq_L$, we have $c \in h(a) \circ_\gamma h(b)$.

b) $h(a) \circ_\gamma h(b) \subseteq h(a \cdot b)$. We know that for all $x \in h(a), x \leq_B a$, and for all $y \in h(b), y \leq_B b$ (lemma 14). Consequently, as it holds in any residuated lattice that $w \leq y, x \leq z \Rightarrow wx \leq yz$, we know that for all $x \in h(a), y \in h(b), x \cdot y \leq_B a \cdot b$, and so $xy \leq_L ab \sim_L a \cdot b$. Consequently, we have $h(a)h(b) \subseteq h(a \cdot b)$. As $h(a \cdot b)$ is closed under γ and γ preserves the inclusion order of sets, it follows that $h(a) \circ_\gamma h(b) \subseteq h(a \cdot b)$.

Case 4: $\star = /$

a) $h(a)/h(b) \supseteq h(a/b)$. Assume $c \in h(a/b)$; then $c \leq_L a/b$. We show that $a/b \in h(a)/h(b)$. By definition we have that $h(a)/h(b)$ is the largest element such that $(h(a)/h(b)) \circ_\gamma h(b) \leq_{Q(\mathbf{B})} h(a)$. As for all $d \in h(b)$, $d \leq_L b$, and $a/b \cdot b \leq_L a$, $a \in h(a)$, we have for all $d \in h(b)$, $a/b \cdot d \leq_L a$, and therefore $a/b \cdot d \in h(a)$. It follows that $a/b \in h(a)/h(b)$, and consequently $c \in h(a)/h(b)$.

b) $h(a)/h(b) \subseteq h(a/b)$. We have $b \in h(b)$ as the largest element with respect to $\leq_L$, and a/b is by definition the largest element such that $a/b \cdot b \leq_B a$. Furthermore, as a is the largest element in $h(a)$ with respect to $\leq_L$, we must have for all $x \in h(a)/h(b)$, $x \leq_L a/b$; and therefore, $x \in h(a/b)$ by lemma 14.

Case 5: $\star = \backslash$ is parallel to 4. □

5.2 Back to Completeness

We now return to the proof of the main theorem. Assume that $\nvdash_{FL_\perp} \Gamma \vdash \alpha$. Then by completeness and finite model property of $\mathbf{FL}_\perp$ for bounded residuated lattices, there is a finite bounded residuated lattice $\mathbf{B}$ and assignment σ, such that $(\mathbf{B}, \sigma) \not\models \Gamma \vdash \alpha$, that is, in $\mathbf{B}$ we have $\sigma(\Gamma) \not\leq \sigma(\alpha)$.

We now take the γ-image $Q(\mathbf{B})$ and the embedding h, as we have described them in the previous section. We have to show that $(Q(\mathbf{B}), h \circ \sigma) \not\models \Gamma \vdash \alpha$, that is, we have $h \circ \sigma(\Gamma) \not\subseteq h \circ \sigma(\alpha)$, where by $h \circ \sigma$ we mean function composition, such that we have $h \circ \sigma(\gamma_1 \bullet ... \bullet \gamma_n) = h(\sigma(\gamma_1)) \cdot ... \cdot h(\sigma(\gamma_n))$.

Lemma 16. *Let $\boldsymbol{B}$ be a (bounded) residuated lattice, s, t terms over B, and $Q(\boldsymbol{B})$, h as defined above. Then $s \leq_B t$ if and only if $h(s) \subseteq h(t)$.*

Actually, for the completeness theorem we would only need the only-if direction; also, we might obtain this as a corollary to theorem 15. But as the proof is quite simple, we show both directions for "completeness".

Proof. *If*: Assume $s \leq_B t$. Then if $u \in h(s)$, we have $u \leq_B s$ (lemma 14). By transitivity, $u \leq_B t$, and therefore $u \in h(t)$. Therefore, $h(s) \subseteq h(t)$.

Only if: Assume $h(s) \subseteq h(t)$. As for all $u \in B$, $u\underline{u} \in L_B$, we have $s \in h(s)$. Therefore, $s \in h(t)$. And so, by lemma 14, $s \leq_B t$. □

To complete the proof of the main theorem, there is one step missing, which is quite straightforward: define L_B for $\mathbf{B}$ as above. Every residuated lattice $\mathbf{B}$ can be embedded into the γ-image $Q(\mathbf{B})$ by a map h, such that if $s \not\leq t$ in $\mathbf{B}$, then $h(s) \not\leq h(t)$ in $Q(\mathbf{B})$. We now have an isomorphic embedding $\mathcal{C} : Q(\mathbf{B}) \to \mathfrak{L}(L_B)$ of $Q(\mathbf{B})$ into the syntactic concept lattice of L_B, which maps the γ closed sets onto their concepts. We thus have the implication: if $s \not\leq t$ in $\mathbf{B}$, then there is a language L_B such that $\mathcal{C} \circ h(s) \not\subseteq \mathcal{C} \circ h(t)$, which is to say $\mathcal{C} \circ h(s) \not\leq_{\mathfrak{L}(L_B)} \mathcal{C} \circ h(t)$. Consequently, if $\nvdash_{FL_\bot} \Gamma \vdash \alpha$, then there is a language L_B over a finite alphabet, such that $(\mathfrak{L}(L_B), \mathcal{C} \circ h \circ \sigma) \not\models \Gamma \vdash \alpha$. This completes the proof of the main theorem.

6 Corollaries

An important feature of our proof is the following: let the logic $\mathcal{L}$ be a fragment of $\mathbf{FL}_\bot$, such that $\mathbf{FL}_\bot$ is a conservative extension of $\mathcal{L}$. We know that there exist such fragments, as $\mathbf{FL}$ is a conservative extension of $L1$, and $\mathbf{FL}_\bot$ is a conservative extension of $\mathbf{FL}$. L (and its fragments) do not satisfy this requirement, and pose some difficulties, which we hope to address in further work.

The algebraic notion corresponding to a fragment in logic is the notion of a reduct. A reduct of an algebra is the same algebra with only a proper subset of connectives; the notion extends easily to classes. So let *RED* be a certain class of reducts of $RL_\bot$, such that for $\mathcal{L}$ a fragment of $\mathbf{FL}_\bot$, $\mathcal{L}$ is complete with respect to *RED*. Then our proof of completeness regarding the class *SCL* of syntactic concept lattices can be easily adapted to give a proof of the completeness of a class of reducts of *SCL* with respect to $\mathcal{L}$, which corresponds to *RED*. The reason is that the crucial step, which is the embedding in theorem 15, is equally valid for any subset of the operators $\{\vee, \wedge, \cdot, /, \backslash\}$. So the question whether a reduct of *SCL* is complete with respect to a fragment $\mathcal{L}$ of $\mathbf{FL}_\bot$ reduces to the question whether there is a strongly complete algebraic semantics for $\mathcal{L}$, in the sense we have specified above.

Now, let SCL_{L1} be the class of SCL reducts with $\{\circ, /, \backslash\}$, which specify a unit, and SCL_{FL} be the class of SCL reducts with operators $\{\circ, /, \backslash, \vee, \wedge\}$, that is, without the constants $\top$ and $\bot$.

We get the following corollaries:

Corollary 17. *The following biimplications hold:*

1. $SCL_{L1} \models \Gamma \vdash \alpha$ *if and only if* $\Vdash_{L1} \Gamma \vdash \alpha$;
2. $SCL_{FL} \models \Gamma \vdash \alpha$ *if and only if* $\Vdash_{FL} \Gamma \vdash \alpha$.

The soundness part follows as a corollary from the soundness theorem 11; the completeness direction can be shown by a simple modification of theorem 15 in the completeness proof. By finite model property, we can again conclude that the same holds if we restrict ourselves to syntactic concept lattices of languages over finite alphabets.

7 Conclusion and Further Work

We have shown strong completeness of the class of syntactic concept lattices for languages over finite alphabets with respect to $\mathbf{FL}_\bot$ and some of its fragments. Our main conclusion is that $\mathbf{FL}_\bot$, $\mathbf{FL}$ and $L1$ are the logics of syntactic concept lattices. Apart from intrinsic mathematical interest, we think that the result presented so far is mainly of preliminary importance for formal linguistic theory. The reason is the following: the main purpose for using syntactic concept lattices in a "post-structuralist" setting is *learning*; so one major interest is to get a finite axiomatization for an infinite lattice. We hope our results will find some application in this task, which is however beyond the scope of this paper.

A further interesting open question is the following[6]: syntactic concept lattices can be extended in a very natural way (see for example [5],[13]). As extents of syntactic concepts, we take subsets of $\Sigma^* \times \Sigma^*$ (instead of subsets of Σ^*), and as intents, we take subsets of $\Sigma^* \times \Sigma^* \times \Sigma^*$ (instead of $\Sigma^* \times \Sigma^*$). Given a language $L \subseteq \Sigma^*$, and a set of pairs of strings $M \subseteq \Sigma^* \times \Sigma^*$, we put $M^\triangleright := \{(x, y, z) : \forall (a, b) \in M, xaybz \in L\}$; $[-]^\triangleleft$ is defined inversely. Obviously, this can be easily generalized to sets of arbitrary $n, n+1$ tuples. Denote the class of (generalized) syntactic concept lattices, where extents are sets of n-tuples, intents sets of $n+1$-tuples, by SCL_n. Now the question is: can our completeness result be generalized from SCL (that is, SCL_1) to SCL_n for any $n \in \mathbb{N}$?

We think the odds are good: all we need is a family of mappings $h_n : n \in \mathbb{N}$, where every $h_n : SCL_1 \to SCL_{n+1}$ is an appropriate embedding of residuated lattices. It might well be that a quite simple mapping will do the job, but we have not found a simple way of verification for the general case. Therefore, we leave this question open for further research.

Acknowledgements. I would like to thank the three anonymous reviewers for many good comments and questions, and Professor Wojciech Buszkowski, who gave me invaluable support in writing this paper.

References

1. Buszkowski, W.: Completeness results for Lambek syntactic calculus. Mathematical Logic Quarterly 32(1-5), 13–28 (1986)
2. Buszkowski, W.: Algebraic structures in categorial grammar. Theor. Comput. Sci. 1998(1-2), 5–24 (1998)
3. Clark, A.: A learnable representation for syntax using residuated lattices. In: de Groote, P., Egg, M., Kallmeyer, L. (eds.) FG 2009. LNCS (LNAI), vol. 5591, pp. 183–198. Springer, Heidelberg (2011)
4. Clark, A.: Learning context free grammars with the syntactic concept lattice. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS, vol. 6339, pp. 38–51. Springer, Heidelberg (2010)
5. Clark, A.: Logical grammars, logical theories. In: Béchet, D., Dikovsky, A. (eds.) LACL 2012. LNCS, vol. 7351, pp. 1–20. Springer, Heidelberg (2012)

[6] This question has been raised by one of the reviewers.

6. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press, Cambridge (1991)
7. Farulewski, M.: On Finite Models of the Lambek Calculus. Studia Logica 80(1), 63–74 (2005)
8. Galatos, N., Jipsen, P., Kowalski, T., Ono, H.: Residuated Lattices: An Algebraic Glimpse at Substructural Logics. Elsevier (2007)
9. Harris, Z.S.: Structural Linguistics. The University of Chicago Press (1963)
10. Kanazawa, M.: The Lambek calculus enriched with additional connectives. Journal of Logic, Language, and Information 1, 141–171 (1992)
11. Lambek, J.: The Mathematics of Sentence Structure. The American Mathematical Monthly 65, 154–169 (1958)
12. Lambek, J.: On the calculus of syntactic types. In: Jakobson, R. (ed.) Structure of Language and its Mathematical Aspects, pp. 166–178. American Mathematical Society, Providence (1961)
13. Morrill, G., Valentín, O., Fadda, M.: The displacement calculus. Journal of Logic, Language and Information 20(1), 1–48 (2011)
14. Okada, M., Terui, K.: The finite model property for various fragments of intuitionistic linear logic. J. Symb. Log. 64(2), 790–802 (1999)
15. Pentus, M.: Lambek grammars are context free. In: Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science, Los Alamitos, California, pp. 429–433. IEEE Computer Society Press (1993)
16. Sestier, A.: Contributions à une théorie ensembliste des classifications linguistiques (Contributions to a set–theoretical theory of classifications). In: Actes du Ier Congrès de l'AFCAL, Grenoble, pp. 293–305 (1960)

On the Expressivity of Optimality Theory versus Ordered Rewrite Rules

Brian Buccola

Department of Linguistics, McGill University
brian.buccola@mail.mcgill.ca

Abstract. I prove that there are phonological patterns which are expressible by ordered rewrite rules but not by any Optimality Theoretic (OT) grammar whose constraint set contains only markedness constraints and *single–state* faithfulness constraints, i.e. faithfulness constraints that assign violation marks to pairs of single input–output segments in correspondence, with no reference to other segments in the input or output. The intention is to capture formally the widespread intuition that certain *opaque* patterns, which are expressible by ordered rewrite rules, are problematic for *classic*, or traditional, OT.

1 Introduction

Generative phonology takes the phonetic form of an utterance to be the *surface* realization (or output) of an abstract, *underlying* phonological form (or input). Natural languages therefore exhibit input–output patterns, which the phonologist is tasked with describing in a formal, algorithmic way. Phonologists have largely described these patterns using two types of grammar: (i) *serial rule grammars*, e.g. as laid out in *The Sound Patterns of English* (SPE) [1], in which an input is mapped serially to intermediate outputs and finally to a terminal output via ordered, context–sensitive rewrite rules; and (ii) *parallel constraint grammars*, e.g. as in Optimality Theory (OT) [2], in which an input is mapped directly to that output which is optimal with respect to a set of ranked constraints, with no intermediate mappings (i.e. everything happens in parallel).

This paper addresses the following two questions: (i) Are there input–output patterns that can be expressed by one type of grammar but not by the other, and (ii) if so, are those patterns attested in natural language? The first is a formal question about the classes of input–output relations expressible by the two formalisms. The second is an empirical question that bears on whether phonologists should favor one type of grammar over another on the basis of empirical coverage.

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 142–158, 2013.

Under the assumption that a rule may not rewrite within that part of a string which it has already rewritten,[1] the patterns expressible by ordered rewrite rules correspond exactly to the *regular relations* [3–5].[2] Without any restrictions, the patterns expressible by OT grammars include non–regular relations, i.e. patterns that are inexpressible by ordered rewrite rules [6, 7].[3]

What is not known is whether every regular relation is expressible by some OT grammar [10]. Put differently, the question is still open whether every pattern expressible by ordered rewrite rules is expressible by some OT grammar.

Nevertheless, there is a widespread intuition that certain *opaque* phonological patterns, which are expressible by ordered rewrite rules (hence, they are regular), are problematic for *classic* [11], or traditional [10], OT grammars, consisting of just two basic types of constraints (output markedness and input–output faithfulness). However, to my knowledge, no one has yet formally proved this claim. Such a proof could reach two possible conclusions: (i) there are regular relations that are completely inexpressible by classic OT grammars, or (ii) all regular relations are expressible in principle by classic OT, albeit sometimes only with ad hoc, linguistically unmotivated, but formally sound constraints.

In this paper I define classic OT grammars as OT grammars that contain only markedness and *single–state* faithfulness constraints, i.e. faithfulness constraints that assign violation marks to pairs of single input–output segments in correspondence, with no reference to other segments in the input or output. These constraints include many, if not most, of the standard constraints proposed in the OT phonology literature, e.g. Ident, Max, and Dep. I then prove that there are regular relations that cannot be expressed by any such OT grammar.

Section 2 reviews SPE–style and OT–style grammars, as well as the notion of opacity. In section 3, I prove, using data from Canadian English, that there are input–output patterns which are expressible by ordered rewrite rules but not by any classic OT grammar, as defined here. In section 4, I discuss several other cases of opacity that are likewise provably expressible by ordered rewrite rules but not by classic OT grammars; and I discuss cases of opacity that seem, in principle, to be expressible by classic OT grammars. Section 5 concludes.

[1] Nowhere in this paper is this assumption violated. Henceforth, when I write "ordered rewrite rules," I mean rule–based grammars in which this assumption is in place.

[2] The definition of a regular relation is not important for this paper; for details, see [4].

[3] Under certain assumptions, however, the patterns expressible by OT grammars lie within the regular relations [8–10]. These assumptions are (i) that both the constraints and the function Gen mapping inputs to sets of output candidates are regular, and (ii) that there is an upper bound on the number of violation marks assigned by any constraint. Without *either* assumption, OT can describe non–regular relations. I thank an anonymous referee for helping me to clarify these points.

2 Preliminaries

In this section I lay out the architectures of SPE–style, rule–based phonologies and OT–style, constraint–based phonologies, and I explain the notion of phonological opacity.[4]

2.1 Rule–Based Phonology

In an SPE–style, rule–based grammar, inputs are mapped to outputs via ordered, context–sensitive rewrite rules of the form

$$A \rightarrow B \ / \ C \_\_ D$$

read as, "A is rewritten as B whenever A occurs immediately after C and immediately before D."[5] A is the *focus* of the rule, while $C \_\_ D$ is the *environment*. The focus and environment together are called the *input description*. Under a single such rule, an input like $/CAD/$ is mapped to the output $[CBD]$.[6] If a rule does not effect any change from input to output, then it applies *vacuously*.

Since rules simply map strings to strings, it is possible for ordered rules to *interact* in the following ways (adapted from [12]).[7]

Definition 1. *$\mathcal{R}_1$ feeds $\mathcal{R}_2$ iff $\mathcal{R}_1$ creates part of $\mathcal{R}_2$'s input description.*

Definition 2. *$\mathcal{R}_1$ bleeds $\mathcal{R}_2$ iff $\mathcal{R}_1$ removes part of $\mathcal{R}_2$'s input description.*

Consider, for example, the rule

$$\mathcal{R} : \text{aɪ} \rightarrow \text{ʌɪ} \ / \ \_\_ \ \text{t}$$

[4] Many phonologists take *features*, rather than *segments*, to be phonological primitives. That is, rules and constraints are assumed to target natural classes of segments, rather than individual segments or arbitrary sets of segments. In principle, this means that feature–based phonologies are expressively more restrictive than segment–based ones. However, in this paper the distinction is immaterial: the patterns considered here are expressible by feature–based rules (hence, also by segment–based rules), and they are not expressible by segment–based constraints (hence, neither by feature–based constraints). Accordingly, I assume that rules and constraints generally target features, but in the proof in section 3, I allow constraints to target arbitrary (sets of) segments, demonstrating that even with such added power, classic OT grammars still cannot express the relevant patterns.

[5] The more familiar notation in formal language theory is: $CAD \rightarrow CBD$.

[6] By convention, symbols between forward slashes are inputs (underlying forms), and those between square brackets are terminal outputs (surface forms). I write non–terminal, i.e. intermediate, outputs with no brackets at all. These conventions also apply to OT inputs and output candidates.

[7] Rather than creating (removing) part of $\mathcal{R}_2$'s input description, Baković [12] writes that $\mathcal{R}_1$ "creates (removes) additional (potential) inputs" to $\mathcal{R}_2$. The two formulations are equivalent. I prefer the former because it facilitates defining feeding and bleeding on environment and focus (below).

where the vowel /aɪ/ before /t/ raises to [ʌɪ], and the rule

$$\mathcal{F} : \text{t, d} \rightarrow \text{ɾ} \ / \ \text{V} \_\_ \ \text{V}$$

where /t/ or /d/ between two vowels surfaces as a flap, [ɾ]. If $\mathcal{F}$ is ordered before $\mathcal{R}$, then an input like /raɪtər/ is mapped by $\mathcal{F}$ to raɪɾər, which $\mathcal{R}$ maps vacuously to [raɪɾər]. $\mathcal{F}$ bleeds $\mathcal{R}$ because if $\mathcal{F}$ were absent, then $\mathcal{R}$ would map /raɪtər/ to [rʌɪtər]. The application of $\mathcal{F}$ before $\mathcal{R}$ removes part of $\mathcal{R}$'s input description (__ t), so that $\mathcal{R}$ no longer applies non–vacuously.

Feeding and bleeding can both be further subcategorized (adapted from [13]).

Definition 3. $\mathcal{R}_1$ feeds on $\mathcal{R}_2$'s focus *iff* $\mathcal{R}_1$ *feeds* $\mathcal{R}_2$ *by creating* $\mathcal{R}_2$*'s focus.* $\mathcal{R}_1$ feeds on $\mathcal{R}_2$'s environment *iff* $\mathcal{R}_1$ *feeds* $\mathcal{R}_2$ *by creating part of* $\mathcal{R}_2$*'s environment.*

Definition 4. $\mathcal{R}_1$ bleeds on $\mathcal{R}_2$'s focus *iff* $\mathcal{R}_1$ *bleeds* $\mathcal{R}_2$ *by removing* $\mathcal{R}_2$*'s focus.* $\mathcal{R}_1$ bleeds on $\mathcal{R}_2$'s environment *iff* $\mathcal{R}_1$ *bleeds* $\mathcal{R}_2$ *by removing part of* $\mathcal{R}_2$*'s environment.*

The example above, with $\mathcal{F}$ ordered before $\mathcal{R}$, is one of bleeding on environment: the change effected by $\mathcal{F}$, i.e. mapping the /t/ in /raɪtər/ to [ɾ], removes part of $\mathcal{R}$'s environment (__ t).

By convention, when I write that $\mathcal{R}_1$ "feeds (bleeds)" $\mathcal{R}_2$, it is implied that $\mathcal{R}_1$ is ordered *before* $\mathcal{R}_2$, even though strictly speaking, feeding (bleeding), as defined here, is independent of rule ordering. When a feeding (bleeding) rule is ordered *after* the rule it feeds (bleeds), the following terms are used.

Definition 5. $\mathcal{R}_1$ counterfeeds (counterbleeds) $\mathcal{R}_2$ *iff* $\mathcal{R}_1$ *both feeds (bleeds) and is ordered after* $\mathcal{R}_2$.

In the example above, if is ordered $\mathcal{F}$ *after* $\mathcal{R}$, then $\mathcal{F}$ counterbleeds on $\mathcal{R}$'s environment.

These notions of rule interaction have been useful in characterizing *phonological opacity*. Kiparsky [14, 15] was the first to identify the phenomenon of opacity, defining it as follows, where "process" can be construed as a rewrite rule.

Definition 6. *A process* $\mathcal{P}$ *of the form* $A \rightarrow B \ / \ C \_\_ \ D$ *is* opaque *to the extent that there are surface representations (outputs) of the form (i)* A *in the environment* $C \_\_ \ D$*, or (ii)* B *derived by* $\mathcal{P}$ *in environments other than* $C \_\_ \ D$.

The idea is that opaque phonological generalizations are (i) generalizations that appear not to hold true of a surface form, or (ii) generalizations that are true of a surface form, but the motivation for their application is obscured.

These two types of opacity are typically associated with counterfeeding and counterbleeding rule ordering, respectively [11, 12, 16]. In counterfeeding, a later rule creates part of the input description of an earlier rule, such that the earlier rule seems not to have applied to the surface form, even though it matches the rule's input description. In counterbleeding, a later rule removes part of the input description of an earlier rule, such that the earlier rule seems, on the surface, to have applied without satisfying its input description.

/raɪtər/	*aɪt	*VtV	Id(low)	Id(son)
a. raɪtər	1	1	0	0
b. rʌɪtər	0	1	1	0
c. raɪɾər	0	0	0	1
d. rʌɪɾər	0	0	1	1

Fig. 1. An example OT tableau

For example, if rule $\mathcal{F}$ from above is ordered after $\mathcal{R}$, giving rise to a counterbleeding on environment rule interaction, then the input /raɪtər/ is mapped by $\mathcal{R}$ to rʌɪtər, which $\mathcal{F}$ maps to [rʌɪɾər]. $\mathcal{R}$ is then opaque in the sense of (ii): [ʌɪ] occurs in an environment other than __ t. In such a case, I will often abstract away from the rules and say that the input–output pattern /raɪtər/ → [rʌɪɾər] is opaque.

Although opacity is defined here in terms of rules, a natural question is whether opaque input–output patterns like /raɪtər/ → [rʌɪɾər] are expressible by other formalisms, like OT grammars. In the next section, I describe the architecture of OT grammars, for which such patterns have been problematic.

2.2 Optimality Theoretic Phonology

In an OT grammar, an input like /raɪtər/ is first fed into a function Gen that generates an infinite set of *output candidates*. In practice, every possible output is a candidate. This set is then filtered down via a potentially infinite set of strictly, totally ordered constraints until a unique output candidate remains. The entire process can be visualized using an OT *tableau*.

For example, Fig. 1 presents a tableau with four output candidates for the input /raɪtər/ and four constraints, ranked left to right from highest to lowest. The process begins with the leftmost constraint, *aɪt, which is an output–markedness constraint that assigns a violation for each occurrence of the sequence [aɪt] in an output candidate. Candidate (a) thus incurs a violation of 1, while the other candidates incur 0 violations; hence, candidate (a) is eliminated. The next constraint, *VtV, which assigns a violation mark for each intervocalic [t] in an output candidate, eliminates candidate (b). Next, Ident–IO(low), an input–output faithfulness constraint which penalizes each occurrence of a low vowel like /aɪ/ mapping to a non–low vowel like [ʌɪ], eliminates candidate (d). At this point, only candidate (c) remains and is therefore the optimal output. The last constraint, Ident–IO(sonorant), which penalizes mapping a non–sonorant segment like /t/ to a sonorant one like [ɾ], effectively does no work here.

Note that in this tableau the opaque input–output pattern /raɪtər/ → [rʌɪɾər] is *not* optimal. Moreover, the violations assigned to [rʌɪɾər] are a proper superset of those assigned to [raɪɾər]. Consequently, no reranking of these particular constraints can possibly make [rʌɪɾər] more optimal than [raɪɾər]. This is the sort of reasoning that underlies the intuition that opacity is problematic for OT.

I now give a formal characterization of OT grammars, following [8, 17]. If inputs and output candidates are taken to be strings of symbols over Σ, then GEN is a relation over $\Sigma^* \times \Sigma^*$: it pairs input strings with (sets of) output candidate strings.[8] An OT grammar can then be defined formally as follows.

Definition 7. *An* OT grammar *is a tuple* $\langle \Sigma, \text{GEN}, C, >_C \rangle$*, where* $\text{GEN} \subseteq \Sigma^* \times \Sigma^*$*;* C *is a set of functions from* GEN *to* $\mathbb{N}$*; and* $>_C$ *is a strict, total order on* C*.*

A relation $>_\mathcal{G}$ over GEN defines optimality relative to two input–output pairs: $\langle i, o \rangle >_\mathcal{G} \langle i, o' \rangle$ means that $\langle i, o \rangle$ is more optimal than $\langle i, o' \rangle$ in the OT grammar $\mathcal{G}$. In such a case, I will often write that o is more optimal than o' with respect to i (in $\mathcal{G}$).

Definition 8. *Given an OT grammar* $\mathcal{G} = \langle \Sigma, \text{GEN}, C, >_C \rangle$*, with* $\langle i, o \rangle$*,* $\langle i, o' \rangle \in \text{GEN}$*,* $\langle i, o \rangle >_\mathcal{G} \langle i, o' \rangle$ *iff there is some* $c_j \in C$ *such that* $c_j(\langle i, o \rangle) < c_j(\langle i, o' \rangle)$*, and for each* c_k *such that* $c_k >_C c_j$*,* $c_k(\langle i, o \rangle) = c_k(\langle i, o' \rangle)$*.*

A single output candidate for some input is then optimal just in case it is more optimal than every other candidate with respect to that input.

Definition 9. *Given an OT grammar* $\mathcal{G} = \langle \Sigma, \text{GEN}, C, >_C \rangle$*,* $\langle i, o \rangle \in \text{GEN}$ *is* optimal *in* $\mathcal{G}$ *iff for each* $\langle i, o' \rangle \in \text{GEN}$ *(o different from o'),* $\langle i, o \rangle >_\mathcal{G} \langle i, o' \rangle$*.*

Turning now to the constraint set, an OT constraint can be represented by a weighted finite state transducer (FST) [7–9], called a *finite state OT constraint*. This FST essentially "reads" or "processes" pairs of input–output strings, one input–output symbol pair at a time, transitioning from state to state and assigning 0 or 1 violations to each pair of corresponding input–output symbols. The symbol ϵ here denotes the empty string. (It should not be confused with the IPA symbol ɛ, which denotes an open–mid front vowel.)

Definition 10. *A* finite state OT constraint *is a tuple* $\langle Q, \Sigma, \delta, q_0, F \rangle$*, where:*

1. Q *is a non–empty set of states;*
2. Σ *is a set of symbols;*
3. δ *is a transition function from* $Q \times \Sigma^\epsilon \times \Sigma^\epsilon$ *to* $\{0, 1\} \times Q$*, where* $\Sigma^\epsilon = \Sigma \cup \{\epsilon\}$*;*
4. $q_0 \in Q$ *is the unique start state;*
5. $F \subseteq Q$ *is the non–empty set of final states.*

The transition function δ takes in a triple $\langle q_j, i, o \rangle$ consisting of some initial state $q_j \in Q$, an input symbol $i \in \Sigma^\epsilon$, and an output symbol $o \in \Sigma^\epsilon$ and returns a pair $\langle n, q_k \rangle$ consisting of some number of violations $n \in \{0, 1\}$ and some terminal state $q_k \in Q$. For ease of exposition, I will write members of δ as $\langle q_j, i, o, n, q_k \rangle$ instead of $\langle \langle q_j, i, o \rangle, \langle n, q_k \rangle \rangle$ and say that δ assigns n violations to the pair $i \rightarrow o$ (on the transition from q_j to q_k).

[8] Recall that Σ^* denotes the set of all strings over the symbols in Σ, including the empty string.

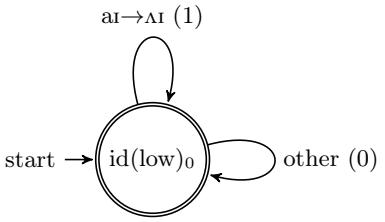


Fig. 2. A visualization of **id(low)**

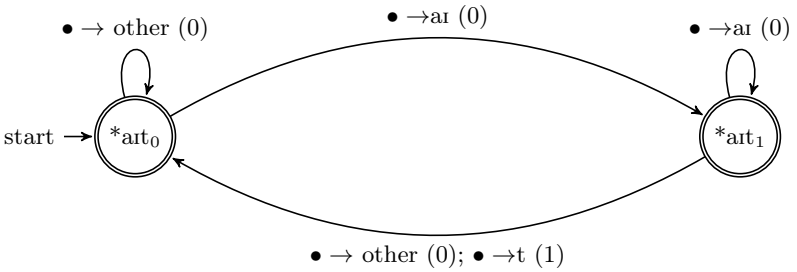


Fig. 3. A visualization of ***aɪt**

Consider, for example, the faithfulness constraint IDENT–IO(low), which, to repeat, penalizes each mapping of a low vowel like /aɪ/ to a non–low vowel like [ʌɪ]. I assume that diphthongs like [aɪ] and [ʌɪ] are single units rather than segments composed of two vowels. Formally, this means aɪ and ʌɪ are each single symbols in Σ. IDENT–IO(low) can then be represented by the single–state FST

$$\textbf{id(low)} = \langle \{\text{id(low)}_0\}, \{\text{aɪ}, \text{ʌɪ}, \dots\}, \delta, \text{id(low)}_0, \{\text{id(low)}_0\} \rangle$$

where δ assigns 1 to the pair aɪ→ʌɪ and 0 to every other pair. This FST is conventionally visualized as in Fig. 2. Thus, an input–output pair like ⟨raɪtər, raɪtər⟩ incurs 0 total violations, whereas ⟨raɪtər, rʌɪtər⟩ incurs 1 total violation.

A markedness constraint like *aɪt, which penalizes each occurrence of the sequence [aɪt] in an output candidate, can be represented by the two–state FST

$$\textbf{*aɪt} = \langle \{*\text{aɪt}_0, *\text{aɪt}_1\}, \{\text{aɪ}, \text{ʌɪ}, \text{t}, \dots\}, \delta, *\text{aɪt}_0, \{*\text{aɪt}_0, *\text{aɪt}_1\} \rangle$$

where δ assigns 1 whenever the sequence [aɪt] occurs in an output, regardless of the input, and 0 otherwise. This FST is visualized as in Fig. 3, where • stands for any symbol from the symbol set (or the empty string).

Of note is that the violations assigned by a markedness FST are independent of the input (hence the $\bullet$ on every transition). By contrast, the violations assigned by a faithfulness FST depend on the input. This can be formalized as follows.[9]

Definition 11. *Given an OT grammar* $\mathcal{G} = \langle \Sigma, \textsc{Gen}, C, >_C \rangle$*, a constraint* $c \in C$ *is a* markedness constraint *iff it can be represented by a finite state constraint* $\langle Q, \Sigma, \delta, q_0, F \rangle$ *such that, if* $\langle q_j, \alpha, \beta, 1, q_k \rangle \in \delta$*, then for any* $\gamma \in \Sigma^\epsilon$ *such that* $\langle q_j, \gamma, \beta, n, q_k \rangle \in \delta$*, it follows that* $n = 1$*. An FST that represents a markedness constraint is a* finite state markedness constraint.

Definition 12. *Given an OT grammar* $\mathcal{G} = \langle \Sigma, \textsc{Gen}, C, >_C \rangle$*, a constraint* $c \in C$ *is a* faithfulness constraint *iff it can be represented by a finite state constraint* $\langle Q, \Sigma, \delta, q_0, F \rangle$ *such that* $\langle q_j, \alpha, \beta, 1, q_k \rangle \in \delta$ *and* $\langle q_j, \gamma, \beta, 0, q_k \rangle \in \delta$*, for some* $\alpha, \beta, \gamma \in \Sigma^\epsilon$ *and for some* $q_j, q_k \in Q$*. An FST that represents a faithfulness constraint is a* finite state faithfulness constraint.

Moreover, whereas (finite state) markedness constraints may have one or more states, standard faithfulness constraints like Ident, Max, and Dep have just one state: when assigning a violation mark to an input–output symbol pair $a \rightarrow b$, these constraints do not "look ahead" or "look behind" at other symbols in the input or output. I call this sort of constraint a *single–state faithfulness constraint.*

Definition 13. *Given an OT grammar* $\mathcal{G} = \langle \Sigma, \textsc{Gen}, C, >_C \rangle$*, a constraint* $c \in C$ *is a* single–state faithfulness constraint *iff it can be represented by a finite state faithfulness constraint* $\langle Q, \Sigma, \delta, q_0, F \rangle$ *such that* $|Q| = 1$*.*

The definition of a classic OT grammar can now be stated as follows.[10,11]

Definition 14. *A* classic OT grammar *is an OT grammar* $\mathcal{G} = \langle \Sigma, \textsc{Gen}, C, >_C \rangle$ *such that (i) if* $\langle i, o \rangle \in \textsc{Gen}$*, then* $\langle i, o' \rangle \in \textsc{Gen}$*, for each* $o' \in \Sigma^*$*; and (ii) each* $c \in C$ *is either is a markedness constraint or a single–state faithfulness constraint.*

3 The Proof

In this section I prove the following claim.

[9] When there is no confusion, I will write "markedness (faithfulness) constraint" rather than "finite state markedness (faithfulness) constraint" to refer to the FST representing some OT constraint.

[10] The requirement on Gen in (i) is intended to capture the assumption, stated above, that every possible output of a given input is considered to be a candidate. See the discussion of *freedom of analysis* in [16, p. 20]. This requirement also ensures that the relevant candidates are in competition in order for the proof in section 3 to proceed, though any weaker requirement that ensures this would suffice, too.

[11] Some phonologists might object that the single–state restriction on faithfulness constraints in (ii) is *not enough*, since, for example, faithfulness constraints penalizing arbitrary input–output symbols are allowed. However, if, as I prove in section 3, even this relatively powerful version of classic OT is incapable of expressing certain regular relations, then so is any more restrictive version of it.

	Input		Output	
a.	/raɪt/	→	[rʌɪt]	"write"
b.	/raɪd/	→	[raɪd]	"ride"
c.	/raɪtər/	→	[rʌɪɾər]	"writer"
d.	/raɪdər/	→	[raɪɾər]	"rider"

Fig. 4. Canadian raising data

		$\mathcal{R}$		$\mathcal{F}$	
a.	/raɪt/	→	rʌɪt	→	[rʌɪt]
b.	/raɪd/	→	raɪd	→	[raɪd]
c.	/raɪtər/	→	rʌɪtər	→	[rʌɪɾər]
d.	/raɪdər/	→	raɪdər	→	[raɪɾər]

Fig. 5. Rule–based derivations of Canadian raising

Claim. There is a relation R such that (i) R is regular, and (ii) there is no classic OT grammar $\mathcal{G}$ such that each $\langle i, o\rangle \in R$ is optimal in $\mathcal{G}$.

The patterns I will use are the Canadian raising patterns already discussed, presented in Fig. 4. I first show that these patterns are expressible by ordered rewrite rules; hence, they are regular, proving part (i) of the claim. I then show, less trivially, that the patterns cannot be expressed by any classic OT grammar, proving part (ii) of the claim. The main result is proved as Theorem 2.

All four patterns in Fig. 4 can be captured by a rule $\mathcal{R}$ of raising /aɪ/ to [ʌɪ] before underlying /t/

$$\mathcal{R} : \text{aɪ} \rightarrow \text{ʌɪ} \;/\; \underline{\quad}\text{t}$$

ordered before a rule $\mathcal{F}$ of flapping (changing intervocalic /t/ or /d/ to [ɾ])

$$\mathcal{F} : \text{t, d} \rightarrow \text{ɾ} \;/\; \text{V}\underline{\quad}\text{V}$$

where V stands for any vowel, e.g. /aɪ/, /ʌɪ/, and /ə/. Fig. 5 shows how the four patterns are expressed by the interaction of these two rules.

Recall that the pattern /raɪtər/ → [rʌɪɾər] is the crucial opaque pattern: /aɪ/ raises to [ʌɪ] because of the underlying /t/ ($\mathcal{R}$'s environment), but /t/ is changed to [ɾ] by $\mathcal{F}$. Thus, $\mathcal{F}$ removes the environment that motivates the application of $\mathcal{R}$, obscuring, on the surface, the reason for raising.

These patterns are intuitively problematic for classic OT for the following reason: given the evidence from /raɪdər/ that [raɪɾər] is an unmarked output, it is always more optimal to map /raɪtər/ to [raɪɾər], incurring just one faithfulness violation (t→ɾ), than to map /raɪtər/ to [rʌɪɾər], incurring two faithfulness violations (t→ɾ, aɪ→ʌɪ). (This explains why in Fig. 1 of the previous section, [raɪɾər] is a more optimal output of /raɪtər/ than [rʌɪɾər].)

Informally, the patterns /raɪtər/ → [rʌɪɾər] and /raɪdər/ → [raɪɾər] are conflicting. More formally, we have the following theorem.

Theorem 1. *Let* $\mathcal{G} = \langle \Sigma, \textsc{Gen}, C, >_C \rangle$ *be a classic OT grammar such that* ⟨raɪtər, rʌɪfər⟩ *and* ⟨raɪdər, raɪfər⟩ *are members of* Gen. *Then it is not the case both that* ⟨raɪtər, rʌɪfər⟩ *is optimal in* $\mathcal{G}$ *and that* ⟨raɪdər, raɪfər⟩ *is optimal in* $\mathcal{G}$.

Proof. The proof is by contradiction. Assume both that ⟨raɪtər, rʌɪfər⟩ is optimal in $\mathcal{G}$ and that ⟨raɪdər, raɪfər⟩ is optimal in $\mathcal{G}$. Then for each ⟨raɪtər, o⟩ ∈ Gen (with o different from [rʌɪfər]), ⟨raɪtər, rʌɪfər⟩ $>_{\mathcal{G}}$ ⟨raɪtər, o⟩; and for each ⟨raɪdər, o⟩ ∈ Gen (with o different from [raɪfər]), ⟨raɪdər, raɪfər⟩ $>_{\mathcal{G}}$ ⟨raɪdər, o⟩.

Since ⟨raɪtər, rʌɪfər⟩ and ⟨raɪdər, raɪfər⟩ are members of Gen, then so are ⟨raɪtər, raɪfər⟩ and ⟨raɪdər, rʌɪfər⟩. It follows both that ⟨raɪtər, rʌɪfər⟩ $>_{\mathcal{G}}$ ⟨raɪtər, raɪfər⟩ and that ⟨raɪdər, raɪfər⟩ $>_{\mathcal{G}}$ ⟨raɪdər, rʌɪfər⟩. By the definition of optimality, the following statements are derived.

$$\text{There is a } c_j \in C \text{ such that } c_j(\langle \text{raɪtər}, \text{rʌɪfər}\rangle) < c_j(\langle \text{raɪtər}, \text{raɪfər}\rangle). \quad (1)$$

$$\text{For all } c' \in C \text{ such that } c' >_C c_j, c'(\langle \text{raɪtər}, \text{rʌɪfər}\rangle) = c'(\langle \text{raɪtər}, \text{raɪfər}\rangle). \quad (2)$$

$$\text{There is a } c_k \in C \text{ such that } c_k(\langle \text{raɪdər}, \text{raɪfər}\rangle) < c_k(\langle \text{raɪdər}, \text{rʌɪfər}\rangle). \quad (3)$$

$$\text{For all } c' \in C \text{ such that } c' >_C c_k, c'(\langle \text{raɪdər}, \text{raɪfər}\rangle) = c'(\langle \text{raɪdər}, \text{rʌɪfər}\rangle). \quad (4)$$

Recall that, as defined here, classic OT grammars contain only markedness and single–state faithfulness constraints. Thus, to prove the theorem, it must be shown that at least one of c_j, c_k is neither of these types. There are four cases to consider.

Case 1. Suppose that both c_j and c_k are markedness constraints. Then c_j assigns m violation marks to [rʌɪfər] and n violations to [raɪfər], regardless of input; hence, from (1) it follows that $m < n$. Similarly, c_k assigns p violation marks to [rʌɪfər] and q violations to [raɪfər], regardless of input; hence, from (3) it follows that $q < p$.

It cannot be the case that c_j and c_k are the same constraint, for then $m = p$ and $n = q$, deriving a contradiction. Thus, c_j and c_k are different constraints, with one ranked above the other. Suppose that $c_j >_C c_k$. Then it follows from (4) that

$$c_j(\langle \text{raɪdər}, \text{raɪfər}\rangle) = c_j(\langle \text{raɪdər}, \text{rʌɪfər}\rangle)$$

and hence $n = m$. But it was already established that $m < n$, so it cannot be that $c_j >_C c_k$.

Now suppose that $c_k >_C c_j$. Then it follows from (2) that

$$c_k(\langle \text{raɪtər}, \text{rʌɪfər}\rangle) = c_k(\langle \text{raɪtər}, \text{raɪfər}\rangle)$$

and hence $p = q$. But it was already established that $q < p$, so it cannot be that $c_k >_C c_j$. We have reached a contradiction, so c_j and c_k cannot both be markedness constraints. At least one of them is a single–state faithfulness constraint.

Case 2. Suppose that c_j is a single–state faithfulness constraint and that c_k is a markedness constraint. From (1) it follows that the FST representing c_j assigns 1 violation to the pair aɪ→ʌɪ and 0 violations to the pair aɪ→ʌɪ, because the two inputs are the same and the two output candidates differ only in having [aɪ] or [ʌɪ] as the second segment. Thus, it follows that

$$c_j(\langle \text{raɪdər}, \text{rʌɪfər}\rangle) < c_j(\langle \text{raɪdər}, \text{raɪfər}\rangle)$$

since, again, the two inputs are the same and the two output candidates differ only in the second segment.

Since c_j and c_k are different constraints, one must outrank the other. It cannot be that $c_j >_C c_k$, for it was just established that

$$c_j(\langle \text{raɪdər}, \text{rʌɪfər}\rangle) \neq c_j(\langle \text{raɪdər}, \text{raɪfər}\rangle)$$

which contradicts (4). Suppose, then, that $c_k >_C c_j$. Since by hypothesis c_k is a markedness constraint, then from (3) it follows that c_k assigns fewer violations to [raɪfər] than to [rʌɪfər], regardless of input, so that

$$c_k(\langle \text{raɪtər}, \text{raɪfər}\rangle) \neq c_k(\langle \text{raɪtər}, \text{rʌɪfər}\rangle)$$

But this contradicts (2). We have reached a contradiction: if c_j is a single–state faithfulness constraint, then c_k cannot be a markedness constraint.

Case 3. Suppose that c_j and c_k are both single–state faithfulness constraints. From (3) it follows that the FST representing c_k assigns 1 violation to the pair aɪ→ʌɪ and 0 violations to the pair aɪ→aɪ. Thus, it follows that

$$c_k(\langle \text{raɪtər}, \text{raɪfər}\rangle) < c_k(\langle \text{raɪtər}, \text{rʌɪfər}\rangle)$$

Since this statement is different from (1), c_j and c_k must be different constraints and hence ranked one over the other. It cannot be that $c_k >_C c_j$, for the above statement contradicts (2). However, as was established in case 2, if c_j is a single–state faithfulness constraint, then it cannot be that $c_j >_C c_k$. We have reached a contradiction: if c_k is a single–state faithfulness constraint, then c_j must be a markedness constraint.

Case 4. Suppose that c_j is a markedness constraint and that c_k is a single–state faithfulness constraint. From (1) it follows that c_j assigns fewer violations to [rʌɪfər] than to [raɪfər], regardless of input; hence, from the statement derived in case 3 concerning c_k as a single–state faithfulness constraint, it follows that c_j and c_k are different constraints, with one ranked above the other.

It cannot be that $c_j >_C c_k$, for otherwise

$$c_j(\langle \text{raɪdər}, \text{rʌɪfər}\rangle) < c_j(\langle \text{raɪdər}, \text{raɪfər}\rangle)$$

which contradicts (4). Thus, $c_k >_C c_j$. However, as already established in case 3, if c_k is a single–state faithfulness constraint, then it cannot be that $c_k >_C c_j$.

We have reached a contradiction and have exhausted all cases: there are no markedness or single–state faithfulness constraints that satisfy (1–4). Hence, it is not the case both that ⟨raɪtər, rʌɪrər⟩ is optimal in $\mathcal{G}$ and that ⟨raɪdər, raɪrər⟩ is optimal in $\mathcal{G}$.

□

From Theorem 1, together with the rule–based analysis above, the claim at the start of this section follows easily.

Theorem 2. *There is a relation R such that (i) R is regular, and (ii) there is no classic OT grammar $\mathcal{G}$ such that each $\langle i, o\rangle \in R$ is optimal in $\mathcal{G}$.*

Proof. Let $R = \{\langle$raɪtər, rʌɪrər$\rangle, \langle$raɪdər, raɪrər$\rangle\}$. For part (i), since there are ordered rewrite rules ($\mathcal{R}$ and $\mathcal{F}$ above) that map /raɪtər/ to [rʌɪrər] and map /raɪdər/ to [raɪrər], it follows that R is regular. Part (ii) follows from Theorem 1.
□

4 Discussion

In the previous section, I proved that Canadian raising, which is an example of counterbleeding on environment opacity, is expressible by ordered rewrite rules but not by classic OT grammars. I now show that there are other examples of counterbleeding on environment opacity, as well as examples of counterfeeding on environment opacity, that are likewise inexpressible by classic OT grammars. Though the patterns are all different, I discuss several formal properties that they share.[12]

4.1 Other Examples of Counterbleeding on Environment Opacity

Polish. Consider the data from Polish in Fig. 6 [12, 18, 19]. All four patterns can be captured by a rule $\mathcal{R}$ of raising /o/ to [u] before voiced segments (/l, b/), ordered before a rule $\mathcal{D}$ of devoicing word–final obstruents, i.e. mapping word–final /z, b/ to [s, p]. $\mathcal{D}$ counterbleeds on $\mathcal{R}$'s environment because $\mathcal{D}$ removes part of $\mathcal{R}$'s environment (__ l, b) by mapping /b/ to [p].

The patterns /ʒwob/ → [ʒwup] and /ʒwop/ → [ʒwop] cannot be expressed by any classic OT grammar. The proof is essentially the same as that of Theorem 1, and the reason is that the Canadian raising patterns and the Polish patterns are formally almost identical. In Canadian raising, /aɪ/ surfaces as [ʌɪ] before /t/, which (along with /d/) surfaces as [ɾ]. In Polish, /o/ surfaces as [u] before /b/, which (along with /p/) surfaces as [p]. The difference is that in Canadian raising, /t, d/ surface as a third segment, [ɾ], whereas in Polish, /b, p/ surface as [p].

[12] In each of the following examples, I employ a hypothetical input that is not actually part of that language's lexicon, as far as I know. Nonetheless, I take such absences to simply be accidental lexical gaps. Moreover, the formal result that these sets of input–outputs patterns are expressible by ordered rules but not by classic OT still holds; the empirical question of whether the patterns are attested is a separate issue.

	Input		Output	
a.	/sol/	→	[sul]	"rubble"
b.	/ɡruz/	→	[ɡrus]	"salt"
c.	/ʒwob/	→	[ʒwup]	"crib"
d.	/ʒwop/	→	[ʒwop]	hypothetical

Fig. 6. Polish data

	Input		Output	
a.	/ħaːkimiːn/	→	[ħaːkʲmiːn]	"ruling (masc. pl.)
b.	/ħaːkmiːn/	→	[ħaːkmiːn]	hypothetical

Fig. 7. Bedouin Arabic data 1

Bedouin Arabic 1. Consider the data from Bedouin Arabic in Fig. 7 [11]. Both patterns are expressible by a rule $\mathcal{P}$ of palatalizing /k/ to [kʲ] before /i/, ordered before a rule $\mathcal{D}$ of deleting /i/ (mapping /i/ to ϵ). $\mathcal{D}$ counterbleeds on $\mathcal{P}$'s environment because $\mathcal{D}$ removes part of $\mathcal{P}$'s environment (__ i) by deleting /i/. These two patterns cannot be expressed by any classic OT grammar, and the reason, again, is that these patterns are formally similar to those in Canadian raising: /k/ surfaces as [kʲ] before /i/, which (along with ϵ) surfaces as ϵ.

4.2 Examples of Counterfeeding on Environment Opacity

Isthmus Nahuat. Consider the data from Isthmus Nahuat in Fig. 8 [16, 19, 20]. All four patterns are expressible by a rule $\mathcal{D}$ of devoicing /l/ to [l̥] word–finally, ordered before a rule $\mathcal{A}$ of apocope, i.e. deleting /i/ word–finally. $\mathcal{A}$ counterfeeds on $\mathcal{D}$'s environment because $\mathcal{A}$ creates part of $\mathcal{D}$'s environment (__ #, where # denotes a word boundary) by deleting a word–final /i/ (mapping i# to ϵ#).

The patterns /ʃikɑkíli/ → [ʃikɑkíl] and /ʃikɑkíl/ → [ʃikɑkíl̥] are not expressible by any classic OT grammar. The proof, omitted here, is essentially the same as that of Theorem 1 for counterbleeding on environment opacity, using ⟨ʃikɑkíli, ʃikɑkíl⟩, ⟨ʃikɑkíli, ʃikɑkíl̥⟩, ⟨ʃikɑkíl, ʃikɑkíl⟩, and ⟨ʃikɑkíl, ʃikɑkíl̥⟩ as the relevant members of Gen. Informally, the reason that these patterns are inexpressible is the following: given the evidence from /ʃikɑkíli/ that [ʃikɑkíl] is an unmarked output, it is always more optimal to map /ʃikɑkíl/ to [ʃikɑkíl], incurring no faithfulness violations, than to map /ʃikɑkíl/ to [ʃikɑkíl̥], incurring one faithfulness violation (l → l̥). Or, in terms of the formalism presented here, /i/ surfaces as ϵ word–finally, and /l/ surfaces as [l̥], but only if /l/ does not precede a word–final /i/; and this latter condition on the identity of /l/ can be expressed only by a multistate faithfulness constraint.

Bedouin Arabic 2. Fig. 9 presents more data from Bedouin Arabic [11]. Both patterns can be captured by a rule $\mathcal{R}$ of raising /a/ to [i] in an open syllable,

	Input		Output	
a.	/támi/	→	[tám]	"it ends"
b.	/tájoːl/	→	[tájoːl̥]	"shelled corn"
c.	/ʃikɑkíli/	→	[ʃikɑkíl]	"put it in it"
d.	/ʃikɑkíl/	→	[ʃikɑkíl̥]	hypothetical

Fig. 8. Isthmus Nahuat data

	Input		Output	
a.	/gabr/	→	[gabur]	"grave"
b.	/gabur/	→	[gibur]	hypothetical

Fig. 9. Bedouin Arabic data 2

ordered before a rule $\mathcal{E}$ of epenthesis, i.e. inserting a [u] (mapping ϵ to [u]) between two consonants like /br/. $\mathcal{E}$ counterfeeds on $\mathcal{R}$'s environment because $\mathcal{E}$ creates part of $\mathcal{R}$'s environment (__ bV) by inserting a vowel, [i], after [b]. These two patterns cannot be expressed by any classic OT grammar, and the reason is that these patterns are formally similar to those in Isthmus Nahuat. In Bedouin Arabic, ϵ surfaces as [u] between /br/, and /a/ surfaces as [i], but only when /a/ does not precede a consonant cluster.

Lomongo. Consider, finally, the data from Lomongo in Fig. 10 [12, 19, 21]. Both patterns can be captured by a rule $\mathcal{G}$ of prevocalic gliding, i.e. mapping /o/ to [w] before a vowel, ordered before a rule $\mathcal{D}$ of intervocalic obstruent deletion, i.e. deleting /b/ between two vowels. $\mathcal{D}$ counterfeeds on $\mathcal{G}$'s environment because $\mathcal{D}$ creates part of $\mathcal{G}$'s environment (__ V) by deleting a prevocalic /b/, i.e. by mapping /bV/ to just [V]. These patterns are not expressible by any classic OT grammar, and the reason, once more, is that these patterns are similar to those in Isthmus Nahuat. In this case, /b/ surfaces as ϵ between two vowels, and /o/ surfaces as [w], but only when /o/ does not precede a prevocalic /b/.

Summary. Fig. 11 summarizes all of the sets of input–output candidate pairs that have been discussed in this paper, with the empty string symbol (ϵ) included where appropriate. Each of the six sets exhibits the following three properties: (i) the two inputs differ by exactly one segment x; (ii) the two output candidates differ by exactly one segment y; and (iii) x is different from y, i.e. they occupy different string positions. (In Fig. 11, the differing segments are in bold.) Formally, then, these sets are essentially the same; hence, it is not surprising that in all six examples the relevant input–output patterns cannot be made optimal by classic OT grammars, as defined here. Moreover, from this perspective it emerges that the difference between the counterbleeding on environment examples and the counterfeeding on environment examples boils down to which two (of the four possible) input–output candidate pairs are supposed to be optimal.

	Input		Output	
a.	/obina/	→	[oina]	"you (sg.) dance"
b.	/oina/	→	[wina]	hypothetical

Fig. 10. Lomongo data

Language	Inputs		Outputs	
Canadian English	raɪ**t**ər	raɪ**d**ər	r**aɪ**rər	r**ʌɪ**rər
Polish	ʒwo**b**	ʒwo**p**	ʒw**o**p	ʒw**u**p
Bedouin Arabic 1	ħaː**k**iməːn	ħaː**k**ɛməːn	ħaː**kʲ**əməːn	ħaː**kʲ**ɛməːn
Isthmus Nahuat	ʃikɑkíl**i**	ʃikɑkíl**ɛ**	ʃikɑkíl**ɛ**	ʃikɑkíl̥**ɛ**
Bedouin Arabic 2	gab**ɛ**r	gab**u**r	g**a**bur	g**i**bur
Lomongo	o**b**ina	o**ɛ**ina	**o**ɛina	**w**ɛina

Fig. 11. A summary of all the input–output candidate pairs in this paper

Notably, the proof technique used in this paper (finding two conflicting input–output patterns) does not seem to work for cases of focus opacity. To give just one example of counterfeeding on focus opacity, in Western Basque [22], /a/ raises to [e], and /e/ raises to [i], but /a/ does not raise to [i]. To express the latter generalization in OT, it suffices to posit an undominated, single–state faithfulness constraint that assigns 1 to the pair a→i and 0 to every other pair. Such a constraint may seem intuitively ad hoc, yet it is formally sound.

More generally, the reason that focus opacity seems unproblematic for OT from this paper's perspective is twofold: (i) classic faithfulness constraints, as defined here, are single–state faithfulness constraints that can penalize arbitrary, single input–output segment pairs; and (ii) in focus opacity, all change occurs at a single focus. Thus, it suffices to posit a single–state faithfulness constraint that assigns 1 violation to those focus changes that are undesired, like a→i, and 0 to those that are desired. Moreover, taking into account Fig. 11 and the three properties of environment opacity mentioned above, it comes as no surprise that the focus opacity of Western Basque works differently: it necessarily lacks at least one of the three properties. Specifically, if the first two properties hold, i.e. the relevant inputs differ by exactly one segment x and the relevant output candidates differ by exactly one segment y, then the third property, according to which x and y occupy different string positions, must not hold, because in focus opacity the segments that change (a, e, i) all occupy the same string position.

5 Conclusion

In this paper I defined a classic OT grammar as any OT grammar that contains only markedness constraints, i.e. constraints representable by an FST that is input–*independent*, and single–state faithfulness constraints, i.e. constraints representable by an FST that is both input–*dependent* and *single*–state. I then

proved, using data from Canadian English, that there are input–output patterns which can be expressed by SPE–style, ordered, context–sensitive rewrite rules, but which cannot be expressed by *any* classic OT grammar (Theorem 1). Hence, there are regular relations that classic OT grammars, as defined here, cannot express (Theorem 2). I also demonstrated that several other cases of counterbleeding on environment opacity, as well as several cases of counterfeeding on environment opacity, are likewise expressible by ordered rewrite rules but not by classic OT. Lastly, I argued that focus opacity, unlike environment opacity, seems unproblematic for classic OT grammars that allow arbitrary single–state faithfulness constraints.

Regarding empirical coverage, assuming that the Canadian raising data are attested, then classic OT, as defined here, undergenerates. Assuming also that natural language phonology is strictly subregular (see [5]), then, since I did not place any upper bound on the number of violations that any constraint may assign, classic OT also overgenerates: it can express unattested, non–regular patterns (see footnote 3). Ordered rewrite rules, under the same assumptions, only overgenerate.[13] However, whether or not this serves as a basis to favor rules over classic OT remains to be seen: it could be, for example, that the classes of patterns by which classic OT grammars over– and undergenerate are in some formal sense smaller (or more manageable) than the class of patterns by which rules overgenerate.

Acknowledgments. I thank Morgan Sonderegger for his supervision at every stage of this research; Brendan Gillon, Heather Goad, and Michael Wagner for helpful discussions; and three anonymous referees for Formal Grammar and two anonymous referees for Mathematics of Language for their comments, which included pointing out an incorrect claim in an earlier version of this paper.

References

1. Chomsky, N., Halle, M.: The Sound Pattern of English. The MIT Press (1968)
2. Prince, A., Smolensky, P.: Optimality Theory: Constraint Interaction in Generative Grammar. Blackwell Publishers (2004)
3. Johnson, C.D.: Formal Aspects of Phonological Description. Mouton (1972)
4. Kaplan, R.M., Kay, M.: Regular models of phonological rule systems. Computational Linguistics 20(3), 331–378 (1994)
5. Heinz, J.: Computational phonology—part I: Foundations. Language and Linguistics Compass 5, 140–152 (2011)
6. Gerdemann, D., Hulden, M.: Practical finite state optimality theory. In: Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing, pp. 10–19 (2012)
7. Riggle, J.: Generation Recognition and Learning in Finite State Optimality Theory. PhD thesis, UCLA (2004)
8. Frank, R., Satta, G.: Optimality Theory and the generative complexity of constraint violability. Computational Linguistics 24(2), 307–315 (1998)

[13] I thank an anonymous referee for raising these points.

9. Karttunen, L.: The proper treatment of optimality in computational phonology. In: Proceedings of the International Workshop on Finite State Methods in Natural Language Processing, Association for Computational Linguistics, pp. 1–12 (1998)
10. Heinz, J.: Computational phonology—part II: Grammars, learning, and the future. Language and Linguistics Compass 5(4), 153–168 (2011)
11. McCarthy, J.: Hidden Generalizations: Phonological Opacity in Optimality Theory. Equinox Publishing Ltd. (2007)
12. Baković, E.: Opacity and ordering. In: Goldsmith, J.A., Riggle, J., Yu, A.C.L. (eds.) The Handbook of Phonological Theory, 2nd edn., Blackwell Publishers (2011)
13. McCarthy, J.: Sympathy and phonological opacity. Phonology 16, 331–399 (1999)
14. Kiparsky, P.: Historical linguistics. In: Dingwall, W.O. (ed.) A Survey of Linguistic Science. University of Maryland Linguistics Program, College Park (1971)
15. Kiparsky, P.: Abstractness, opacity, and global rules. In: Fujimura, O., Smith, D.L. (eds.) Three Dimensions of Linguistic Theory, TEC (1973)
16. Kager, R.: Optimality Theory. Cambridge University Press (1999)
17. Graf, T.: Reference–set constraints as linear tree transductions via controlled optimality systems. In: de Groote, P., Nederhof, M.-J. (eds.) Formal Grammar 2010/2011. LNCS, vol. 7395, pp. 97–113. Springer, Heidelberg (2012)
18. Bethin, C.Y.: Phonological rules in the nominative singular and genitive plural of the Slavic substantive declension. PhD thesis, University of Illinois, Champaign–Urbana (1978)
19. Kenstowicz, M., Kisseberth, C.: Generative Phonology: Description and Theory. Academic Press (1979)
20. Law, H.: Morphological structure of Isthmus Nahuat. International Journal of American Languages 24, 108–129 (1958)
21. Hulstaert, G.: Grammaire du Lomongo. Musée royal de l'Afrique centrale (1961)
22. de Rijk, R.: Vowel interaction in Bizcayan Basque. Fontes Linguae Vasconum 2(5), 149–167 (1970)

Adjectives in a Modern Type-Theoretical Setting*

Stergios Chatzikyriakidis[1] and Zhaohui Luo[2]

[1] Dept. of Computer Science, Royal Holloway, Univ. of London
Egham, Surrey TW20 0EX, U.K. Open University of Cyprus
stergios.chatzikyriakidis@cs.rhul.ac.uk
[2] Dept. of Computer Science, Royal Holloway, Univ. of London
Egham, Surrey TW20 0EX, U.K.
zhaohui@hotmail.ac.uk

Abstract. In this paper we discuss the semantics of adjectives from the perspective of a Modern Type Theory (MTT) with an adequate subtyping mechanism. In an MTT, common nouns (CNs) can be interpreted as types and, in particular, CNs modified by intersective and subsective adjectives can be given semantics by means of Σ-types. However, an interpretation of CNs as types would not be viable without a proper notion of subtyping which, as we explain, is given by coercive subtyping, an adequate notion of subtyping for MTTs. It is also shown that suitable uses of universes are one of the key ingredients that have made such an analysis adequate. Privative and non-committal adjectives require different treatments than the use of Σ-types. We propose to deal with privative adjectives using the disjoint union type while non-committal adjectives by making use of the type-theoretical notion of context, as used by Ranta [27] to approximate the model-theoretic notion of a possible world. Our approach to adjectives has a number of advantages over those proposed within the Montagovian setting, one of which is that the inferences related with the adjectives arise via typing and not by some kind of extra semantic meaning in the form of a meaning postulate.

1 Introduction

The semantics of adjectives is a well-studied issue in the Montagovian tradition and a number of proposals have been put forth by the years (see e.g.[22],[10] [24] and [25] among others). Another prominent line of research on adjectives is based on Davidson's [6] treatment of adverbials and adjectives. In these approaches, the semantics of adverbial and adjectival modification are derived by exploiting the additional event argument assumed in Davidsonian semantics (see for example [11]). In modern Type Theories (MTTs), i.e. TTs within the tradition of Martin-Löf, adjectival modification has been treated as a Σ-type (see e.g. [27], [15]). However, such an approach in the form it is proposed (e.g. as in [27]) can only

* This work is supported by the research grant F/07-537/AJ of the Leverhulme Trust in the U.K.

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 159–174, 2013.

deal with subsective adjectives. The intersective adjectival class can be treated with Σ-types, but as we shall see, not in the way proposed in the literature.

In this paper, we discuss the issue of adjectival modification within an MTT equipped with an adequate subtyping mechanism (coercive subtyping). We first show that a Σ-type analysis can accommodate both intersective and subsective adjectives. This is due to the subtyping mechanism as well as the use of the universe CN of (the interpretations of) common nouns for the cases where these are needed.[1] In order to deal with the privative class, we propose to treat privative adjectival modification via disjoint union types. Such a move is quite close (at least on a pre-theoretical level) to Partee's treatment of adjectives like *fake* as being subsective but applied to CNs with coerced meanings [25]. Lastly, the case of non-committal adjectives is discussed arguing that one can have an adequate MTT account by exploiting the constructive notion of context.

The paper is structured as follows: in §2, we introduce the framework to be used, concentrating on the features that are relevant for the treatment of adjectives. Starting from §3 to §5, we shall study intersective/subsective, privative and non-committal adjectives, respectively. In §3 we consider the Σ-type analysis of CNs modified by intersective and subsective adjectives and discuss how subsective adjectives should be dealt with in such a context. In §4, by further developing a proposal by the second author [19], we study how privative adjectives may be interpreted by means of disjoint union types together with coercive subtyping. Non-committal adjectives are studied in §5, where it is shown that they may be given semantics by considering Ranta's formulation of belief contexts [27]. Finally, in §6, adjectives like *former* are briefly studied as special temporal cases, while we further make a proposal on how time may be incorporated by means of dependent types.

2 An MTT with Coercive Subtyping

In this section, we give a brief introduction to the formal semantics based on Modern Type Theories (MTTs) [27,14,17]. A Modern Type Theory (MTT) is a variant of a class of type theories as studied by Martin-Löf [20,21] and others, which have dependent types and inductive types, among others. We choose to call them Modern Type Theories in order to distinguish them from Church's simple type theory [5] that is commonly employed within the Montagovian tradition in formal semantics.

Among the variants of MTTs, we are going to employ the Unified Theory of dependent Types (UTT) [12] with the addition of the coercive subtyping mechanism (see, for example, [13,18] and below). UTT is an impredicative type theory in which a type of all logical propositions ($Prop$) exists.[2] This stands as part of the study of linguistic semantics using MTTs rather than simple typed ones, including the early studies such as [28,27] inter alios.

[1] See §2.4 for the notion of a universe.

[2] This is similar to the simple type theory where there is a type t of truth values.

	Example	Montague semantics	MTT-based Semantics
CN	man, human	$[\![man]\!],\ [\![human]\!] : e \to t$	$[\![man]\!],\ [\![human]\!] : Type$
IV	talk	$[\![talk]\!] : e \to t$	$[\![talk]\!] : [\![human]\!] \to Prop$
ADJ	handsome	$[\![handsome]\!] : (e \to t) \to (e \to t)$	$[\![handsome]\!] : [\![man]\!] \to Prop$
MCN	handsome man	$[\![handsome]\!]([\![man]\!]) : e \to t$	$\Sigma m : [\![man]\!] . [\![handsome]\!](m) : Type$
S	A man talks	$\exists m : e.\ [\![man]\!](m) \& [\![talk]\!](m) : t$	$\exists m : [\![man]\!] . [\![talk]\!](m) : Prop$

Fig. 1. Examples in formal semantics

2.1 Formal Semantics Based on MTTs: The Basics

In MTT-based semantics, the basic ways to interpret various linguistic categories is as follows:[3]

- A sentence (S) is interpreted as a proposition of type $Prop$.
- A common noun (CN) can be interpreted as a type.
- A verb (IV) can be interpreted as a predicate over the type D that interprets the domain of the verb (ie, a function of type $D \to Prop$).
- An adjective (ADJ) can be interpreted as a predicate over the type that interprets the domain of the adjective (ie, a function of type $D \to Prop$).
- Modified common nouns (MCNs) can be interpreted by means of Σ-types (see below).

In what follows, we shall give further explanations of various aspects of semantics based on MTTs, explicating along the way the basic features of MTTs and coercive subtyping. We try to bring out the linguistic relevance of the system used rather than being meticulous as regards the formal details in each case.

2.2 Common Nouns as Types and Many-Sortedness of MTTs

A key difference between the formal semantics based on MTTs on the one hand and Montague semantics on the other, lies in the interpretation of common nouns (CNs). This is in turn based on the fact that MTTs are essentially 'many-sorted' logical systems.

In Montague semantics [23], the underlying logic (Church's simple type theory [5]) can be seen as 'single-sorted' in the sense that there is only one type e of all entities. The other types such as t of truth values and the function types generated from e and t do not stand for types of entities. In this respect, there are no fine-grained distinctions between the elements of type e and as such all individuals are interpreted using the same type. For example, *John* and *Mary* have the same type in simple type theories, the type e of individuals. An MTT, on the other hand, can be regarded as a 'many-sorted' logical system in that it contains many types and. In this respect, in an MTT-based semantics one can make fine-grained distinctions between individuals and use those different types

[3] Basic examples are shown in Figure 1, along with a comparison with their counterparts in Montague semantics.

to interpret subclasses of individuals. For example, we can have $John : [\![man]\!]$ and $Mary : [\![woman]\!]$, where $[\![man]\!]$ and $[\![woman]\!]$ are different types.

An important trait of MTT-based semantics is the interpretation of common nouns (CNs) as *types* [27] rather than sets or predicates (i.e., objects of type $e \rightarrow t$) as in Montague semantics. The CNs man, human, table and book are interpreted as types $[\![man]\!]$, $[\![human]\!]$, $[\![table]\!]$ and $[\![book]\!]$, respectively. Then, individuals are interpreted as being of one of the types used to interpret CNs.

Modified common nouns (MCNs in Figure 1) can be interpreted by means of Σ-types, types of dependent pairs. For instance, 'handsome man' can be interpreted as the type $\Sigma m : [\![man]\!] . [\![handsome]\!](m)$, the type of pairs of a man and a proof that the man is handsome.

This many-sortedness (i.e., the fact that there are many types in an MTT) has the welcoming result that a number of semantically infelicitous sentences like e.g. *the ham sandwich walks*, which are however syntactically well-formed, can be explained easily given that a verb like walks will be specified as being of type $Animal \rightarrow Prop$ while the type for *ham sandwich* will be $[\![food]\!]$ or $[\![sandwich]\!]$, which is not compatible with the typing for *walks*:[4]

(1) $the\ ham\ sandwich : [\![food]\!]$
(2) $walk : [\![human]\!] \rightarrow Prop$

The idea of common nouns being interpreted as types rather than predicates has been argued in [16] on philosophical grounds as well. There, the author argues that Geach's observation that common nouns, in contrast to other linguistic categories, have criteria of identity that enable common nouns to be compared, counted or quantified, has an interesting link with the constructive notion of set/type: in constructive mathematics, sets (types) are not constructed only by specifying their objects but they additionally involve an equality relation. The argument is then that the interpretation of CNs as types in MTTs is explained and justified to a certain extent.[5]

Interpreting CNs as types rather than predicates has also a significant methodological implication: this is compatible with various subtyping relations one may consider in formal semantics. For instance, in modelling some linguistic phenomena semantically, one may introduce various subtyping relations by postulating a collection of subtypes (physical objects, informational objects, eventualities, etc.) of the type of entities [1]. It has become clear that, if CNs are interpreted as predicates as in the traditional Montagovian setting, introducing such subtyping relations would cause difficult problems: even some basic semantic interpretations would go wrong and it is very difficult to deal with some linguistic phenomena such as copredication satisfactorily. Instead, if CNs are interpreted as types, as in the type-theoretical semantics based on MTTs, copredication can be given a straightforward and satisfactory treatment [14].

[4] This is of course based on the assumption that the definite NP is of a lower type and not a Generalized Quantifier.
[5] See [16] for more details on this.

2.3 Subtyping in Formal Semantics

As briefly explained above, because of many-sortedness of MTTs, CNs can be interpreted as types. For instance, in a Montagovian setting, all of the verbs below are given the same type $e \to t$, but in an MTT, we can have

(3) $drive : [\![human]\!] \to Prop$

(4) $eat : [\![animal]\!] \to Prop$

(5) $disappear : [\![object]\!] \to Prop$

which have different domain types. This has the advantage of disallowing interpretations of some infelicitous examples like *the ham sandwich walks*.

However, interpreting CNs by means of different types could lead to serious undergeneralizations without a subtyping mechanism: *subtyping* is crucial for an MTT-based semantics. For instance, consider the interpretation of the sentence 'A man talks' in Figure 1: for m of type $[\![man]\!]$ and $[\![talk]\!]$ of type $[\![human]\!] \to Prop$, the function application $[\![talk]\!](m)$ is only well-typed because we have that $[\![man]\!]$ is a subtype of $[\![human]\!]$.

Coercive subtyping [13,18] provides an adequate framework to be employed for MTT-based formal semantics [14,17].[6] It can be seen as an abbreviation mechanism: A is a (proper) subtype of B ($A < B$) if there is a unique implicit coercion c from type A to type B and, if so, an object a of type A can be used in any context $\mathfrak{C}_B[\_]$ that expects an object of type B: $\mathfrak{C}_B[a]$ is legal (well-typed) and equal to $\mathfrak{C}_B[c(a)]$.

As an example, in the case that both $[\![man]\!]$ and $[\![human]\!]$ are base types, one may introduce the following as a basic subtyping relation:

(6) $[\![man]\!] < [\![human]\!]$

In case that $[\![man]\!]$ is defined as a composite Σ-type (see §2.4 below for details), where $male : [\![human]\!] \to Prop$:

(7) $[\![man]\!] = \Sigma h : [\![human]\!] \,.\, male(h)$

we have that (6) is the case because the above Σ-type is a subtype of $[\![human]\!]$ via the first projection π_1:

(8) $(\Sigma h : [\![human]\!] \,.\, male(h)) <_{\pi_1} [\![human]\!]$

Equipped with this coercive subtyping mechanism, the undergeneration problems can be straightforwardly solved while still retaining the ability to rule out semantically infelicitous cases like *the ham sandwich walks*. In effect, many-sortedness in MTTs turns out to be superior than single sortedness in simple

[6] It is worth mentioning that subsumptive subtyping, the traditional notion of subtyping that adopts the subsumption rule (if $A \leq B$, then every object of type A is also of type B), is inadequate for MTTs in the sense that it would destroy some important properties of MTTs (see, for example, §4 of [18] for details).

type theory (at least in this respect). Furthermore, many inferences concerning the monotonicity on the first argument of generalized quantifiers can be directly captured using the subtyping mechanism. In effect an inference of the sort exemplified in the example (12) below, can be captured given that $[\![man]\!] < [\![human]\!]$:

(9) Some man runs ⇒ Some human runs

Thus, an $x : [\![man]\!]$ can be used as an $x : [\![human]\!]$, and as such the inference goes through for 'free' in a way.[7]

2.4 Σ-Types, Π-Types and Universes

Dependent Σ-types. One of the basic features of MTTs is the use of Dependent Types. A dependent type is a family of types that depend on some values.

The constructor/operator Σ is a generalization of the Cartesian product of two sets that allows the second set to depend on values of the first. For instance, if $[\![human]\!]$ is a type and $male : [\![human]\!] \rightarrow Prop$, then the Σ-type $\Sigma h : [\![human]\!] . male(h)$ is intuitively the type of humans who are male.

More formally, if A is a type and B is an A-indexed family of types, then $\Sigma(A, B)$, or sometimes written as $\Sigma x{:}A.B(x)$, is a type, consisting of pairs (a, b) such that a is of type A and b is of type $B(a)$. When $B(x)$ is a constant type (i.e., always the same type no matter what x is), the Σ-type degenerates into product type $A \times B$ of non-dependent pairs. Σ-types (and product types) are associated projection operations π_1 and π_2 so that $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$, for every (a, b) of type $\Sigma(A, B)$ or $A \times B$.

The linguistic relevance of Σ-types can be directly appreciated once we understand that in its dependent case, Σ-types can be used to interpret linguistic phenomena of central importance, like adjectival modification (see above for interpretation of modified CNs) [27].[8] For example, *handsome man* is interpreted as Σ-type (10), the type of handsome men (or more precisely, of those men together with proofs that they are handsome):

(10) $\Sigma m : [\![man]\!] . [\![handsome]\!](m)$

where $[\![handsome]\!](m)$ is a family of propositions/types that depends on the man m.

The use of Σ-types for dealing with adjectival modification will be further explained later on, when our proposal as regards the different classes of adjectives is going to be discussed.

[7] These kinds of inferences can be straightforwardly proven in Coq by using a standard analysis for quantifier *some* plus the subtyping relation $[\![man]\!] < [\![human]\!]$. See [4] for more details on treating NLIs as valid theorems in Coq.

[8] Σ-types also provide tools to give proper semantic interpretations of the so-called "Donkey-sentences" [28].

Dependent Π-types. The other basic constructor for dependent types is Π. Π-types can be seen as a generalization of the normal function space where the second type is a family of types that might be dependent on the values of the first. A Π-type degenerates to the function type $A \to B$ in the non-dependent case. In more detail, when A is a type and P is a predicate over A, $\Pi x{:}A.P(x)$ is the dependent function type that, in the embedded logic, stands for the universally quantified proposition $\forall x{:}A.P(x)$. For example, the following sentence (11) is interpreted as (12):

(11) Every man walks.

(12) $\Pi x : [\![man]\!] \,.\, [\![walk]\!](x)$

Π-types are very useful in formulating the typings for a number of linguistic categories like VP adverbs or quantifiers. The idea is that adverbs and quantifiers range over the universe of (the interpretations of) CNs and as such we need a way to represent this fact. In this case, Π-types can be used, universally quantifying over the universe CN. (13) the type for VP adverbs[9] while (14) is the type for quantifiers:

(13) $\Pi A : \text{CN}.\ (A \to Prop) \to (A \to Prop)$

(14) $\Pi A : \text{CN}.\ (A \to Prop) \to Prop$

Further explanations of the above types are given after we have introduced the concept of type universe below.

Type Universes. An advanced feature of MTTs, which will be shown to be very relevant in interpreting NL semantics in general as well as adjectival modification specifically, is that of universes. Informally, a universe is a collection of (the names of) types put into a type [21].[10] For example, one may want to collect all the names of the types that interpret common nouns into a universe $\text{CN} : Type$. The idea is that for each type A that interprets a common noun, there is a name $\overline{A}$ in CN. For example,

$$\overline{[\![man]\!]} : \text{CN} \quad \text{and} \quad T_{\text{CN}}(\overline{[\![man]\!]}) = [\![man]\!] \,.$$

In practice, we do not distinguish a type in CN and its name by omitting the overlines and the operator T_{CN} by simply writing, for instance, $[\![man]\!] : \text{CN}$. Thus, the universe includes the collection of the names that interpret common nouns. For example, in CN, we shall find the following types:

(15) $[\![man]\!]$, $[\![woman]\!]$, $[\![book]\!]$, ...

[9] This was proposed for the first time in [15].

[10] There is quite a long discussion on how these universes should be like. In particular, the debate is largely concentrated on whether a universe should be predicative or impredicative. A strongly impredicative universe U of all types (with $U : U$ and Π-types) is shown to be paradoxical [7] and as such logically inconsistent. The theory UTT we use here has only one impredicative universe $Prop$ (representing the world of logical formulas) together with an infinitely many predicative universes which as such avoids Girard's paradox (see [12] for more details).

(16) $\Sigma m : [\![man]\!] . [\![handsome]\!](m)$

(17) $G_R + G_F$

where the Σ-type in (16 is the proposed interpretation of 'handsome man' and the disjoint union type in (17) is that of 'gun' (the disjoint union of real guns and fake guns – see the discussion in §4).

Having introduced the universe CN, it is now possible to explain (13) and (14). The type in (14) says that for all elements A of type CN, we get a function type $(A \rightarrow Prop) \rightarrow Prop$. The idea is that the element A is now the type used. To illustrate how this works let us imagine the case of quantifier *some* which has the typing in (14). The first argument we need, has to be of type CN. Thus *some human* is of type $([\![human]\!] \rightarrow Prop) \rightarrow Prop$ given that the A here is $[\![human]\!] : \text{CN}$ (A becomes the type $[\![human]\!]$ in $([\![human]\!] \rightarrow Prop) \rightarrow Prop$). Then given a predicate like $walk : [\![human]\!] \rightarrow Prop$, we can apply *some human* to get $[\![some\ human]\!]([\![walk]\!]) : Prop$.[11]

3 Σ-Type Analysis of Modified CNs

Not much work focusing on adjectives has been done in formal semantics based on modern type theories. Adjectives are mainly studied in a Σ-type analysis on modified common nouns [27,17], but not in general.[12] From this section on, we shall study adjectives in the MTT-based semantics more systematically.

In [27], the use of Σ-types to interpret common nouns modified by adjectives is proposed[13] and, in [14,17], it is pointed out that subtyping is essential for such an interpretation of CNs as types to be adequate and proposed that coercive subtyping provide such a framework where one can have $\Sigma x{:}N.Adj(x) < N$, where N interprets a CN and Adj an adjective that modifies the CN.[14]

The Σ-type treatment is quite straightforward. CNs are interpreted as types and adjectives as predicates. Given that one has many types, the type of a predicate that interprets an adjective can vary according to the adjective. Thus, for example, *black* will be of type $[\![object]\!] \rightarrow Prop$ while *married* of type

[11] The idea of universes has been proved useful in accounting for NL phenomena from an MTT perspective. For example, in [3], the authors introduce a universe of Linguistic Types, *LType*, to capture the flexibility associated with NL coordination and in this paper we are going to use the universe CN to deal with some cases of adjectival modification.

[12] There is the interesting work by Jespersen & Primiero [9] on using a constructive type theory to deal with different classes of adjectives. However, the system they have used seems quite far from being an MTT as we have considered; it lacks multiple types or a subtyping mechanism, and they have considered CNs as predicates (following pretty much all of the Montagovian literature) rather than types.

[13] In a more traditional logic, Gupta [8] has suggested a special form of formulae $(K, x)A$, called *restrictions*, for interpreting modified CNs. Linking formulae to types, we can easily see the close correspondence between $(K, x)A$ and $\Sigma x{:}K.A$.

[14] In the Coq proof assistant, the record types are (top-level) Σ-types and used in [15].

$[\![human]\!] \rightarrow Prop$.[15] Given subtyping, *black man* can still be interpreted as $\Sigma m{:}[\![man]\!] \,.\, [\![black]\!](m)$ because $[\![man]\!] < [\![object]\!]$ and hence, by contravariance, *black* of type $[\![object]\!] \rightarrow Prop$ can also be regarded as of type $[\![man]\!] \rightarrow Prop$.

Now, intersective adjectives are associated with two main types of inference. The first one is not specific to intersective adjectives but is rather shared with subsective adjectives as well. It involves the entailment shown below:

(18) Adj(N) $\Rightarrow$ N.

Thus, according to the above, a black man is a man, a married man is also a man, and so on. Given that one can always have that the first projection π_1 of Σ-types be a coercion (see §2.3) the following always holds:

(19) From $\Sigma(N, Adj)$, infer N.

So, this first type of inference is easily taken care of by subtyping.

The second inference associated with intersective adjectives has to do with the fact that intersective modification not only entails that a given x of $Adj(N)$ is a N (e.g. a black man is a man), but further entails that $Adj(x)$ is also the case (e.g. that a black man is something black). Now, what does this mean in terms of inference? It implies that, for example, in *black man*, *black* can not only be applied to men (objects of $[\![man]\!]$) but also to any object whose type is a supertype of $[\![man]\!]$. Furthermore, no interpretation arises for the types that have no subtyping relation with $[\![man]\!]$; for instance, if *beautiful* is interpreted of type $[\![wowan]\!] \rightarrow Prop$ and $[\![man]\!]$ is not a subtype of $[\![woman]\!]$, there is no interpretation of *beautiful man*. Also, it is straightforward to see that the non-exsistence of a subtyping relation prohibits one from unwanted inferences. The inferences below are illustrative of the phenomenon:[16]

(20) A black man $\Rightarrow$ a black human

(21) A black man $\not\Rightarrow$ a black woman

The analysis proposed captures this fact given that coercions propagate through the various type constructors as well, e.g. Σ and Π. As such, besides the relation $[\![man]\!] < [\![human]\!]$, the subtyping relation $\Sigma([\![man]\!], [\![black]\!]) < \Sigma([\![human]\!], [\![black]\!])$ also holds via coercion propagation. Thus, the Σ-type analysis provides us with all the correct inferences as regards intersective adjectives.

It is not difficult to notice that an approach like the one given above will not work for some of the subsective adjectives. This is because subsective adjectives do not give rise to entailments like (20). In this sense, one might very well

[15] The discussion on how one builds the type ontology is of great importance but it is something that cannot be discussed here.

[16] We are a bit informal here, as an anonymous reviewer has noticed, saying that we should also deal with the determiner *a* in these cases. The determiner *a* is interpreted as the existential quantifier whose type is the same as the other quantifiers, i.e. (14). The inference (20), for example, is just saying that if m is of type $[\![black\ man]\!]$, it is also of type $[\![black\ human]\!]$.

argue that the treatment proposed for intersective adjectives will overgenerate for subsective adjectives and as such the Σ-type analysis must be abandoned in these cases. However, this is not the case and as we are going to explain, the Σ-type analysis can be maintained in the case of subsective adjectives as well. Let us see how.

The reason why subsective adjectives do not give rise to inferences like (20) has to do with the fact that they are only relevant for a particular class of words (CNs) they modify. Thus, a skilful surgeon is only skilful as a surgeon and not as a man or a human being. Implementing this idea, subsective adjectives like *large* can be given the type below:

(22) $\Pi A : \text{CN}.\ (A \to Prop)$

Using the above type, we have many instances of *large* depending on the choice of A. $large(\llbracket man \rrbracket)$ is of type $\llbracket man \rrbracket \to Prop$, $large(\llbracket animal \rrbracket)$ is of type $\llbracket animal \rrbracket \to Prop$, and so on. In this respect, we get different 'larges' as such for different As. Using this, one can achieve the meaning of subsective adjectives, i.e. that if something is large, it is only large for its class denoted by the CN (a large elephant is thus only large as an elephant). This way of treating subsective adjectives will correctly account for the inferences associated with subsective adjectives. In particular, inferences like (18) are taken care of via the usual first projection coercion of the Σ-type, while inferences similar to (20) are avoided given that the adjective is only meaningful with respect to the specific class in each case.

However, we are not done yet. This is because, a type like the above, cannot be used for cases like *skilful*. The reason is that *skilful* cannot have such a general type. If we assume such a type, we will be able to get interpretations for *skilful rock* or *skilful car*, which does not seem correct. *Skilful* in this respect must apply to CNs of type $\llbracket human \rrbracket$ or subtypes of this latter type, e.g. $\llbracket doctor \rrbracket, \llbracket violonist \rrbracket < \llbracket human \rrbracket$. This problem can be solved as follows: one can introduce a subuniverse of CN containing the names of the types $\llbracket human \rrbracket$ and its subtypes only. Let us call this universe CN_H, which is a subtype of CN: $\text{CN}_H < \text{CN}$. Now, we can propose the following type for an adjective like skilful:

(23) $\Pi A : \text{CN}_H.\ (A \to Prop)$

Similar cases can be treated accordingly.

4 Privative Adjectives

Besides intersective and subsective adjectives, there is another adjectival class that does not give rise to any of the inferences associated with the aforementioned classes. This class of adjectives, is further subdivided into privative and non-committal adjectives. The former give rise to inferences like (24) while the latter do not give rise to any inference whatsoever:

(24) Adj(N) $\Rightarrow \neg$ N.

The standard way of dealing with privative adjectives as well as with the other classes of adjectives within the Montagovian tradition is via meaning postulates (see [24] for example). According to these types of approaches, the inferences are captured by postulating that certain types of adjectives are associated with the specific inferences. In the case of privative adjectives ADJ of type $(e \to t) \to (e \to t)$, the meaning postulate would be:

(25) $\forall Q : e \to t \forall x : e.\ ADJ(Q, x) \supset \neg Q(x)$

It is worth mentioning that meaning postulates are needed for all adjectival categories within a Montagovian setting, with an exception when one assumes that intersective adjectives be of a lower $e \to t$ type but again this has the disadvantage of disrupting type uniformity [24]. Partee in the same paper, and using data from Polish NP-split phenomena goes on to argue that the class of privative adjectives does not really exist. The reasoning in [24] as well as in [25] is that the interpretation of privative adjectives is in fact subsective. Partee argues that in cases of privative modification the interpretation of the CN is coerced to include the denotations of CNs modified by privative adjectives. For example in the case of (26) and (27), Partee argues that the denotation of *fur* is expanded to include both *real* and *fake* furs:

(26) I don't care whether that fur is fake fur or real fur.

(27) I don't care whether that fur is fake or real.

The idea is that in the case of *fake fur*, *fur* is coerced to include fake furs as well, while in the second case it is not. The idea in itself is very intriguing and indeed plausible given the data.

What we are going to propose is to use the disjoint union type in MTTs to formalise the semantics of privative adjectives. This was first proposed in an unpublished note by the second author [19], which can arguably be regarded as formalising the above idea of Partee in an MTT. Let us see how this can be done by discussing the case of fake and real guns.

We first assume that G_R and G_F be the type of (real) guns and that of fake guns, respectively. Then,

$$G = G_R + G_F$$

is the type of all guns. It consists of the objects of the form $inl(r)$ and $inr(f)$, where $r : G_R$ and $f : G_F$. Furthermore, we declare the associated injection operators $inl : G_F \to G$ and $inr : G_R \to G$ as coercions:

$$G_R <_{inl} G \quad and \quad G_F <_{inr} G.$$

We contend that the above employment of disjoint union type, together with the above declaration of subtyping relations, gives an adequate semantics of the privative adjective *fake*.

For instance, we can now define the following predicates $real\_gun$ and $fake\_gun$ of type $G \to Prop$:

$$real\_gun(inl(r)) = True \quad and \quad real\_gun(inr(f)) = False;$$
$$fake\_gun(inl(r)) = False \quad and \quad fake\_gun(inr(f)) = True.$$

If is easy to see that, for any $g : G$,

(28) $real\_gun(g)$ iff $\neg fake\_gun(g)$.

Now, the following interpretations can be given (both are true): for $g : G_R$:

(29) $[\![g \ is \ a \ real \ gun]\!] = real\_gun(g)$

and for $f : G_F$,

(30) $[\![f \ is \ not \ a \ real \ gun]\!] = \neg real\_gun(f)$

Note that in the above, $real\_gun(f)$ is only well-typed because $G_F <_{inr} G$ and in fact we have $real\_gun(f) = real\_gun(inr(f)) = True$. Similarly, with the above, it is not difficult to see that the sentences like those below can easily be interpreted as expected:

(31) Is that gun real or fake?

(32) A fake gun is not a gun.

In the above, we have only considered guns but not other objects. One may have the desire to type the word real and fake directly so that they can be applied to other objects different from guns. A possibility is to consider a type $Object$ (of all objects) of which, for example, G is a subtype:

$$G <_{gun} Object.$$

Employing $Object$, we could have:

(33) $real, \ fake : Object \to Prop$

and it is then easy to see that

$$G_R <_{gun \circ inl} Object \quad and \quad G_F <_{gun \circ inr} Object.$$

This allows us to give more general types (33) to *real* and *fake* so that we can cover cases like *fake car*, *real president* etc.

Please note that the above is also a rather welcomed result in that it predicts that a fake gun is an object (and not a fake object). It seems in this respect that the above MTT analysis of privative adjectives can produce further welcoming

results due to the nature of the subtyping mechanism. Other privative adjectives like *imaginary* can be treated accordingly.

5 Non-committal Adjectives

Privative adjectives, as already mentioned, comprise one of the subcategories of non-subsective adjectives, the other being the class of non-committal adjectives as these are usually called within the Montagovian tradition. In this category, we find modal adjectives like *alleged*, *possible* and *potential*. According to Partee [24], these are the only adjectives that do not give rise to any inferences at all:[17]

(34) Adj(N) $\Rightarrow$?.

Adjectives like *alleged* (and similar ones like *potential* and *possible*) involve a flavour of modality missing from the other classes of adjectives. Ranta [27] discusses the use of the notion of context in MTTs in order to deal with phenomena that have traditionally been dealt with using possible worlds in the model-theoretic tradition. Ranta in discussing the various issues associated with epistemic logic, proposes the notion of belief contexts: a belief context is a sequence of assumptions that an agent p has made. More precisely, the belief context of an agent p, notation Γ_p, is a context of the form:

(35) $\Gamma_p = x_1 : A_1, ..., x_n : A_n$

Based on this, Ranta proposes the belief operator B_p, defined as

$$B_p A = \Pi \Gamma_p.\ A = \Pi x_1{:}A_1 ... \Pi x_n{:}A_n.\ A.$$

As a consequence, $B_p A$ is true if and only if A is true in Γ_p.

Now, an adjective like alleged can be interpreted as follows. Let A_N : CN be the interpretation of a common noun N. Then, we interpret

(36) $[\![alleged\ N]\!] = \Sigma p{:}Human.\ B(p, A_N)$

where $B(p, A) = \Pi \Gamma_p.\ A$ with Γ_p being the belief context of $p : Human$.[18] Intuitively, the above says that, for some human being p, p believes that A_N (the semantics of N) is true.[19] For example, the following sentence (37) is interpreted as (38):

[17] This of course does not mean that they are devoid of meaning. This is a separate issue.

[18] Note that, strictly speaking, p in B_p is a meta-level entity; we are abusing the notation here. Formally, we can use a universe U that contains the Π-types and inductively define $B : Human \rightarrow U \rightarrow U$. Details are omitted.

[19] This is the analog of a formula that involves existential quantifications. One may turn such types into propositions by means of the following operation: for any type A, $Exists(A) = \exists x{:}A.True$. Then, with this mechanism, (36) can be represented as the proposition $\exists p{:}Human.\ Exists(B(p, A_N))$.

(37) John is an alleged criminal.

(38) $[\![John]\!] : [\![alleged\ criminal]\!] = \Sigma p : Human.\ B(p, [\![criminal]\!])$

Similar cases of adjectives seem in principle to be accountable within the same line of approach. On a more general note, the constructive notion of context that has been claimed by Ranta [26,27] to be the equivalent of the notion of a possible world in model-theoretic semantics is an idea that we believe needs to be taken into consideration more seriously. Such an approach may potentially provide us with a general account of intensionality. Indeed, a number of proposals have been put forth both by Ranta himself as well as other researchers building on work by Ranta.[20] We hope that our work will contribute towards this direction as well.

6 The Case of *Former*

In the last section, we shall deal with some temporal adjectives such as *former* and *past*. If we follow Partee [24] and assume that *former* behaves similarly to adjectives like *fake* or *imaginary*, then one is committed to a similar analysis for *former* as we have done in §4. Indeed one could propose an analysis for *former* within the same lines as the one proposed for *fake*, assuming the necessary modifications are made.

Another way to deal with *former* is not via the disjoint union type but rather via using an explicit $Time$ argument. Such an argument is independently needed if one wants to deal with any kind of tense or aspectual phenomenon. Whether this $Time$ parameter will be an argument of the verb or a parameter in a more complex argument, like for example an event argument is something that we will not discuss here. For the moment, let's assume a simple model of tense – a type $Time$ with an ordering relation $<$ (see, for example, [27]). Then we assume that some CNs are indexed by the time parameter. For example, instead just having a CN president, we have a family of types

$$president(t) : \text{CN},$$

indexed by $t : Time$. We further assume that $now : Time$ stand for the 'current time'; for example, $president(now)$ is the CN president at the current time.

With the above mechanisms available, we can now interpret CNs modified by former as follows: for example,[21]

(39) $[\![former\ president]\!] = \neg president(now) \wedge \exists t : Time.\ t < now \wedge president(t)$.

[20] See for example the work by [9] on adjectives like alleged or the work by [2] on NL phenomena involving beliefs.

[21] For understandability of the readers who are unfamiliar with MTTs, we abuse the notation here, using $\neg A$ to stand for $A \to \emptyset$, $\wedge$ for $\times$ and $\exists$ for Σ. One may ignore these formal details.

In general, we have $[\![former]\!] : (Time \to \text{CN}) \to \text{CN}$, obtained by abstracting *president* in the above definition:[22] for any $p : Time \to \text{CN}$,

(40) $[\![former]\!](p) = \neg p(now) \wedge \exists t : Time.\ t < now \wedge p(t)$.

With *president* : $Time \to \text{CN}$, we have $[\![former\ president]\!] = [\![former]\!]([\![president]\!])$.

The above use of dependent types in semantic interpretations may have the potential to be generalised. Further research is needed in this direction.

7 Conclusion

In this paper, we proposed an account of the various classes of adjectives within an MTT setting. We have shown that the Σ-type analysis for adjectives can cover the subsective and intersective classes adequately thanks to the use of the subtyping mechanism as well as the use of the notion of a universe. However, it was shown that privative adjectives require a different treatment and proposed to treat this type of adjectives via disjoint union types. This type of approach gives us the correct results as regards the inferences associated with these types of adjectives. Lastly, non-committal adjectives were discussed and an account that makes use of the constructive notion of context as approximating possible worlds was given.

References

1. Asher, N.: Lexical Meaning in Context: a Web of Words. Cambridge University Press (2012)
2. Boldini, P.: Formalizing context in intuitionistic type theory. Fundamenta Informaticae 42(2), 1–23 (2000)
3. Chatzikyriakidis, S., Luo, Z.: An account of natural language coordination in type theory with coercive subtyping. In: Proc. of Constraint Solving and Language Processing, Orleans (2012)
4. Chatzikyriakidis, S., Luo, Z.: Natural Language Inference Using Coq (2013) (manuscript)
5. Church, A.: A formulation of the simple theory of types. J. Symbolic Logic 5(1) (1940)
6. Davidson, D.: Compositionality and coercion in semantics: The semantics of adjective meaning. In: Rescher, N. (ed.) The Logical Form of Action Sentences, pp. 81–95. University of Pittsburgh Press (1967)

[22] Note that this type does not give rise to *Prop* after functional application but rather to CN. This is compatible with the fact that this type of adjectives cannot appear in predicative positions. In case one thinks that this is not a semantic issue but rather a syntactic one, one can use a slightly different definition so that $[\![former]\!]$ has type $(Time \to \text{CN}) \to Prop$, preserving a kind of type uniformity across all adjectival classes.

7. Girard, J.Y.: Une extension de l'interpretation fonctionelle de gödel à l'analyse et son application à l'élimination des coupures dans et la thèorie des types'. In: Proc. 2nd Scandinavian Logic Symposium, North-Holland (1971)
8. Gupta, A.: The Logic of Common Nouns. Yale University Press (1980)
9. Jespersen, B., Primiero, G.: Alleged assassins: realist and constructivist semantics for modal modification. In: Bezhanishvili, G., Lbner, S., Marra, V., Richter, F. (eds.) 9th International Tbilisi Symposium on Logic, Language, and Computation (2012)
10. Kamp, H.: Formal semantics of natural language. In: Keenan, E. (ed.) Two Theories About Adjectives, pp. 123–155. Cambridge University Press (1975)
11. Larson, R.: Events and modification in nominals. In: Proceedings of Semantics and Linguistic Theory (SALT) VIII, Tokyo (1998)
12. Luo, Z.: Computation and Reasoning: A Type Theory for Computer Science. Oxford Univ. Press (1994)
13. Luo, Z.: Coercive subtyping. Journal of Logic and Computation 9(1), 105–130 (1999)
14. Luo, Z.: Type-theoretical semantics with coercive subtyping. Semantics and Linguistic Theory 20 (SALT20), Vancouver (2010)
15. Luo, Z.: Contextual analysis of word meanings in type-theoretical semantics. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS, vol. 6736, pp. 159–174. Springer, Heidelberg (2011)
16. Luo, Z.: Common nouns as types. In: Béchet, D., Dikovsky, A. (eds.) LACL 2012. LNCS, vol. 7351, pp. 173–185. Springer, Heidelberg (2012)
17. Luo, Z.: Formal semantics in modern type theories with coercive subtyping. Linguistics and Philosophy 35(6) (2012)
18. Luo, Z., Soloviev, S., Xue, T.: Coercive subtyping: theory and implementation. Information and Computation 223 (2012)
19. Luo, Z.: Adjectives and adverbs in type-theoretical semantics. Notes (2011)
20. Martin-Löf, P.: An intuitionistic theory of types: predicative part. In: Rose, H., Shepherdson, J.C. (eds.) Logic Colloquium'73 (1975)
21. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)
22. Montague, R.: The proper treatment of quantification in ordinary English. In: Hintikka, J., Moravcsik, J., Suppes, P. (eds.) Approaches to Natural Languages (1973)
23. Montague, R.: Formal Philosophy. Yale University Press (1974)
24. Partee, B.: Compositionality and coercion in semantics: The semantics of adjective meaning. In: Bouma, G., Krämer, I., Zwarts, J. (eds.) Cognitive Foundations of Interpretation, Royal Netherlands Academy of Arts and Sciences (2007)
25. Partee, B.: Presuppositions and discourse: Essays offered to hans kamp. In: Bauerle, R., Reyle, U. (eds.) Privative Adjectives: Subsective Plus Coercion, pp. 123–155. Emerald Group Publishing (2010)
26. Ranta, A.: Constructing possible worlds. Theoria 52(1-2), 77–99 (1991)
27. Ranta, A.: Type-Theoretical Grammar. Oxford University Press (1994)
28. Sundholm, G.: Proof theory and meaning. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic III: Alternatives to Classical Logic. Reidel (1986)

Tree Wrapping for Role and Reference Grammar

Laura Kallmeyer, Rainer Osswald, and Robert D. Van Valin, Jr.

Sonderforschungsbereich 991*
Heinrich-Heine-Universität Düsseldorf, Germany
{kallmeyer,osswald,vanvalin}@phil.uni-duesseldorf.de

Abstract. We present a tree rewriting system that aims at formalizing the composition of syntactic templates in Role and Reference Grammar, a linguistic grammar developed mainly for typological analysis. Building on ideas from Tree Adjoining Grammar, we devise two basic operations for syntactic composition: *(wrapping) substitution* and *sister adjunction*. The first operation models plain argument insertion as well as the construction of long distance dependencies. The second operation implements adjunction to non-binary trees. We complement the definition of this tree rewriting system, called *Tree Wrapping Grammar*, by giving a CYK parser for grammars of this type.

1 Introduction

The approach to tree construction and parsing presented in this paper is part of a larger project that aims at a full formalization of Role and Reference Grammar (RRG). RRG is a theory of the grammar of natural language which has been developed as a descriptive tool for the analysis of typologically distinct languages, and which takes into account the interaction between syntax, semantics, and pragmatics [1–3]. The focus of the present paper is on formalizing the syntactic templates proposed in RRG and the compositional mechanisms operating on them. Building on ideas from Tree Adjoining Grammar (TAG), we present a tree rewriting system, called Tree Wrapping Grammar, that captures the basic tree composition principles of RRG. One of the advantages of such a formalization is that it paves the way for a computational treatment of the grammar. We define a CYK parsing schema for Tree Wrapping Grammars, which can be employed for RRG parsing.

Role and Reference Grammar. RRG is inspired by both typological and theoretical concerns. It emphasizes the importance of taking into account typologically diverse languages in the formulation of a linguistic theory, and it is a theory in which semantics and pragmatics play significant roles. In contrast to, *e.g.*, minimalist grammar [4], in RRG there is a direct mapping between the semantic and syntactic representations of a sentence, unmediated by any kind of abstract syntactic representations. *I.e.*, there is only a single syntactic representation for a sentence that corresponds to its actual form.

One of the basic assumptions of RRG is that clauses have a *layered structure* which reflects the distinction between predicates, arguments, and non-arguments [3]. The *core*

* This work was partly funded by the Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center (SFB) 991.

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 175–190, 2013.

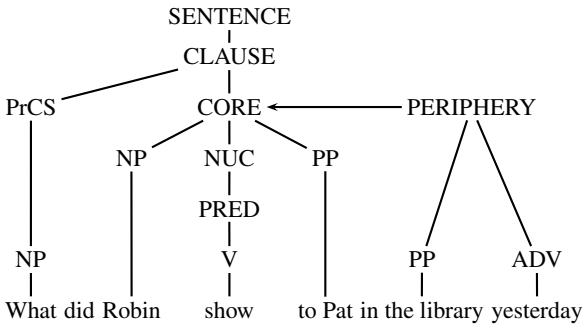


Fig. 1. RRG layered structure of an English clause

layer consists of the *nucleus*, which specifies the (verbal) predicate, and the syntactic arguments. The *clause* layer contains the core as well as extracted arguments. Each of the layers can have a *periphery* where adjuncts are attached to. As an example consider the structure of *What did Robin show to Pat in the library yesterday?* in Fig. 1.

Tree Adjoining Grammar. Tree Adjoining Grammar (TAG; [5–7]) is a grammar *formalism*, which is, *per se*, neutral with respect to its application domain, be it natural language, genetic sequences, or any other string domain. In a TAG, so-called *elementary trees* are combined into larger trees by two rewriting operations, *substitution* and *adjunction*. Fig. 2 gives a simple linguistic example: The elementary trees anchored by *John*, *laughs* and *sometimes* are used to compose the derived tree on the right. Substitution consists of replacing a non-terminal leaf with an elementary tree (see the combination of *John* and *laughs*) while adjunction consists of replacing an internal node with a new tree (see the combination of *sometimes* and *laughs*).

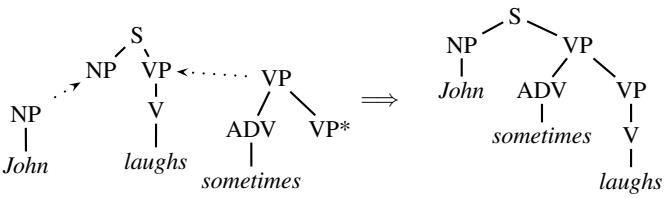


Fig. 2. Elementary TAG trees and derived tree for *John sometimes laughs*

It is a common assumption in applications of TAG to natural language that all grammatical dependencies imposed by a lexical item are represented "locally" within the elementary tree selected by the item [8]. The domain of locality with respect to dependency is thus larger than in grammars based on context-free rules. This *extended domain of locality* is one of the defining features of *Lexicalized* Tree Adjoining Grammar (LTAG) [6], where every elementary tree includes a terminal frontier node called the *lexical anchor*. The elementary tree associated with a lexical predicate contains non-terminal leaves ("argument slots") exactly for all its syntactic arguments.

Toward a Tree Formalism for RRG. The organization of the syntactic inventory of RRG has obvious connections to LTAG since the concepts of core and clause are closely related to the idea of the locality of syntactic dependencies within LTAG elementary trees. The core/clause part is non-recursive, localizes predicate-argument dependencies, and contains larger structures than the small structures corresponding to traditional phrase structure rules. In other words, the basic clause structures of RRG constitute extended domains of locality.

Viewed from this perspective, a formalization of the grammatical structures of RRG consists of two parts: the specification of the elementary trees and the definition of the compositional operations acting on them. Our general proposal is to specify, as in LTAG, the elementary trees (or, rather, tree templates) by means of a *metagrammar* [9], that is, to define the trees as minimal models of a set of appropriate tree constraints. It is thus part of the metagrammar to capture the linguistic facts and generalizations expressed within and across the elementary trees.

The present paper focusses on formalizing the *composition* of elementary trees, that is, we take them as given and put aside the issue of their metagrammatical specification. The way in which trees are combined in RRG differs from the standard tree operations provided by LTAG in several respects. Section 2 gives an overview of the different tree composition mechanisms used in RRG. In Section 3, we give a formal definition of the operations on trees as part of a tree rewriting formalism called *Tree Wrapping Grammar* (TWG), and Section 4 presents then a parser for TWG.

2 The Composition of Elementary Templates in RRG

In this section, we informally characterize the composition of RRG's elementary syntactic templates. We will go through a series of examples which are adapted from [3].

Subordination. Let us start with the simple example (1) of nominal complements in base position. As in LTAG, they can be selected for by corresponding substitution nodes that have to be filled by NP trees (see the elementary template for *carry* in Fig. 3).

(1) He carried her.

In contrast to LTAG, we do not assume a fully lexicalized grammar. Templates containing a predicate, such as the templates for the CORE structure of the verb *carry* are lexicalized, the lexical anchor being the predicate. In the template, we mark the anchor node with a diamond, following LTAG conventions. Besides this, there are also general templates without anchor nodes describing for instance the structure of a sentence.

Clausal Complements. English clausal complements in object position are, in contrast to NP arguments, not embedded under the CORE node of the verb but under its CLAUSE node.[1] Complementizers are added by an operation similar to adjunction (see below). Fig. 4 shows how the templates for the verbs in (2) and the clausal template combine.

[1] Cf. [3, pp. 199f] for a motivation of this assumption. One of the crucial observations is that peripheral elements such as temporal adverbials can occur immediately after the verb in (2) but not in (1).

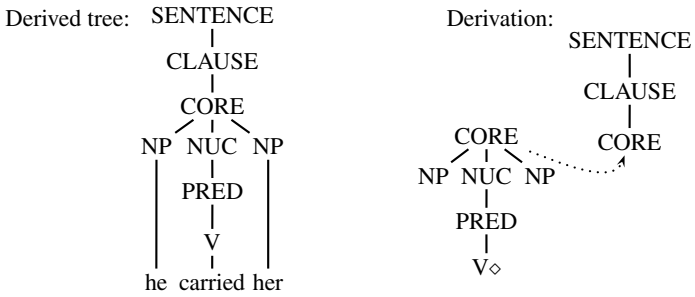


Fig. 3. Nominal complements in base position in RRG

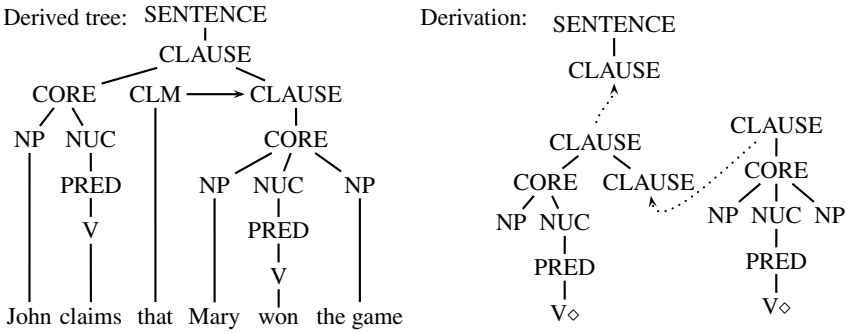


Fig. 4. RRG structure of object clausal complements

(2) John claims that Mary won the game.

The tree templates in Fig. 4 show a fundamental difference between LTAG and our formalization of RRG. The standard proposal of LTAG is to treat clausal complements not by substitution but by adjunction, in order to allow extraction out of them. This is, the tree of the matrix verb adjoins to the complement tree. In RRG, by comparison, clausal complements are realized by substitution. RRG thus allows us to define a derivation structure that is closer to a dependency structure than the LTAG derivation trees are. (The precise definition of the derivation structure is left to future research.)

(3) What$_i$ does John claim Mary loves t_i

For long-distance dependencies as in (3) we have (as in LTAG) a more complex clausal complement and the substitution is a *wrapping substitution*. This means that a subtree of the complement is inserted in a substitution node while the higher part of the complement attaches above the root of the target tree. This operation is similar to the idea of flexible composition from [10] that allows one to interpret LTAG adjunction as a wrapping operation, such that the complement is actually added to its predicate. The derivation for (3) is sketched in Fig. 5. In the complement clause, one of the nominal

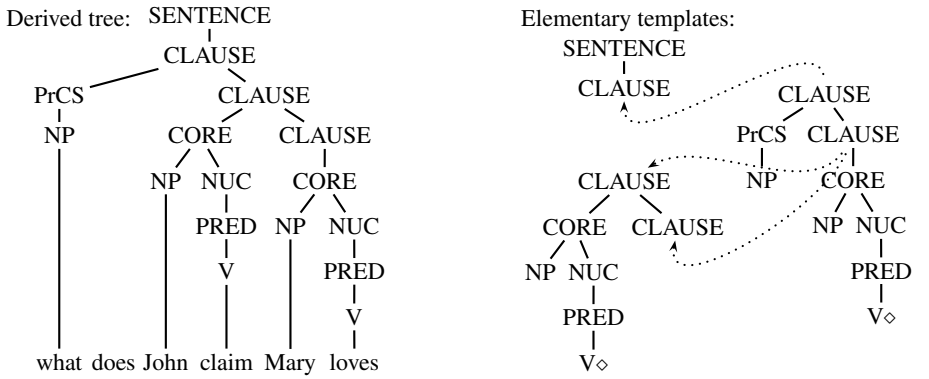


Fig. 5. Extraction from complement clauses in RRG

arguments is in pre-clause position. The complement clause wraps around the matrix clause, which yields the desired extraction tree.[2]

With this analysis, long-distance dependencies such as (4) can also be treated.

(4) Whom$_i$ does Mary think Bill claims Susan likes t_i.

The derivation of (4) is sketched in Fig. 6. Note that, crucially, we have to first substitute the *claim* tree into the *think* tree and only then, we can wrap the *likes* tree around the resulting derived tree. Otherwise, the upper part of the *likes* tree would end up between the *think* and the *claim* tree. We can also think of this as a delay in the sense of MCTAG with delayed tree-locality as defined in [11]: When substituting the lower part of the complement clause into the clausal substitution node, we need not immediately wrap the upper part around the target tree. Instead, we can delay this part of the wrapping substitution, obtaining thereby a long-distance dependency.

The RRG analysis of long-distance dependencies described here is actually less restricted than in LTAG where the analysis starts with the *likes* tree and the *claims* and *think* trees are subsequently adjoined to the lower CLAUSE node (which would be an S node in LTAG). The choice of trees that can be adjoined here depends in LTAG on the information on this CLAUSE node, i.e., depends on the *likes* tree. With our RRG analysis, depending on how long we wait until finally adding the *likes* tree by wrapping substitution, the extracted *wh* element can be arbitrarily high. In order to restrict this, RRG assumes that on the path from the lower part of the *likes* tree to the extracted elements, only CORE and CLAUSE nodes can occur. In the following, as a first formulation of this subjacency prediction, we assume that, in general, in every wrapping

[2] Note that an advantage of this analysis is that the tree of the bridge verb *claim* does not encode the fact that extraction out of the complement clause is possible (as is the case in TAG where the corresponding argument node is a foot node). As a consequence, we can use the same wrapping mechanism to extract parts of other arguments, as for instance in (i) where parts of the object NP have been extracted.

(i) which painting$_i$ did John own [$_{NP}$a copy of t_i]

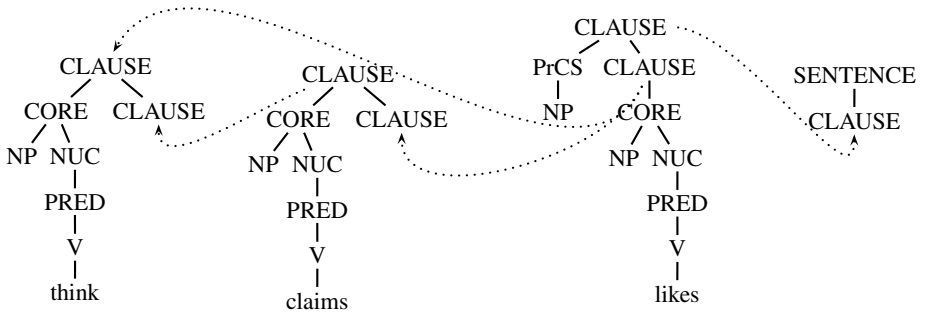


Fig. 6. Long-distance dependencies, analysis of (4)

substitution, all nodes between the substitution site and the root of the target tree must be labeled either CLAUSE or CORE. This mechanism of formulating island constraints is comparable to the integrity constraints introduced in V-TAG [12] or the path constraints from D-Tree Substitution Grammars [13].

A second crucial difference to LTAG is that, in principle, with wrapping substitution, we could have extractions out of more than one complement.[3] This is not the case in LTAG since there is at most one foot node per elementary tree. Obviously, the number of parallel wrapping substitutions directly influences parsing complexity. We assume in the following that there is some limit to the number of wrapping substitutions active at a time. Our first hypothesis is actually that this limit is 1. In other words, during derivation, for each node, the number of wrapping substitutions where this node dominates the substitution node that gets filled, is limited to 1.

The idea of wrapping substitution, namely that a lower part of a tree is added by substitution while the upper part ends up higher in the derived tree, is also present in the *subsertion* operation of D-Tree Grammar (DTG) [15] and in the *generalized substitution* from D-Tree Substitution Grammar (DSG) [13]. These operations are, however, less restricted. There can be more than one higher subtree, all of them related by dominance links, and these subtrees may not only be wrapped around the target tree but they can also be inserted into dominance links inside the target tree. In this sense wrapping substitution is a special case of the operations from DTG and DSG. Because of this freedom concerning the order of the higher subtrees, DTG and DSG can probably better deal with free word order phenomena. On the other hand, complex fixed word orders such as cross-serial dependencies are probably easier to deal with in our RRG

[3] More generally, more than one complement could be discontinuous, as in the German example (i), adapted from [14].

(i) Bücher hat derjenige Student drei gekauft der am meisten Geld hatte
books has that student three bought who the most money had
'the student with the most money bought three books'

(i) combines a split quantifier with an NP having an extraposed relative clause. In the RRG analysis, one could wrap the *Bücher ... drei* and *derjenige Student ... der am meisten Geld hatte* trees both around the *bought* tree.

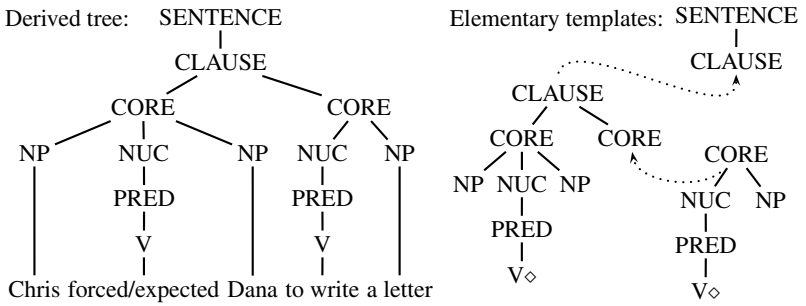


Fig. 7. RRG structure of control and raising constructions

formalization. [13] conjecture that DSG cannot generate the copy language, which exhibits a cross-serial dependency pattern, while our formal grammar framework allows to generate this language (see Fig. 11 on p. 186).

Control and Raising. RRG treats control and raising cases such as (5-a) and (5-b) in parallel, in contrast to LTAG. In both cases, the NP (here *Dana*) fills the object position of the CORE headed by *force* or *expect*; see Fig. 7.[4] The infinitive tree has category CORE and is positioned under the CLAUSE node of the matrix verb.

(5) a. Chris forced Dana to write a letter.
b. Chris expected Dana to write a letter.

(6) What$_i$ did Dana force/expect John to write t_i

For (6), we assume elementary templates as for extraction out of clausal complements (see Fig. 5), except for a label CORE instead of CLAUSE for the substitution node. However, we have then the further problem that the lower part of the *write* tree substitutes into a CORE node while the higher part wraps around a CLAUSE node. In order to deal with this, we assume that nodes where a split can occur have (possibly different) top and bottom categories (see Fig. 8). We notate them separated by a horizontal double line. In a wrapping substitution, the bottom category must equal the one at the substitution site while the top must equal the root node category of the target tree.

Periphery and Functional Elements. Modifiers such as *deliberately* and *earlier than usual* in (7) and functional elments such as complementizers, auxiliaries etc. do not constitute arguments and are therefore not added by substitution. RRG assumes a flat structure concerning the placement of these elements, i.e., they are all added as new sisters of already existing nodes.

(7) Mary deliberately left the party earlier than usual.

In RRG, some categories allow for modification and their modifiers are taken to be their *periphery*. Periphery elements constitute new daughters of the modified category.

[4] In LTAG, in (5-a), *Dana* is substituted into the *forced* tree while in (5-b), *Dana* is substituted into the *write* tree.

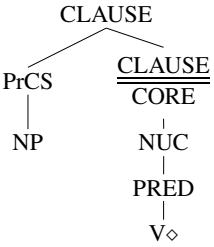


Fig. 8. RRG elementary template for extraction out of infinitives

We therefore assume that they anchor special periphery trees. These trees are such that the root category determines the category that can be modified by this tree. The root has only a single daughter which, when adjoining this tree, is added as a new daughter to the modified category. In other words, modifiers are adjoined by an operation close to the sister adjunction sometimes used in LTAG that goes back to the idea that it should be possible that several modifiers adjoin to the same target node [16] which is not allowed in standard LTAG. *Sister adjunction* was defined in [15]; it allows to add a new leftmost or rightmost subtree below the node that is the adjunction site.

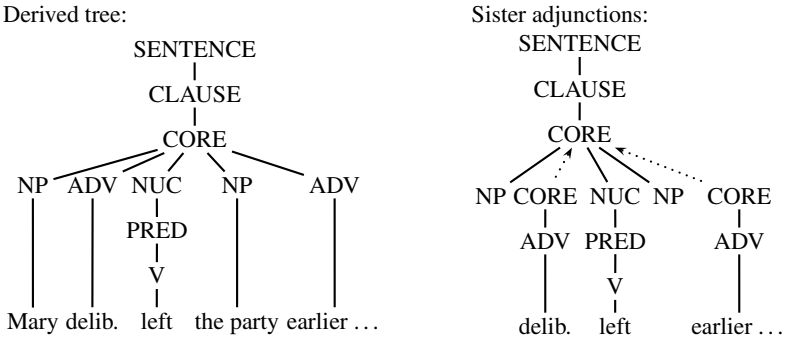


Fig. 9. Sister adjunction of periphery elements in RRG

Periphery elements can add a new daughter to a node at any position in between its daughters. Depending on the language, the positions of modifiers might be more restricted. But in general, and in particular for English, we have to assume that periphery elements can be inserted at any position below the modified category.

Functional elements are also added by sister adjunction. But they cannot be added at any position between the daughters of the node they target. Therefore, in our formalization, we assume that some adjunct trees are restricted to adding always a rightmost (resp. always a leftmost) daughter.

The fact that the adjunction of functional elements can be very restricted (for instance, at most one complementizer can be added) is left aside in this paper. In order to decide on the exact formulation of these constraints, we have to inspect a larger range

of examples. One way to implement constraints would be to use features, similarly to the ones in LTAG, to prevent ungrammatical adjunctions.

Another reason for using features could be that they provide an elegant way to percolate syntactic information and to express certain requirement as, for instance, the equality of the subject and the verb agreement features in a finite clause. We leave this to future research. The formalization given in the next section and the parsing algorithm presented here can be extended to a feature structure based version, if needed.

3 Formalization of the Syntactic Inventory

We now give a formal definition of the elementary trees and the composition operations *(wrapping) substitution* and *sister adjunction* that operate on them. We call the resulting grammar formalism a *Tree Wrapping Grammar*. The syntactic inventory of RRG can thus be characterized by a formal grammar of this type.

Tree Wrapping Grammar. An *ordered finite labeled tree* $\langle V, E, \prec, r, l\rangle$ consists of a set V of *nodes*, an *immediate dominance* relation E on V, an *immediate linear precedence* relation $\prec$ defined on sister nodes, a *root* node r, and a *labeling function* l. Two nodes are *sisters* if they are daughters of the same mother. A node $v' \in V$ is a *daughter* of $v \in V$ iff $\langle v, v'\rangle \in E$, in which case v is called the *mother* of v'. A *leaf* is a node without daughters. Let N and T be disjoint alphabets of non-terminal and terminal symbols. A *syntactic tree* is an ordered labeled tree such that $l(v) \in N \cup N^2$ for each non-leaf v, where at most one node has a label from N^2, and $l(v) \in N \cup T$ if v is a leaf. A non-terminal leaf is called a *substitution node*.[5] The labels from N^2 represent the split categories motivated in the previous section. Whether a syntactic tree is used for a standard substitution or a wrapping substitution depends on whether it has an internal node with a split category. Following the LTAG terminology, trees that can be added by substitution are called *initial* trees. In addition, we need *adjunct trees* for modeling modifiers and functional elements in RRG. These trees are added by sister adjunction. We distinguish **l**eft-adjoining, **r**ight-adjoining and unrestricted adjunct trees, resp. called *l-adjunct*, *r-adjunct* and *d-adjunct* trees. (The latter can add a **d**aughter at any position.)

Definition 1 (Tree Wrapping Grammar). *A* Tree Wrapping Grammar *(TWG) is a tuple* $G = \langle N, T, I, A, A_L, A_R, C\rangle$ *where*

- N, T *are disjoint alphabets of non-terminal and terminal symbols,*
- I, A, A_L *and* A_R *are disjoint finite sets of syntactic trees, and the root of each tree in* $A \cup A_L \cup A_R$ *has exactly one daughter.*
- $C \subseteq N$ *is the set of non-terminals that can occur on a* wrapping spine *(i.e., between root and substitution site of the target tree of a wrapping substitution).*

Every tree in I *is called an* initial *tree, every tree in* $A \cup A_L \cup A_R$ *an* adjunct *tree and every tree in* $I \cup A \cup A_L \cup A_R$ *an* elementary tree.

[5] Note that, in contrast to TAG, there is no need for auxiliary trees, i.e., there are no foot nodes.

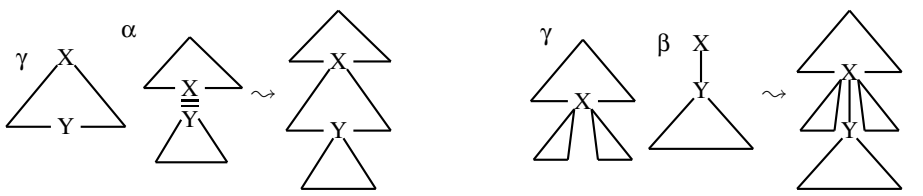


Fig. 10. Operations in TWG

Syntactic Composition. There are two composition operations on the elementary and derived trees of a TWG (see Fig. 10):

- *Standard/Wrapping substitution*: a substitution node v in a tree γ gets replaced with a subtree α' of an initial tree α. If α' is not the entire tree α, then the root node v' of α' must be labeled with a split category $\langle X,Y\rangle$ such that the root of γ is labeled X and the substitution node v is labeled Y. In this case, α is split at v' and wraps around γ, i.e., the upper part of α ends up above the root of γ while α' fills the substitution slot. Otherwise ($\alpha = \alpha'$), the root of α (i.e., v') must have the same label as the substitution node that gets replaced with α. We notate such a substitution $\gamma[v,\alpha,v']$.
- *Sister adjunction*: an adjunct tree β with root category X is added to a node v of γ labeled with category X. As a result, the root of β is identified with v and the (unique) daughter of β's root node is added as a new ith daughter to v. Furthermore, if $\beta \in A_L$ (resp. $\beta \in A_R$), then the new daughter must be a leftmost (resp. rightmost) daughter. We notate the resulting tree $\gamma[v,\beta,i]$.

Definition 2 (Standard/Wrapping substitution). *Let $\gamma = \langle V,E,\prec,r,l\rangle$ be a syntactic tree and $\alpha = \langle V',E',\prec',r',l'\rangle$ an initial tree with $V \cap V' = \emptyset$, and let v be a substitution node of γ and v' a node of α. If either a) $v' = r'$ and $l(v') = l(v)$ or b) $v' \neq r'$ and $l(v') = \langle l(r),l(v)\rangle$, then $\gamma[v,\alpha,v'] = \langle V'',E'',\prec'',r'',l''\rangle$, the result of* substituting α *with respect to v' into γ at node v is defined as follows:*

(1) $V'' = V \cup V' \setminus \{v'\}$,

(2) $E'' = E \cup (E' \setminus \{\langle v_1,v_2\rangle \mid v_1 = v' \text{ or } v_2 = v'\})$
$\cup \{\langle v_1,r\rangle \mid \langle v_1,v'\rangle \in E'\} \cup \{\langle v,v_2\rangle \mid \langle v',v_2\rangle \in E'\}$

(3) $\prec'' = \prec \cup (\prec' \setminus \{\langle v_1,v_2\rangle \mid v_1 = v' \text{ or } v_2 = v'\})$
$\cup \{\langle v_1,r\rangle \mid \langle v_1,v'\rangle \in \prec'\} \cup \{\langle r,v_2\rangle \mid \langle v',v_2\rangle \in \prec'\}$

(4) If $v' = r'$, then $r'' = r$, otherwise ($v' \neq r'$), $r'' = r'$.

(5) $l''(x) = l(x)$ *for all* $x \in V$ *and* $l''(x) = l'(x)$ *for all* $x \in V' \setminus \{v'\}$.

Otherwise, $\gamma[v,\alpha,v']$ is undefined.

In a wrapping substitution, the subtree of α rooted by v' is called the *substituting* tree, α without the nodes strictly dominated by v' is called the *wrapping* tree, and the nodes dominating the substitution node in γ are called the *wrapping spine*.

Definition 3 (Sister adjunction). *Let $\gamma = \langle V,E,\prec,r,l\rangle$ be a syntactic tree and $\beta = \langle V',E',\prec',r',l'\rangle$ an adjunct tree, and let $v \in V$. If a) $l(v) = l(r')$, b) v has at least $i-1$ daughters and c) it is not the case that $v = r$ and γ is an adjunct tree, then $\gamma[v,\beta,i] = \langle V'',E'',\prec'',r'',l''\rangle$, the result of* sister adjoining β *into γ at node v as its i-th daughter is defined as follows:*

(1) $V'' = V \cup V' \setminus \{r'\}$ *and* $r'' = r$,

(2) $E'' = E \cup (E' \setminus \{\langle v_1,v_2\rangle \mid v_1 = r'\}) \cup \{\langle v,v_1\rangle \mid \langle r',v_1\rangle \in E'\}$

(3) $\prec'' = (\prec \setminus \{\langle v_1,v_2\rangle \mid v_1$ *is the* $(i-1)$*th daughter of* v *in* $\gamma\}) \cup \prec'$
$\cup \{\langle v_1,v_2\rangle \mid v_1$ *is the* $(i-1)$*th daughter of* v *in* γ *and* $\langle r',v_2\rangle \in E'\}$
$\cup \{\langle v_1,v_2\rangle \mid v_2$ *is the* (i)*th daughter of* v *in* γ *and* $\langle r',v_1\rangle \in E'\}$

(4) $l''(x) = l(x)$ *for all* $x \in V$ *and* $l''(x) = l'(x)$ *for all* $x \in V' \setminus \{r'\}$.

Otherwise, $\gamma[v,\beta,i]$ is not defined.

Tree Language. In the following, we call a tree an instance of a tree γ if it is isomorphic to γ while preserving the labeling. TWG derived trees have as a further component a function *ws* that assigns to every node a wrapping substitution count that specifies the number of wrapping spines that this node belongs to.

Definition 4 (Derived trees). *Let $G = \langle N,T,I,A,A_L,A_R,C\rangle$ be an TWG.*

1. *For every instance $\gamma = \langle V,E,\prec,r,l\rangle$ of a $\gamma_e \in I$ (resp. $\in A$, $\in A_L$ or $\in A_R$), the tuple $\langle V,E,\prec,r,l,ws\rangle$ with $ws(v) = 0$ for all $v \in V$ is a derived initial (resp. d-adjunct or l-adjunct or r-adjunct) tree in G.*
2. *For every derived initial (resp. d/l/r-adjunct) tree $\gamma = \langle V,E,\prec,r,l,ws\rangle$ with $v \in V$ and every derived initial tree $\gamma' = \langle V',E',\prec',r',l',ws'\rangle$ with $v' \in V'$:*
   If $\gamma'' = \gamma[v,\gamma',v'] = \langle V'',E'',\prec'',r'',l''\rangle$ is defined
   a) *and $v' = r'$, then $\langle V'',E'',\prec'',r'',l'',ws''\rangle$ is a derived initial (resp. d/l/r-adjunct) tree with $ws''(x) = ws(x)$ for all $x \in V$ and $ws''(x) = ws'(x)$ for all $x \in V' \setminus \{v'\}$.*
   b) *and $v' \neq r'$, γ is an initial tree, and for all $v'' \in V$ with $\langle r,v''\rangle, \langle v'',v\rangle \in E^+$ or $r = v''$ (i.e., v'' on the wrapping spine), v'' is a node coming from an instance of an elementary initial tree, $l(v'') \in C$ and $ws(v'') = 0$, then $\langle V'',E'',\prec'',r'',l'',ws''\rangle$is a derived initial tree with $ws''(x) = ws(x) + 1$ for all $x \in V$ with $\langle r,x\rangle, \langle x,v\rangle \in E^*$ and for all other $x \in V''$, we set $ws''(x) = ws(x)$ if $x \in V$ and $ws''(x) = ws'(x)$ if $x \in V'$.*
3. *For every derived initial (resp. d/l/r-adjunct) tree $\gamma = \langle V,E,\prec,r,l,ws\rangle$ with $v \in V$ and every derived adjunct tree $\gamma' = \langle V',E',\prec',r',l',ws'\rangle$: If*
   a) *$\langle V,E,\prec,r,l\rangle[v,\langle V',E',\prec',r',l'\rangle,i] = \langle V'',E'',\prec'',r'',l''\rangle$ is defined for some i,*
   b) *if γ' is an l-adjunct tree, then $i = 1$, and*
   c) *if γ' is an r-adjunct tree, then $i = |\{v_2 \mid \langle v,v_2\rangle \in E\}| + 1$*
   then $\langle V'',E'',\prec'',r'',l'',ws''\rangle$ with $ws''(x) = ws(x)$ for all $x \in V$ and $ws''(x) = ws'(x)$ for all $x \in V'' \setminus V$ is a derived initial (resp. d/l/r-adjunct) tree.
4. *These are all derived trees in G.*

Wrapping substitutions require that all categories on the wrapping spine are in C. Furthermore, for every node v the count $ws(v)$ (initially 0) is incremented whenever v is on

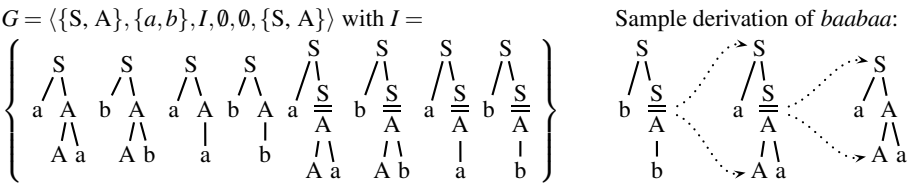


Fig. 11. TWG for the copy language $\{ww \mid w \in \{a,b\}^+\}$

the wrapping spine in a wrapping substitution. *I.e.*, $ws(v)$ tells us how many wrapping substitutions stretch across v. In our definition we assume $ws(v)$ to be limited to 1. If needed, one could later extend this to the general case of maximally k wrapping substitutions stretching across a node. A further constraint is that wrapping substitution can target only initial trees, i.e., we cannot wrap a tree around an adjunct tree.

Definition 5 (Tree language). *Let $G = \langle N, T, I, A_P, A_F, C \rangle$ be an TWG.*

1. *A* saturated *derived tree is a derived tree without substitution nodes and without nodes with labels from N^2, i.e., split categories.*
2. *The* tree language *of G is $L_T(G) = \{\gamma \mid \gamma$ is a saturated derived initial tree in $G\}$. The* string language *of G is the set of yields of trees in $L_T(G)$.*

As an example, Fig. 11 gives a TWG for the copy language. In this grammar, all substitution nodes must be filled by wrapping substitutions since there are no initial trees with root label A, and the nodes with the split categories are always the middle nodes. Along the same lines, one can analyze cross-serial dependencies, occurring in Dutch [17] or Swiss German [18], that cannot be treated with context-free grammars.

4 A CYK Parsing Scheme

So far, the only RRG parser is [19, 20]. It starts with a POS tagging and then uses the POS tags in order to determine which parts of a sentence are adjuncts, *i.e.*, modifiers. These are removed and then the remaining sentence is parsed with a bottom-up chart parser. It seems counterintuitive to use RRG-external information (the POS tags) to decide which elements are modifiers. As explained above, a crucial part of RRG is to provide a distinction between modifiers and arguments, based on a theory of linking. So the core RRG parser should make use of this instead of relying on RRG-external information in order to separate modifiers from arguments. Furthermore, a distinction of modifiers and arguments based on POS tags is not possible in general since, for instance, NPs such as *the whole day* can occur not only as arguments but also as modifiers.

In the following, we define a CYK parser for TWG (*i.e.*, a bottom-up chart parser), along the lines of the CYK parser for LTAG [21–23]. This parser can then be used to parse an RRG defined as a TWG. The parser is defined using as a deductive system over appropriate *parse items*. [24]. The items tell us about the node in an elementary tree we are in and about the span below this node. As in TAG, the span can have at most one gap, i.e., we need 4 indices to capture it. The gap contains the span of a substituting tree

$$\textbf{Scan}: \frac{}{[\gamma, p_\top, i, -, -, i+1, -, -]} \; {\scriptstyle label(\gamma,p) = w_{i+1}} \qquad \textbf{Move-up}: \frac{[\gamma, (p\cdot k)_\top, i, f_1, f_2, j, p_{ws}, X]}{[\gamma, p_\bot, i, f_1, f_2, j, p_{ws}, X]} \; \begin{array}{l}\text{node address}\\ p\cdot(k+1) \text{ does}\\ \text{not exist in } \gamma\end{array}$$

$$\textbf{Combine-sisters}: \frac{[\gamma, (p\cdot k)_\top, i_1, f_1, f_2, i_2, p_{ws}, X], [\gamma, (p\cdot(k+1))_\bot, i_2, f_1', f_2', i_3, p_{ws}', Y]}{[\gamma, (p\cdot(k+1))_\top, i_1, f_1 \oplus f_1', f_2 \oplus f_2', i_3, p_{ws} \oplus p_{ws}', X \oplus Y]}$$

$$\textbf{No-left-sister}: \frac{[\gamma, p_\bot, i, f_1, f_2, j, p_{ws}, X]}{[\gamma, p_\top, i, f_1, f_2, j, p_{ws}, X]} \; \begin{array}{l}\text{there is no left sister of } \gamma(p)\\ \text{and } \gamma(p) \text{ is not the root of a tree in } A_P \cup A_F\end{array}$$

Fig. 12. Deduction rules for movements on a single tree

in a wrapping substitution. We need to keep track of its indices since, at some point, this information needs to be combined with the wrapping tree. Besides the indices, we distinguish between position $\bot$ or $\top$ in a node. Furthermore, we need the position of a node where a wrapping substituion has started and the category of the root node of a tree targeted by a wrapping substitution. Therefore, our parse items have the form $[\gamma, p_t, i, f_1, f_2, j, p_{ws}, X]$ where

- $\gamma \in I \cup A \cup A_L \cup A_R$, p a node address[6] in γ, and $t \in \{\top, \bot\}$ a position on that node that specifies whether the tree below can be used in a substitution ($\bot$) or not ($\top$).
- $0 \le i \le f_1 \le f_2 \le j \le n$ are indices with i, j indicating the left and right boundaries of the yield of the subtree at position p and f_1, f_2 indicating the yield of a gap, if such a gap exists. We write $f_1 = f_2 = -$ if no gap is involved.
- p_{ws} is either $-$ or, if a substitution node was filled below with a wrapping substitution, the position of this substitution node.
- $X \in N \cup \{-\}$ is the root category of a wrapping substitution target tree.

We furthermore require that for every item $[\gamma, p_t, i, f_1, f_2, j, p_{ws}, X]$ with $f_1 \neq -$ and $f_2 \neq -$, $label(\gamma, p) \in C$ holds. Our **goal** items are all $[\alpha, \varepsilon_\top, 0, -, -, n, -, -]$ for $\alpha \in I$. For combining indices and categories, we use the operator $x' \oplus x'' = x$ where $x = x'$ if $x'' = -$, $x = x''$ if $x' = -$, and x is undefined otherwise.

The deduction rules are given in Fig. 12–16. **Scan** processes terminal leaves. The position is $\top$ since leaves cannot fill substitution nodes. To move in a tree, we have the following possibilities: In the bottom of a node v, we can a) either combine with the top of the left sister of v and move to the top of v (**Combine-sisters**, see Fig. 13)

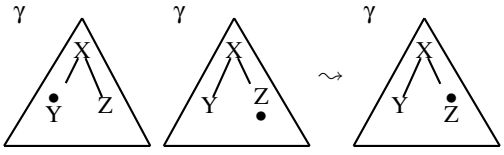


Fig. 13. Illustration of **Combine-sisters**

[6] We use Gorn addresses: ε is the address of the root and $p \cdot i$ the address of the ith daughter of the node with address p.

Predict-wrapping: $\dfrac{[\alpha, p_\perp, i, -, -, j, p_{ws}, -]}{[\gamma, p'_\perp, i, i, j, j, p', label(\gamma, \varepsilon)]}$ $label(\alpha, p)[1] = label(\gamma, p')$, $p \neq \varepsilon, \gamma(p')$ a substitution node and $\gamma \in I$

Substitute-no-gap: $\dfrac{[\alpha, \varepsilon_\perp, i, -, -, j, -, -]}{[\gamma, p'_\top, i, -, -, j, -, -]}$ $label(\alpha, \varepsilon) = label(\gamma, p')$, $\gamma(p')$ a substitution node

Substitute-with-gap: $\dfrac{[\alpha, \varepsilon_\perp, i, f_1, f_2, j, -, X]}{[\gamma, p'_\top, i, i, j, j, p', Y]}$ $label(\alpha, \varepsilon) = label(\gamma, p')$, $i \leq f_1 \leq f_2 \leq j, \gamma \in I$, $label(\gamma, \varepsilon) = Y$, $\gamma(p')$ a substitution node

Complete-wrapping 1: $\dfrac{[\gamma, \varepsilon_\top, i, f_1, f_2, j, p', X], [\alpha, p_\perp, f_1, -, -, f_2, -, -]}{[\alpha, p_\perp, i, -, -, j, -, -]}$ $label(\alpha, p)[1] = label(\gamma, p')$, $label(\alpha, p)[0] = X, p \neq \varepsilon$, $\gamma(p')$ a substitution node

Complete-wrapping 2: $\dfrac{[\gamma, \varepsilon_\top, i, f_1, f_2, j, p, X], [\alpha, \varepsilon_\perp, f_1, g_1, g_2, f_2, p_{ws}, Y]}{[\alpha, \varepsilon_\top, i, g_1, g_2, j, p_{ws}, X]}$ $label(\alpha, \varepsilon) = label(\gamma, p)$, $p \neq \varepsilon$, $\gamma(p)$ a substitution node

As a further condition, we require $\alpha \in I$ for all these rules.

Fig. 14. Deduction rules for substitution

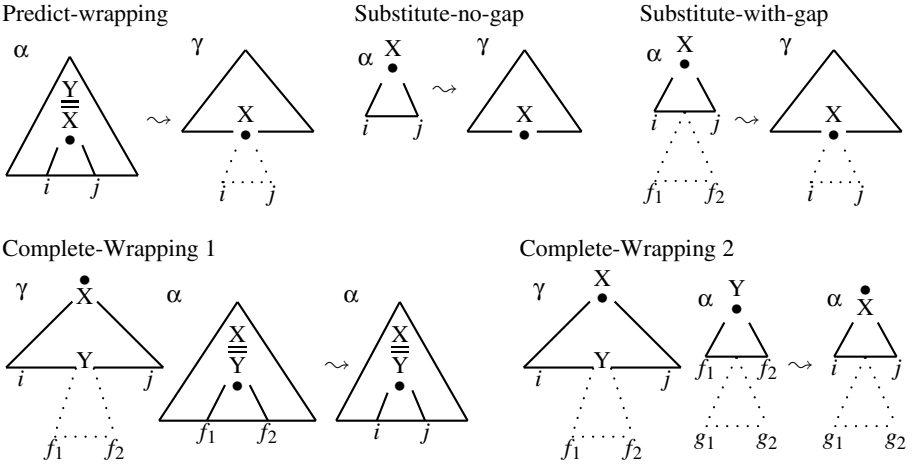


Fig. 15. Illustration of substitution operations

or b), if there is no left sister, move to the top (**No-left-sister**). In the top of a node, we can move right (**Combine-sisters**) or, if there is no right sister, move up (**Move-up**).

(Wrapping) substitutions are triggered when being in the bottom of the root of the substituting tree. Adjunctions are triggered when being in the bottom of the root of the adjunct tree and in the top of the node that will become the left/right sister of the adjoined tree. The substitution rules are given in Fig. 14 and they are illustrated in Fig. 15. Substitutions are only possible if we are in bottom position in the root of the substituting tree. If the node is not the root, the substitution is a wrapping substitution and we have to keep track of the indices of the gap (**Predict-Wrapping**). If the node is the root (address ε), it could be the case that below this node, we started a wrapping substitution that needs to be completed later. Then the item indicates a gap. If there is no gap, we can perform **Substitute-no-gap**. If there is such a gap, we have to keep track

Adjoin-right: $\dfrac{[\gamma,(p\cdot k)_\top,i_1,f_1,f_2,i_2,p_{ws},X][\beta,\varepsilon_\bot,i_2,-,-,i_3,-,-],}{[\gamma,p_\top,i_1,f_1,f_2,i_3,p_{ws},X]}$ $label(\gamma,p)=label(\beta,\varepsilon)$ and $\beta\in A$ or ($\beta\in A_R$ and $p\cdot(k+1)$ does not exist in γ)

Adjoin-left: $\dfrac{[\gamma,(p\cdot 1)_\top,i_2,f_1,f_2,i_3,p_{ws},X][\beta,\varepsilon_\bot,i_1,-,-,i_2,-,-],}{[\gamma,(p\cdot 1)_\top,i_1,f_1,f_2,i_3,p_{ws},X]}$ $label(\gamma,p)=label(\beta,\varepsilon)$ and $\beta\in A\cup A_L$

Fig. 16. Deduction rules for sister adjunction

of the indices of the substituting tree since we have to go back to it later in order to complete the wrapping substitution (**Substitute-with-gap**). Once we have finished an initial tree such that one of its leaves was filled by a wrapping substitution, we must go back to the argument tree and finish its upper part, i.e., the wrapping part (**Complete-wrapping 1**). It could also be the case that the gap did not come from a tree substituted into γ but from a tree substituted somewhere below some α that was substituted into γ. In this case, we have to go back to α, passing the information about the root category of the wrapping target tree (**Complete-wrapping 2**).

Concerning sister adjunction, in the top of a node, we can add further sisters: To any such node, we can add adjunct trees to the right (**Adjoin-right**), thereby increasing its span. Furthermore, to the leftmost daughter of a node, we can add an adjunct tree to its left (**Adjoin-left**). D-adjunct trees can be added at any position while l/r-adjunct trees are added to the left of the first and right of the last daughter, respectively.

Since the rules contain at most 6 different indices, parsing is of complexity $O(n^6)$.

5 Conclusion

This paper contributes to formalizing Role and Reference Grammar (RRG). Motivated by the fact that RRG assumes an extended domain of locality where the linking takes place, and inspired by Lexicalized Tree Adjoining Grammar (LTAG), we propose an RRG formalization along the following lines: A finite set of elementary trees is specified via a metagrammar, including the formulation of RRG's linking theory and the specification of different syntactic realizations of a given valency frame. These elementary trees are then combined into larger trees by the tree rewriting operations wrapping substitution and sister adjunction. In this paper, we have focused on the tree generation process. We have defined Tree Wrapping Grammar (TWG) as a formalization of the underlying tree rewriting grammar. TWG can trivially generate all context-free languages and, furthermore, cross-serial dependencies can be described with TWG. We have given a parsing algorithm for TWG that is of complexity $O(n^6)$.

References

1. Van Valin Jr., R.D., Foley, W.A.: Role and reference grammar. In: Moravcsik, E.A., Wirth, J.R. (eds.) Current Approaches to Syntax. Syntax and semantics, vol. 13, pp. 329–352. Academic Press, New York (1980)
2. Van Valin Jr., R.D., LaPolla, R.: Syntax: Structure, meaning and function. Cambridge University Press (1997)
3. Van Valin Jr., R.D.: Exploring the Syntax-Semantics Interface. Cambridge University Press (2005)

4. Stabler, E.P.: Derivational Minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)
5. Joshi, A.K., Levy, L.S., Takahashi, M.: Tree Adjunct Grammars. Journal of Computer and System Science 10, 136–163 (1975)
6. Joshi, A.K., Schabes, Y.: Tree-Adjoning Grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, pp. 69–123. Springer, Berlin (1997)
7. Abeillé, A., Rambow, O.: Tree Adjoining Grammar: An Overview. In: Abeillé, A., Rambow, O. (eds.) Tree Adjoining Grammars: Formalisms, Linguistic Analysis and Processing, pp. 1–68. CSLI (2000)
8. Frank, R.: Phrase Structure Composition and Syntactic Dependencies. MIT Press, Cambridge (2002)
9. Crabbé, B., Duchier, D.: Metagrammar Redux. In: International Workshop on Constraint Solving and Language Processing, Copenhagen (2004)
10. Joshi, A.K., Kallmeyer, L., Romero, M.: Flexible composition in LTAG: Quantifier scope and inverse linking. In: Bunt, H., van der Sluis, I., Morante, R. (eds.) Proceedings of the Fifth International Workshop on Computational Semantics IWCS-5, Tilburg, pp. 179–194 (2003)
11. Chiang, D., Scheffler, T.: Flexible composition and delayed tree-locality. In: TAG+9 Proceedings of the Ninth International Workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+9), Tübingen, pp. 17–24 (June 2008)
12. Rambow, O.: Formal and Computational Aspects of Natural Language Syntax. PhD thesis, University of Pennsylvania (1994)
13. Rambow, O., Vijay-Shanker, K., Weir, D.: D-Tree Substitution Grammars. Computational Linguistics (2001)
14. Chen-Main, J., Joshi, A.: A dependency perspective on the adequacy of tree local multicomponent tree adjoining grammar. Journal of Logic and Computation Advance Access (June 2012)
15. Rambow, O., Vijay-Shanker, K., Weir, D.: D-Tree Grammars. In: Proceedings of ACL (1995)
16. Schabes, Y., Shieber, S.M.: An Alternative Conception of Tree-Adjoining Derivation. Computational Linguistics 20(1), 91–124 (1994)
17. Bresnan, J., Kaplan, R.M., Peters, S., Zaenen, A.: Cross-serial dependencies in Dutch. Linguistic Inquiry 13(4), 613–635 (1982)
18. Shieber, S.M.: Evidence against the context-freeness of natural language. Linguistics and Philosophy 8, 333–343 (1985)
19. Guest, E.: Parsing for role and reference grammar. In: Van Valin Jr., R.D. (ed.) Investigations of the Syntax-Semantics-Pragmatics Interface, pp. 435–454. John Benjamins B. V., Amsterdam (2008)
20. Guest, E.: Parsing using the role and reference grammar paradigm. In: The 13th World Multi-Conference on Systemics, Cybernetics and Informatics, WMSCI 2009 (July 2009)
21. Vijay-Shanker, K., Joshi, A.K.: Some computational properties of Tree Adjoining Grammars. In: Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics, pp. 82–93 (1985)
22. Kallmeyer, L., Satta, G.: A Polynomial-Time Parsing Algorithm for TT-MCTAG. In: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, pp. 994–1002. Association for Computational Linguistics, Suntec, Singapore (2009)
23. Kallmeyer, L.: Parsing Beyond Context-Free Grammars. Cognitive Technologies. Springer, Heidelberg (2010)
24. Shieber, S.M., Schabes, Y., Pereira, F.C.N.: Principles and implementation of deductive parsing. Journal of Logic Programming 24(1+2), 3–36 (1995)

The String-Meaning Relations Definable by Lambek Grammars and Context-Free Grammars

Makoto Kanazawa[1] and Sylvain Salvati[2]

[1] National Institute of Informatics, 2–1–2 Hitotsubashi,
Chiyoda-ku, Tokyo, 101–8430, Japan
[2] INRIA Bordeaux Sud-Ouest, LaBRI, 351, cours de la Libération,
F-33405 Talence cedex, France

Abstract. We show that the class of string-meaning relations definable by the following two types of grammars coincides: (i) Lambek grammars where each lexical item is assigned a (suitably typed) lambda term as a representation of its meaning, and the meaning of a sentence is computed according to the lambda-term corresponding to its derivation; and (ii) cycle-free context-free grammars that do not generate the empty string where each rule is associated with a (suitably typed) lambda term that specifies how the meaning of a phrase is determined by the meanings of its immediate constituents.

1 Introduction

It is well known since Pentus's work [4,5,6] that Lambek grammars and context-free grammars can generate the same class of string languages (modulo the empty string). We show that the equivalence continues to hold when semantics is taken into account. Specifically, when Lambek grammars and cycle-free (i.e., finitely ambiguous) context-free grammars are enriched with Montague semantics, they define the same class of relations between (non-empty) strings and meanings (represented as typed λ-terms).

2 Preliminaries

2.1 Lambda Terms over a Higher-Order Signature

If $\mathcal{A}$ is a finite set, then the set $\mathrm{Tp}(\mathcal{A}, \to)$ of *simple types* over $\mathcal{A}$ is the smallest superset of $\mathcal{A}$ such that $A, B \in \mathrm{Tp}(\mathcal{A}, \to)$ implies $A \to B \in \mathrm{Tp}(\mathcal{A}, \to)$. A *higher-order signature* is a triple $\Sigma = (\mathcal{A}, \mathcal{C}, \tau)$, where $\mathcal{A}$ is a finite set of *atomic types*, $\mathcal{C}$ is a finite set of *constants*, and τ is a function from $\mathcal{C}$ to $\mathrm{Tp}(\mathcal{A}, \to)$. If Var is a countably infinite set of *variables*, disjoint from $\mathcal{C}$, then the set $\Lambda(\Sigma)$ of λ-*terms* over Σ is the smallest superset of $\mathcal{C} \cup \mathrm{Var}$ such that $M, N \in \Lambda(\Sigma)$ and $x \in \mathrm{Var}$ imply $MN \in \Lambda(\Sigma)$ and $\lambda x.M \in \Lambda(\Sigma)$. A *type environment* is a finite partial function from Var to $\mathrm{Tp}(\mathcal{A}, \to)$, written as a list of typing declarations $x_1 : A_1, \ldots, x_n : A_n$. A λ-term $M[x_1, \ldots, x_n]$ with free variables $x_1, \ldots, x_n$ may be assigned a type B under a typing environment $x_1 : A_1, \ldots, x_n : A_n$, or in symbols,

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 191–208, 2013.

$x_1 : A_1, \dots, x_n : A_n \vdash_\Sigma M[x_1, \dots, x_n] : B$. (The subscript Σ may be omitted when $M[x_1, \dots, x_n]$ is a *pure* λ-term, i.e., does not contain any constants.) Such a *typing judgment* is derived according to the following rules:

$$x : A \vdash_\Sigma x : A \qquad \vdash_\Sigma c : \tau(c)$$

$$\frac{\Gamma \vdash_\Sigma M : A \to B \quad \Delta \vdash_\Sigma N : A}{\Gamma \cup \Delta \vdash_\Sigma MN : B} \qquad \frac{\Gamma \vdash_\Sigma M : B}{\Gamma - \{x : A\} \vdash_\Sigma \lambda x.M : A \to B}$$

(In the last rule, x may not be in the domain of $\Gamma - \{x : A\}$.)

We assume that the reader is familiar with basic notions in λ-calculus, such as β-reduction and β-normal form. We write $M \twoheadrightarrow_\beta M'$ when M β-reduces to M', and write $|M|_\beta$ for the β-normal form of M. As is customary, we adopt the informal practice of identifying λ-terms that are identical modulo renaming of bound variables.

2.2 Product-Free Lambek Calculus

We mostly follow the notations of Pentus [5]. We let $\mathrm{Pr} = \{p_1, p_2, \dots\}$ be a countably infinite set of *primitive types*. If $\mathcal{B}$ is a subset of Pr, we let $\mathrm{Tp}(\mathcal{B}, \backslash, /)$ denote the smallest superset of $\mathcal{B}$ such that $A, B \in \mathrm{Tp}(\mathcal{B}, \backslash, /)$ implies $A\backslash B, B/A \in \mathrm{Tp}(\mathcal{B}, \backslash, /)$. Elements of $\mathrm{Tp}(\mathrm{Pr}, \backslash, /)$ are called *(directional) types*. We let p range over Pr and $A, B, C, \dots$ range over $\mathrm{Tp}(\mathrm{Pr}, \backslash, /)$. When Γ is a finite string of types, we let $|\Gamma|$ denote the number of types in Γ; thus, $|A_1 \dots A_n| = n$.

An expression of the form $\Gamma \to A$, where Γ is a non-empty finite string of types and A is a type, is called a *sequent*. The sequent calculus presentation of the *Lambek calculus* consists of the following axioms and rules:

- Axioms: $p \to p$
- Rules:

$$\frac{\Pi \to A \quad \Gamma B \Delta \to C}{\Gamma \Pi (A\backslash B) \Delta \to C}(\backslash\to) \qquad \frac{A\Pi \to B}{\Pi \to A\backslash B}(\to\backslash) \text{ where } \Pi \neq \varepsilon$$

$$\frac{\Pi \to A \quad \Gamma B \Delta \to C}{\Gamma (B/A) \Pi \Delta \to C}(/\to) \qquad \frac{\Pi A \to B}{\Pi \to B/A}(\to/) \text{ where } \Pi \neq \varepsilon$$

$$\frac{\Pi \to C \quad \Gamma C \Delta \to A}{\Gamma \Pi \Delta \to A}\mathrm{Cut}$$

A derivation is *cut-free* if it does not contain any applications of the Cut rule. It is easy to see that every sequent has only finitely many cut-free derivations.

Curry-Howard Homomorphism. Every derivation $\mathcal{D}$ is associated with a pure λ-term $h(\mathcal{D})$ according to the following rules ($x_1, x_2, \dots$ are specially reserved variables):

– If $\mathcal{D}$ is an axiom $p \to p$, then $h(\mathcal{D}) = x_1$.
– If $\mathcal{D}$ is of the form

$$\dfrac{\begin{array}{cc}\vdots\mathcal{F} & \vdots\mathcal{E}\\ \Pi \to A & \Gamma B \Delta \to C\end{array}}{\Gamma\Pi(A\backslash B)\Delta \to C}(\backslash\to)$$

then

$$h(\mathcal{D}) = M[x_1, \ldots, x_{i-1}, x_{i+n}N[x_i, \ldots, x_{i+n-1}], x_{i+n+1}, \ldots, x_{m+n}],$$

where $|\Gamma| = i - 1$, $h(\mathcal{E}) = M[x_1, \ldots, x_m]$, and $h(\mathcal{F}) = N[x_1, \ldots, x_n]$.
– If $\mathcal{D}$ is of the form

$$\dfrac{\vdots\mathcal{E}}{\dfrac{A\Pi \to B}{\Pi \to A\backslash B}(\to\backslash)}$$

then $h(\mathcal{D}) = \lambda z.M[z, x_1, \ldots, x_{m-1}]$, where $h(\mathcal{E}) = M[x_1, \ldots, x_m]$.
– If $\mathcal{D}$ ends in $(/\to)$ or $(\to/)$, $h(\mathcal{D})$ is defined similarly to the preceding two cases.
– If $\mathcal{D}$ is of the form

$$\dfrac{\begin{array}{cc}\vdots\mathcal{F} & \vdots\mathcal{E}\\ \Pi \to C & \Gamma C \Delta \to A\end{array}}{\Gamma\Pi\Delta \to A}\,\mathrm{Cut}$$

then

$$h(\mathcal{D}) = M[x_1, \ldots, x_{i-1}, N[x_i, \ldots, x_{i+n-1}], x_{i+n}, \ldots, x_{m+n-1}],$$

where $|\Gamma| = i - 1$, $h(\mathcal{E}) = M[x_1, \ldots, x_m]$, and $h(\mathcal{F}) = N[x_1, \ldots, x_n]$.

We also use h for the mapping from directional types to simple types defined by $h(p) = p$, $h(A\backslash B) = h(B/A) = h(A) \to h(B)$. If $\mathcal{D}$ is a derivation of $A_1 \ldots A_n \to B$, then we always have

$$x_1 : h(A_1), \ldots, x_n : h(A_n) \vdash h(\mathcal{D}) : h(B).$$

Another important fact is that if $\mathcal{D}$ is cut-free, then $h(\mathcal{D})$ is in β-normal form.

Cut Elimination.

$$\dfrac{p \to p \quad \begin{array}{c}\vdots\mathcal{E}\\ \Gamma p \Delta \to A\end{array}}{\Gamma p \Delta \to A}\,\mathrm{Cut} \rightsquigarrow \begin{array}{c}\vdots\mathcal{E}\\ \Gamma p \Delta \to A\end{array} \tag{C1}$$

$$\dfrac{\begin{array}{cc}\vdots\mathcal{F} & \\ \Pi \to p & p \to p\end{array}}{\Pi \to p}\,\mathrm{Cut} \rightsquigarrow \begin{array}{c}\vdots\mathcal{F}\\ \Pi \to p\end{array} \tag{C2}$$

$$\dfrac{\dfrac{\begin{array}{cc}\vdots\mathcal{F}_1 & \vdots\mathcal{F}_2\\ \Pi \to A & \Gamma B \Delta \to C\end{array}}{\Gamma\Pi(A\backslash B)\Delta \to C}(\backslash\to) \quad \begin{array}{c}\vdots\mathcal{E}\\ \Phi C \Psi \to D\end{array}}{\Phi\Gamma\Pi(A\backslash B)\Delta\Psi \to D}\,\mathrm{Cut} \rightsquigarrow \dfrac{\begin{array}{cc}\vdots\mathcal{F}_1 & \dfrac{\begin{array}{cc}\vdots\mathcal{F}_2 & \vdots\mathcal{E}\\ \Gamma B \Delta \to C & \Phi C \Psi \to D\end{array}}{\Phi\Gamma B \Delta\Psi \to D}\,\mathrm{Cut}\\ \Pi \to A & \end{array}}{\Phi\Gamma\Pi(A\backslash B)\Delta\Psi \to D}(\backslash\to) \tag{C3}$$

$$\dfrac{\dfrac{\Phi \to C \quad \dfrac{\Pi' C \Pi'' \to A \quad \Gamma B \Delta \to D}{\Gamma \Pi' C \Pi''(A\backslash B)\Delta \to D}(\backslash\to)}{\Gamma \Pi' \Phi \Pi''(A\backslash B)\Delta \to D}\mathrm{Cut}}{} \rightsquigarrow \dfrac{\dfrac{\Phi \to C \quad \Pi' C \Pi'' \to A}{\Pi' \Phi \Pi'' \to A}\mathrm{Cut} \quad \Gamma B \Delta \to D}{\Gamma \Pi' \Phi \Pi''(A\backslash B)\Delta \to D}(\backslash\to) \tag{C4}$$

$$\dfrac{\dfrac{\Phi \to C \quad \dfrac{\Pi \to A \quad \Gamma' C \Gamma'' B \Delta \to D}{\Gamma' C \Gamma'' \Pi(A\backslash B)\Delta \to D}(\backslash\to)}{\Gamma' \Phi \Gamma'' \Pi(A\backslash B)\Delta \to D}\mathrm{Cut}}{} \rightsquigarrow \dfrac{\dfrac{\Pi \to A \quad \dfrac{\Phi \to C \quad \Gamma' C \Gamma'' B \Delta \to D}{\Gamma' \Phi \Gamma'' B \Delta \to D}\mathrm{Cut}}{\Gamma' \Phi \Gamma'' \Pi(A\backslash B)\Delta \to D}(\backslash\to)}{} \tag{C5}$$

$$\dfrac{\dfrac{\Phi \to C \quad \dfrac{\Pi \to A \quad \Gamma B \Delta' C \Delta'' \to D}{\Pi(A\backslash B)\Delta' C \Delta'' \to D}(\backslash\to)}{\Gamma \Pi(A\backslash B)\Delta' \Phi \Delta'' \to D}\mathrm{Cut}}{} \rightsquigarrow \dfrac{\dfrac{\Pi \to A \quad \dfrac{\Phi \to C \quad \Gamma B \Delta' C \Delta'' \to D}{\Gamma B \Delta' \Phi \Delta'' \to D}\mathrm{Cut}}{\Gamma \Pi(A\backslash B)\Delta' \Phi \Delta'' \to D}(\backslash\to)}{} \tag{C6}$$

$$\dfrac{\Phi \to C \quad \dfrac{A\Pi' C \Pi'' \to B}{\Pi' C \Pi'' \to A\backslash B}(\to\backslash)}{\Pi' \Phi \Pi'' \to A\backslash B}\mathrm{Cut} \rightsquigarrow \dfrac{\dfrac{\Phi \to C \quad A\Pi' C \Pi'' \to B}{A\Pi' \Phi \Pi'' \to B}\mathrm{Cut}}{\Pi' \Phi \Pi'' \to A\backslash B}(\to\backslash) \tag{C7}$$

Similar to (C3)–(C7), with / in place of \. (C8)–(C12)

$$\dfrac{\dfrac{A\Phi \to B}{\Phi \to A\backslash B}(\to\backslash) \quad \dfrac{\Pi \to A \quad \Gamma B \Delta \to D}{\Gamma \Pi(A\backslash B)\Delta \to D}(\backslash\to)}{\Gamma \Pi \Phi \Delta \to D}\mathrm{Cut} \rightsquigarrow \dfrac{\dfrac{\Pi \to A \quad A\Phi \to B}{\Pi\Phi \to B}\mathrm{Cut} \quad \Gamma B \Delta \to D}{\Gamma \Pi \Phi \Delta \to D}\mathrm{Cut} \tag{C13}$$

$$\dfrac{\dfrac{A\Phi \to B}{\Phi \to A\backslash B}(\to\backslash) \quad \dfrac{\Pi \to A \quad \Gamma B \Delta \to D}{\Gamma \Pi(A\backslash B)\Delta \to D}(\backslash\to)}{\Gamma \Pi \Phi \Delta \to D}\mathrm{Cut} \rightsquigarrow \dfrac{\Pi \to A \quad \dfrac{A\Phi \to B \quad \Gamma B \Delta \to D}{\Gamma A \Phi \Delta \to D}\mathrm{Cut}}{\Gamma \Pi \Phi \Delta \to D}\mathrm{Cut} \tag{C14}$$

Similar to (C13)–(C14), with / in place of \. (C15)–(C16)

If $\mathcal{D} \rightsquigarrow \mathcal{D}'$ by one of (C1)–(C16), then $h(\mathcal{D}) \twoheadrightarrow_\beta h(\mathcal{D}')$. Every derivation $\mathcal{D}$ reduces to some cut-free derivation $\mathcal{D}'$ by repeated applications of (C1)–(C16).

In general, a derivation may reduce to many different cut-free derivations, although the β-normal λ-terms associated with these derivations are all equal.[1]

[1] The non-confluence property is due to the fact that (C3) and (C8) have overlapping domains of application with (C4)–(C7) and (C9)–(C12), and the fact that (C13) and (C14) have identical domains of application, as do (C15) and (C16). We note that (C13) and (C15) were not among the rules described by Lambek [3] in his proof of cut elimination. For our purposes, it is convenient, though not essential, to have these rewriting rules, in addition to (C14) and (C16).

2.3 Lambek Grammars with Montague Semantics

A *Lambek grammar with Montague semantics* (*Lambek grammar* for short) is a tuple $G = (\mathcal{B}, \mathcal{T}, \Sigma, f, \mathcal{R}, S)$, where

- $\mathcal{B}$ is a finite subset of Pr,
- $\mathcal{T}$ is a finite set of *terminals*,
- $\Sigma = (\mathcal{A}, \mathcal{C}, \tau)$ is a higher-order signature called the *semantic vocabulary*,
- f is a function from $\mathcal{B}$ to $\mathrm{Tp}(\mathcal{A}, \to)$,
- $\mathcal{R}$ is a finite subset of $\mathcal{T} \times \mathrm{Tp}(\mathcal{B}, \backslash, /) \times \Lambda(\Sigma)$ such that if $(a, A, M) \in \mathcal{R}$, then $\vdash_\Sigma M : f(h(A))$,[2]
- S is a distinguished element of $\mathrm{Tp}(\mathcal{B}, \backslash, /)$.

The *string-meaning relation* defined by G is

$$\mathbb{R}(G) = \{\, (a_1 \dots a_n, |M[M_1, \dots, M_n]|_\beta) \mid \mathcal{D} \text{ is a derivation of } B_1 \dots B_n \to S, M[x_1, \dots, x_n] = h(\mathcal{D}), (a_i, B_i, M_i) \in \mathcal{R} \text{ for } i = 1, \dots, n \,\}.$$

Whenever $(w, M) \in \mathbb{R}(G)$, it holds that $\vdash_\Sigma M : f(h(S))$.

2.4 Context-Free Grammars with Montague Semantics

A *context-free grammar with Montague semantics* (*context-free grammar* for short) is a tuple $G = (\mathcal{N}, \mathcal{T}, \Sigma, f, \mathcal{P}, S)$, where

- $\mathcal{N}$ is a finite set of *nonterminals*,
- $\mathcal{T}$ is a finite set of *terminals*,
- $\Sigma = (\mathcal{A}, \mathcal{C}, \tau)$ is a higher-order signature called the *semantic vocabulary*,
- f is a function from $\mathcal{N}$ to $\mathrm{Tp}(\mathcal{A}, \to)$,
- $\mathcal{P}$ is a finite set of *rules* of the form

$$B \to w_0 B_1 w_1 \dots B_n w_n : M[x_1, \dots, x_n] \tag{1}$$

  where $n \geq 0$, $B, B_1, \dots, B_n \in \mathcal{N}$, $w_0, w_1, \dots, w_n \in \mathcal{T}^*$, $M[x_1, \dots, x_n] \in \Lambda(\Sigma)$, and

$$x_1 : f(B_1), \dots, x_n : f(B_n) \vdash_\Sigma M[x_1, \dots, x_n] : f(B),$$

- S is a distinguished element of $\mathcal{N}$ called the *start symbol*.

A *derivation tree of sort* B is a tree of the form $\pi T_1 \dots T_n$, where π is a rule of the form (1) and for $i = 1, \dots, n$, T_i is a derivation tree of sort B_i. We write $\mathbb{D}(G)$ for the set of derivation trees of G (of any sort). The *string yield* of a derivation tree $T = \pi T_1 \dots T_n$ is defined recursively by

$$\mathbf{y}(T) = w_0\, \mathbf{y}(T_1)\, w_1 \ \dots\ \mathbf{y}(T_n)\, w_n.$$

[2] Here, f is homomorphically extended to a function from $\mathrm{Tp}(\mathcal{B}, \to)$ to $\mathrm{Tp}(\mathcal{A}, \to)$.

The *meaning* of T is defined by

$$\mathbf{m}(T) = M[\mathbf{m}(T_1), \ldots, \mathbf{m}(T_n)].$$

Note that whenever T is a derivation tree of sort B, we have

$$\vdash_{\Sigma} \mathbf{m}(T) : f(B).$$

We write

$$\vdash_G B(w, M)$$

to mean that there is a derivation tree T of sort B such that $\mathbf{y}(T) = w$ and $\mathbf{m}(T) = M$. The *string-meaning relation* defined by G is

$$\mathbb{R}(G) = \{\, (w, |M|_\beta) \mid \vdash_G S(w, M) \,\}.$$

In addition to the notion of a derivation tree, we need the notion of a *derivation tree context*. A derivation tree context is a derivation tree with holes, each denoted by a symbol of the form $\Box_D$, where D is a nonterminal. A *derivation tree context of sort* B is defined inductively as follows:

- $\Box_B$ is a derivation tree context of sort B.
- If π is a rule of the form (1) and T_i is a derivation tree context of sort B_i for $i = 1, \ldots, n$, then $\pi T_1 \ldots T_n$ is a derivation tree context of sort B.

The yield and meaning of a derivation tree context are defined as follows:

$$\begin{aligned}
\mathbf{y}(\Box_D) &= D, \\
\mathbf{y}(\pi T_1 \ldots T_n) &= w_0\, \mathbf{y}(T_1)\, w_1 \ \ldots\ \mathbf{y}(T_n)\, w_n, \\
\mathbf{m}(\Box_D) &= x_1, \\
\mathbf{m}(\pi T_1 \ldots T_n) &= M[P_1[x_1, \ldots, x_{k_1}], \ldots, P_n[x_{k_1+\cdots+k_{n-1}+1}, \ldots, x_{k_1+\cdots+k_n}]], \\
&\qquad \text{where } P_i[x_1, \ldots, x_{k_i}] = \mathbf{m}(T_i).
\end{aligned}$$

If T is a derivation tree context of sort B with n holes, labeled $\Box_{D_1}, \ldots, \Box_{D_n}$, respectively, from left to right, then

$$\begin{gathered}
\mathbf{y}(T) \in \mathcal{T}^* D_1 \mathcal{T}^* \ldots D_n \mathcal{T}^*, \\
x_1 : f(D_1), \ldots, x_n : f(D_n) \vdash_{\Sigma} \mathbf{m}(T) : f(B).
\end{gathered}$$

We write

$$\vdash_G B(\gamma, M)$$

to mean that there is a derivation tree context T of sort B such that $\mathbf{y}(T) = \gamma$ and $\mathbf{m}(T) = M$.

Let T be a derivation tree context of sort B with m holes, and $i \in \{1, \ldots, m\}$. If $\Box_D$ is the label of the i-th hole (from the left) of T and U is a derivation tree context of sort D with n holes, then the result of replacing the i-th hole of T

by U, call it T', is a derivation tree context of sort B with $m+n-1$ holes. If $\gamma D\delta = \mathbf{y}(T)$, where $\gamma \in (\mathcal{T}^*\mathcal{N})^{i-1}\mathcal{T}^*$, then

$$\mathbf{y}(T') = \gamma\mathbf{y}(U)\delta,$$

and

$$\mathbf{m}(T') = M[x_1, \ldots, x_{i-1}, N[x_i, \ldots, x_{i+n-1}], x_{i+n}, \ldots, x_{m+n-1}],$$

where $M[x_1, \ldots, x_m] = \mathbf{m}(T)$ and $N[x_1, \ldots, x_n] = \mathbf{m}(U)$.

If we ignore the components Σ, f of $G = (\mathcal{N}, \mathcal{T}, \Sigma, f, \mathcal{P}, S)$ and remove colons and λ-terms from the rules in $\mathcal{P}$, we get an ordinary context-free grammar. We write $\Rightarrow_G$ for the relation of one-step rewriting associated with this context-free grammar. We write $\Rightarrow_G^+$ and $\Rightarrow_G^*$ for the transitive and reflexive transitive closure of this relation, respectively. Clearly, for every $B \in \mathcal{N}$, $w \in \mathcal{T}^*$, and $\delta \in (\mathcal{N} \cup \mathcal{T})^*$, we have

- $B \Rightarrow_G^* w$ iff there is a derivation tree T of sort B such that $\mathbf{y}(T) = w$, and
- $B \Rightarrow_G^* \delta$ iff there is a derivation tree context T of sort B such that $\mathbf{y}(T) = \delta$.

We write $\mathbb{L}(G)$ for

$$\{\, w \in \mathcal{T}^* \mid S \Rightarrow_G^* w \,\} = \{\, \mathbf{y}(T) \mid T \text{ is a derivation tree of } G \text{ of sort } S \,\}.$$

We call $G = (\mathcal{N}, \mathcal{T}, \Sigma, f, \mathcal{P}, S)$ *cycle-free* if G does not allow a cycle $B \Rightarrow_G^+ B$ for any $B \in \mathcal{N}$. If G is cycle-free, then for any $w \in \mathcal{T}^*$, the set $\{\, T \in \mathbb{D}(G) \mid \mathbf{y}(T) = w \,\}$ is finite, and a fortiori, the set of meanings associated with each w, $\{\, M \mid (w, M) \in \mathbb{R}(G) \,\}$, is finite.

3 From Lambek to Context-Free Grammars

3.1 Pentus's Interpolation Lemma and Cut Elimination

Pentus's proof of his interpolation lemma for product-free Lambek calculus (Lemma 7 of [5]) amounts to an algorithm that, given a cut-free derivation $\mathcal{D}$ of $\Gamma \to C$ and a partition (Φ, Θ, Ψ) of Γ (i.e., $\Phi\Theta\Psi = \Gamma$), returns a sequence of cut-free derivations $(\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_n)$ $(n \geq 0)$ satisfying the following properties:

(i) for $i = 1, \ldots, n$, $\mathcal{D}_i$ is a derivation of $\Theta_i \to D_i$,
(ii) $\Theta_1 \ldots \Theta_n = \Theta$,
(iii) $\mathcal{D}_0$ is a derivation of $\Phi D_1 \ldots D_n \Psi \to C$,
(iv) for every atomic type p, if p occurs in D_i, then p occurs in both Θ_i and $\Phi\Psi C$.

We may add the following condition:

(v)
$$\dfrac{\dfrac{\vdots\,\mathcal{D}_n \qquad \vdots\,\mathcal{D}_0}{\dfrac{\Theta_n \to D_n \quad \Phi D_1 \ldots D_n \Psi \to C}{\Phi D_1 \ldots D_{n-1}\Theta_n\Psi \to C}\,\mathrm{Cut}} \qquad \vdots\,\mathcal{D}_1}{\dfrac{\Theta_1 \to D_1 \qquad \Phi D_1\Theta_2 \ldots \Theta_n\Psi \to C}{\Phi\Theta_1 \ldots \Theta_n\Psi \to C}\,\mathrm{Cut}} \rightsquigarrow^* \quad \vdots\,\mathcal{D}$$
$$\Phi\Theta_1 \ldots \Theta_n\Psi \to C$$

That is, the cut-free derivations found by Pentus's interpolation algorithm can be combined by the Cut rule to form a derivation that reduces to the original one.[3]

Lemma 1. *Condition* (v) *holds of Pentus's algorithm for interpolation.*

Proof (sketch). We refer to the numbering of cases used in Pentus's proof [5]. Square brackets indicate the selected (i.e., middle) part of the three-way partition of antecedents. We only treat two subcases of Case 4.

Case 4. $\mathcal{D}$ ends in an application of $(\backslash\!\rightarrow)$.

Case 4e.

$$\mathcal{D} = \dfrac{\begin{array}{cc}\vdots\,\mathcal{F} & \vdots\,\mathcal{E}\\ \Pi \rightarrow A & \Gamma'[\Gamma''B\Delta']\Delta'' \rightarrow C\end{array}}{\Gamma'[\Gamma''\Pi(A\backslash B)\Delta']\Delta'' \rightarrow C}\,(\backslash\!\rightarrow)$$

By induction hypothesis, we have

$$\dfrac{\dfrac{\begin{array}{cc}\vdots\,\mathcal{E}_r & \vdots\,\mathcal{E}_0\\ \Theta_r \rightarrow E_r & \Gamma' E_1 \dots E_r\Delta'' \rightarrow C\end{array}}{\Gamma' E_1 \dots E_{r-1}\Theta_r\Delta'' \rightarrow C}\,\mathrm{Cut} \qquad \begin{array}{cc}\vdots\,\mathcal{E}_1 & \vdots\\ \Theta_1 \rightarrow E_1 & \Gamma' E_1\Theta_2 \dots \Theta_r\Delta'' \rightarrow C\end{array}}{\Gamma'\Theta_1 \dots \Theta_r\Delta'' \rightarrow C}\,\mathrm{Cut} \rightsquigarrow^* \dfrac{\vdots\,\mathcal{E}}{\Gamma'\Theta_1 \dots \Theta_r\Delta'' \rightarrow C}$$

where $\Theta_1 \dots \Theta_r = \Gamma''B\Delta'$. Let k, Ξ, Υ be such that

$$\Theta_1 \dots \Theta_{k-1}\Xi = \Gamma'', \qquad \Theta_k = \Xi B\Upsilon, \qquad \Upsilon\Theta_{k+1} \dots \Theta_r = \Delta'.$$

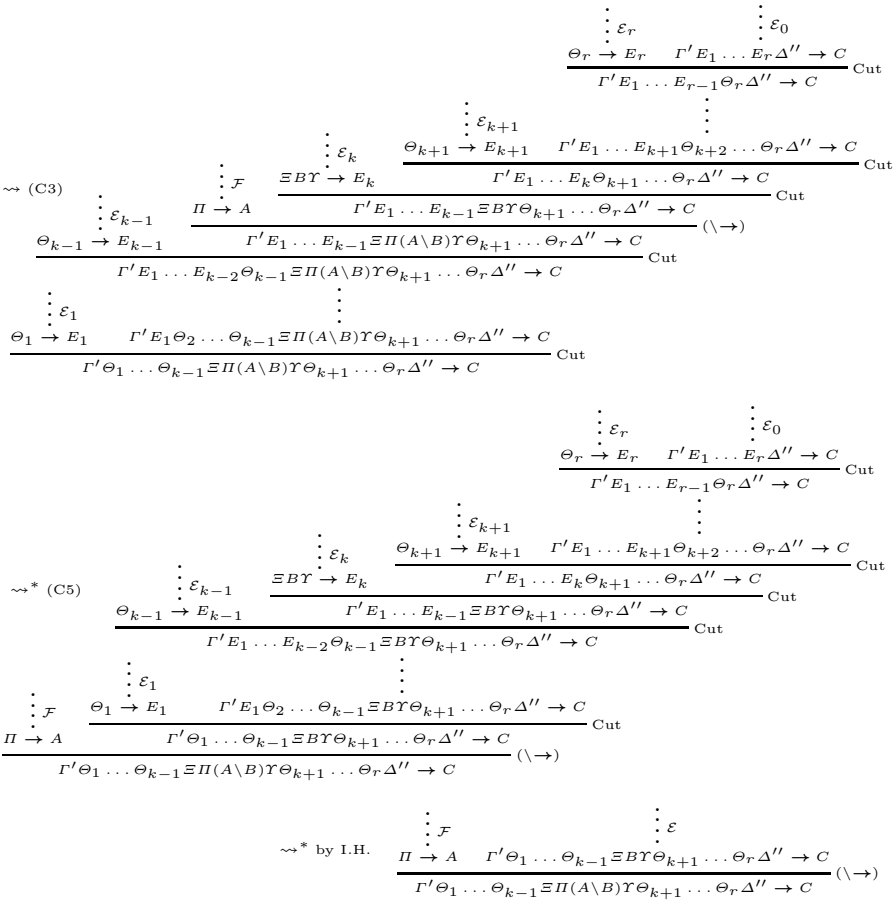
In this case, Pentus's algorithm gives $(\mathcal{E}_0, \mathcal{E}_1, \dots, \mathcal{E}_{k-1}, \tilde{\mathcal{E}}_k, \mathcal{E}_{k+1}, \dots, \mathcal{E}_r)$, where

$$\tilde{\mathcal{E}}_k = \dfrac{\begin{array}{cc}\vdots\,\mathcal{F} & \vdots\,\mathcal{E}_k\\ \Pi \rightarrow A & \Xi B\Upsilon \rightarrow E_k\end{array}}{\Xi\Pi(A\backslash B)\Upsilon \rightarrow E_k}\,(\backslash\!\rightarrow)$$

We have

$$\dfrac{\dfrac{\begin{array}{cc}\vdots\,\mathcal{E}_{k-1} & \\ \Theta_{k-1} \rightarrow E_{k-1} & \end{array} \qquad \dfrac{\begin{array}{cc}\vdots\,\mathcal{F} & \vdots\,\mathcal{E}_k\\ \Pi \rightarrow A & \Xi B\Upsilon \rightarrow E_k\end{array}}{\Xi\Pi(A\backslash B)\Upsilon \rightarrow E_k}\,(\backslash\!\rightarrow) \qquad \dfrac{\begin{array}{cc}\vdots\,\mathcal{E}_{k+1} & \\ \Theta_{k+1} \rightarrow E_{k+1} & \end{array} \qquad \begin{array}{cc}\vdots & \\ \Gamma' E_1 \dots E_{k+1}\Theta_{k+2} \dots \Theta_r\Delta'' \rightarrow C & \end{array}}{\Gamma' E_1 \dots E_k\Theta_{k+1} \dots \Theta_r\Delta'' \rightarrow C}\,\mathrm{Cut}}{\Gamma' E_1 \dots E_{k-1}\Xi\Pi(A\backslash B)\Upsilon\Theta_{k+1} \dots \Theta_r\Delta'' \rightarrow C}\,\mathrm{Cut}}{\Gamma' E_1 \dots E_{k-2}\Theta_{k-1}\Xi\Pi(A\backslash B)\Upsilon\Theta_{k+1} \dots \Theta_r\Delta'' \rightarrow C}\,\mathrm{Cut} \qquad \dfrac{\dfrac{\begin{array}{cc}\vdots\,\mathcal{E}_r & \vdots\,\mathcal{E}_0\\ \Theta_r \rightarrow E_r & \Gamma' E_1 \dots E_r\Delta'' \rightarrow C\end{array}}{\Gamma' E_1 \dots E_{r-1}\Theta_r\Delta'' \rightarrow C}\,\mathrm{Cut} \qquad \begin{array}{cc}\vdots\,\mathcal{E}_1 & \vdots\\ \Theta_1 \rightarrow E_1 & \Gamma' E_1\Theta_2 \dots \Theta_{k-1}\Xi\Pi(A\backslash B)\Upsilon\Theta_{k+1} \dots \Theta_r\Delta'' \rightarrow C\end{array}}{\Gamma'\Theta_1 \dots \Theta_{k-1}\Xi\Pi(A\backslash B)\Upsilon\Theta_{k+1} \dots \Theta_r\Delta'' \rightarrow C}\,\mathrm{Cut}$$

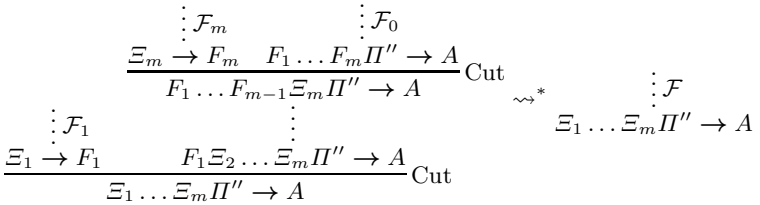
[3] For the purpose of the present paper, it is actually enough to know that the λ-terms corresponding to the two derivations in (v) are β-equal, but the stronger property may be of independent interest. For an analogous (but more involved) property of interpolation in the sequent calculus for intuitionistic implicational logic, see [1].



Case 4f.

$$\mathcal{D} = \dfrac{\begin{array}{cc}\vdots\,\mathcal{F} & \vdots\,\mathcal{E} \\ [\Pi']\Pi'' \to A & \Gamma[B\Delta']\Delta'' \to C\end{array}}{\Gamma\Pi'[\Pi''(A\backslash B)\Delta']\Delta'' \to C}(\backslash\!\to)$$

where $\Pi' \neq \varepsilon$. By induction hypothesis, we have



$$\dfrac{\dfrac{\vdots\,\mathcal{E}_1}{\Theta_1 \to E_1} \quad \dfrac{\vdots\,\mathcal{E}_r \qquad \vdots\,\mathcal{E}_0}{\dfrac{\Theta_r \to E_r \quad \Gamma E_1 \dots E_r \Delta'' \to C}{\Gamma E_1 \dots E_{r-1} \Theta_r \Delta'' \to C}\,\mathrm{Cut}}}{\Gamma\Theta_1 \dots \Theta_r \Delta'' \to C}\,\mathrm{Cut} \quad \rightsquigarrow^* \quad \dfrac{\vdots\,\mathcal{E}}{\Gamma\Theta_1 \dots \Theta_r \Delta'' \to C}$$

where $m, r \geq 1$, $\Xi_1 \dots \Xi_m = \Pi'$, and $\Theta_1 \dots \Theta_r = B\Delta'$. In this case, Pentus's algorithm gives $(\tilde{\mathcal{E}}_0, \tilde{\mathcal{E}}_1, \mathcal{E}_2, \dots, \mathcal{E}_r)$, where

$$\tilde{\mathcal{E}}_0 = \dfrac{\dfrac{\vdots\,\mathcal{F}_m}{\Xi_m \to F_m} \quad \dfrac{\dfrac{\vdots\,\mathcal{F}_1 \qquad \vdots\,\mathcal{E}_0}{\Xi_1 \to F_1 \quad \Gamma E_1 \dots E_r \Delta'' \to C}(\backslash\!\to)}{\Gamma \Xi_1 (F_1 \backslash E_1) E_2 \dots E_r \Delta'' \to C}}{\Gamma \Xi_1 \dots \Xi_m (F_m \backslash (\dots \backslash (F_1 \backslash E_1) \dots)) E_2 \dots E_r \Delta'' \to C}(\backslash\!\to)$$

$$\tilde{\mathcal{E}}_1 = \dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_0 \qquad \vdots\,\mathcal{E}_1}{F_1 \dots F_m \Pi'' \to A \quad B\Upsilon \to E_1}(\backslash\!\to)}{F_1 \dots F_m \Pi''(A\backslash B)\Upsilon \to E_1}(\to\!\backslash)}{F_2 \dots F_m \Pi''(A\backslash B)\Upsilon \to (F_1 \backslash E_1)} \quad \vdots \quad \dfrac{F_m \Pi''(A\backslash B)\Upsilon \to (F_{m-1} \backslash (\dots \backslash (F_1 \backslash E_1) \dots))}{\Pi''(A\backslash B)\Upsilon \to (F_m \backslash (\dots \backslash (F_1 \backslash E_1) \dots))}(\to\!\backslash)$$

with $\Theta_1 = B\Upsilon$ and $\Delta' = \Upsilon\Theta_2 \dots \Theta_r$. In the following derivations, we abbreviate a sequence of types $C_i \dots C_j$ by $C_{i..j}$, a concatenation of sequences of types $\Gamma_i \dots \Gamma_j$ by $\Gamma_{i..j}$, and a type of the form $(C_i \backslash (\dots \backslash (C_j \backslash D) \dots))$ by $(C_{i..j} \backslash D)$. We also omit rule labels other than "Cut". We have

$$\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_0 \qquad \vdots\,\mathcal{E}_1}{\dfrac{F_{1..m}\Pi'' \to A \quad B\Upsilon \to E_1}{F_{1..m}\Pi''(A\backslash B)\Upsilon \to E_1}}{F_{2..m}\Pi''(A\backslash B)\Upsilon \to (F_1 \backslash E_1)} \quad \dfrac{\dfrac{\vdots\,\mathcal{E}_r \qquad \dfrac{\vdots\,\mathcal{F}_m \qquad \dfrac{\vdots\,\mathcal{F}_1 \qquad \vdots\,\mathcal{E}_0}{\Xi_1 \to F_1 \quad \Gamma E_{1..r} \Delta'' \to C}}{\Xi_m \to F_m \quad \Gamma \Xi_{1..m-1}(F_{m-1..1} \backslash E_1) E_{2..r} \Delta'' \to C}}{\Theta_r \to E_r \qquad \Gamma \Xi_{1..m}(F_{m..1} \backslash E_1) E_{2..r} \Delta'' \to C}\,\mathrm{Cut}}{\Gamma \Xi_{1..m}(F_{m..1} \backslash E_1) E_{2..r-1} \Theta_r \Delta'' \to C}}{F_m \Pi''(A\backslash B)\Upsilon \to (F_{m-1..1} \backslash E_1)} \quad \dfrac{\vdots\,\mathcal{E}_2}{\Theta_2 \to E_2 \quad \Gamma \Xi_{1..m}(F_{m..1} \backslash E_1) E_2 \Theta_{3..r} \Delta'' \to C}}{\Pi''(A\backslash B)\Upsilon \to (F_{m..1} \backslash E_1)} \quad \dfrac{}{\Gamma \Xi_{1..m}(F_{m..1} \backslash E_1) E_{2..r} \Delta'' \to C}\,\mathrm{Cut}}{\Gamma \Xi_{1..m} \Pi''(A\backslash B)\Upsilon \Theta_{2..r} \Delta'' \to C}\,\mathrm{Cut}$$

$$\rightsquigarrow^{*}\ (\mathrm{C6})\quad \dfrac{\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_0}{F_{1..m}\Pi'' \to A}\quad \dfrac{\vdots\,\mathcal{E}_1}{B\Upsilon \to E_1}}{F_{1..m}\Pi''(A\backslash B)\Upsilon \to E_1}}{F_{2..m}\Pi''(A\backslash B)\Upsilon \to (F_1\backslash E_1)}}{\dfrac{\vdots}{F_m\Pi''(A\backslash B)\Upsilon \to (F_{m-1..1}\backslash E_1)}}\quad \dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_1}{\Xi_1 \to F_1}\quad \dfrac{\dfrac{\vdots\,\mathcal{E}_2}{\Theta_2 \to E_2}\quad \dfrac{\vdots\,\mathcal{E}_r}{\Theta_r \to E_r}\quad \dfrac{\vdots\,\mathcal{E}_0}{\Gamma E_{1..r}\Delta'' \to C}\,\mathrm{Cut}}{\Gamma E_{1..r-1}\Theta_r\Delta'' \to C}\,\mathrm{Cut}}{\Gamma E_1\Theta_{2..r}\Delta'' \to C}\,\mathrm{Cut}}{\Gamma\Xi_1(F_1\backslash E_1)\Theta_{2..r}\Delta'' \to C}}{\dfrac{\vdots}{\Xi_m \to F_m}\quad \Gamma\Xi_{1..m-1}(F_{m-1..1}\backslash E_1)\Theta_{2..r}\Delta'' \to C}\,\mathrm{Cut}}{\Gamma\Xi_{1..m}\Pi''(A\backslash B)\Upsilon\Theta_{2..r}\Delta'' \to C}\,\mathrm{Cut}$$

$$\rightsquigarrow\ (\mathrm{C13})\quad \dfrac{\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_0}{F_{1..m}\Pi'' \to A}\quad \dfrac{\vdots\,\mathcal{E}_1}{B\Upsilon \to E_1}}{F_{1..m}\Pi''(A\backslash B)\Upsilon \to E_1}}{F_{2..m}\Pi''(A\backslash B)\Upsilon \to (F_1\backslash E_1)}}{\dfrac{\vdots\,\mathcal{F}_m}{\Xi_m \to F_m}\quad F_{m-1}F_m\Pi''(A\backslash B)\Upsilon \to (F_{m-2..1}\backslash E_1)}\,\mathrm{Cut}}{F_m\Pi''(A\backslash B)\Upsilon \to (F_{m-1..1}\backslash E_1)}\,\mathrm{Cut}\quad \dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_1}{\Xi_1 \to F_1}\quad \dfrac{\dfrac{\vdots\,\mathcal{E}_2}{\Theta_2 \to E_2}\quad \dfrac{\vdots\,\mathcal{E}_r}{\Theta_r \to E_r}\quad \dfrac{\vdots\,\mathcal{E}_0}{\Gamma E_{1..r}\Delta'' \to C}\,\mathrm{Cut}}{\Gamma E_{1..r-1}\Theta_r\Delta'' \to C}\,\mathrm{Cut}}{\Gamma E_1\Theta_{2..r}\Delta'' \to C}\,\mathrm{Cut}}{\Gamma\Xi_1(F_1\backslash E_1)\Theta_{2..r}\Delta'' \to C}}{\dfrac{\vdots\,\mathcal{F}_{m-1}}{\Xi_{m-1} \to F_{m-1}}\quad \Gamma\Xi_{1..m-2}(F_{m-2..1}\backslash E_1)\Theta_{2..r}\Delta'' \to C}}{\Xi_m\Pi''(A\backslash B)\Upsilon \to (F_{m-1..1}\backslash E_1)}\quad \Gamma\Xi_{1..m-1}(F_{m-1..1}\backslash E_1)\Theta_{2..r}\Delta'' \to C\,\mathrm{Cut}}{\Gamma\Xi_{1..m}\Pi''(A\backslash B)\Upsilon\Theta_{2..r}\Delta'' \to C}\,\mathrm{Cut}$$

$$\rightsquigarrow^{*}\ (\mathrm{C7})\quad \dfrac{\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_0}{\vdots\,\mathcal{F}_m}\quad \dfrac{F_{1..m}\Pi'' \to A\quad \dfrac{\vdots\,\mathcal{E}_1}{B\Upsilon \to E_1}}{F_{1..m}\Pi''(A\backslash B)\Upsilon \to E_1}}{\Xi_m \to F_m}\,\mathrm{Cut}}{F_{1..m-1}\Xi_m\Pi''(A\backslash B)\Upsilon \to E_1}}{F_{2..m-1}\Xi_m\Pi''(A\backslash B)\Upsilon \to (F_1\backslash E_1)}}{\dfrac{\vdots}{F_{m-1}\Xi_m\Pi''(A\backslash B)\Upsilon \to (F_{m-2..1}\backslash E_1)}}\quad \dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_1}{\Xi_1 \to F_1}\quad \dfrac{\dfrac{\vdots\,\mathcal{E}_2}{\Theta_2 \to E_2}\quad \dfrac{\vdots\,\mathcal{E}_r}{\Theta_r \to E_r}\quad \dfrac{\vdots\,\mathcal{E}_0}{\Gamma E_{1..r}\Delta'' \to C}\,\mathrm{Cut}}{\Gamma E_{1..r-1}\Theta_r\Delta'' \to C}\,\mathrm{Cut}}{\Gamma E_1\Theta_{2..r}\Delta'' \to C}\,\mathrm{Cut}}{\Gamma\Xi_1(F_1\backslash E_1)\Theta_{2..r}\Delta'' \to C}}{\dfrac{\vdots\,\mathcal{F}_{m-1}}{\Xi_{m-1} \to F_{m-1}}\quad \Gamma\Xi_{1..m-2}(F_{m-2..1}\backslash E_1)\Theta_{2..r}\Delta'' \to C}}{\Xi_m\Pi''(A\backslash B)\Upsilon \to (F_{m-1..1}\backslash E_1)}\quad \Gamma\Xi_{1..m-1}(F_{m-1..1}\backslash E_1)\Theta_{2..r}\Delta'' \to C\,\mathrm{Cut}}{\Gamma\Xi_{1..m}\Pi''(A\backslash B)\Upsilon\Theta_{2..r}\Delta'' \to C}\,\mathrm{Cut}$$

$$\rightsquigarrow\ (\mathrm{C4})\quad \dfrac{\dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_m}{\Xi_m \to F_m}\quad \dfrac{\vdots\,\mathcal{F}_0}{F_{1..m}\Pi'' \to A}\,\mathrm{Cut}}{F_{1..m-1}\Xi_m\Pi'' \to A}\quad \dfrac{\vdots\,\mathcal{E}_1}{B\Upsilon \to E_1}}{F_{1..m-1}\Xi_m\Pi''(A\backslash B)\Upsilon \to E_1}}{F_{2..m-1}\Xi_m\Pi''(A\backslash B)\Upsilon \to (F_1\backslash E_1)}}{\dfrac{\vdots}{F_{m-1}\Xi_m\Pi''(A\backslash B)\Upsilon \to (F_{m-2..1}\backslash E_1)}}\quad \dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_1}{\Xi_1 \to F_1}\quad \dfrac{\dfrac{\vdots\,\mathcal{E}_2}{\Theta_2 \to E_2}\quad \dfrac{\vdots\,\mathcal{E}_r}{\Theta_r \to E_r}\quad \dfrac{\vdots\,\mathcal{E}_0}{\Gamma E_{1..r}\Delta'' \to C}\,\mathrm{Cut}}{\Gamma E_{1..r-1}\Theta_r\Delta'' \to C}\,\mathrm{Cut}}{\Gamma E_1\Theta_{2..r}\Delta'' \to C}\,\mathrm{Cut}}{\Gamma\Xi_1(F_1\backslash E_1)\Theta_{2..r}\Delta'' \to C}}{\dfrac{\vdots\,\mathcal{F}_{m-1}}{\Xi_{m-1} \to F_{m-1}}\quad \Gamma\Xi_{1..m-2}(F_{m-2..1}\backslash E_1)\Theta_{2..r}\Delta'' \to C}}{\Xi_m\Pi''(A\backslash B)\Upsilon \to (F_{m-1..1}\backslash E_1)}\quad \Gamma\Xi_{1..m-1}(F_{m-1..1}\backslash E_1)\Theta_{2..r}\Delta'' \to C\,\mathrm{Cut}}{\Gamma\Xi_1\ \ _m\Pi''(A\backslash B)\Upsilon\Theta_2\ \ _r\Delta'' \to C}\,\mathrm{Cut}$$

$$\rightsquigarrow^{*}\ \text{(C13), (C7), (C4)}\quad \dfrac{\dfrac{\dfrac{\vdots\,\mathcal{F}_1}{\Xi_1 \to F_1} \quad \dfrac{\dfrac{\vdots\,\mathcal{F}_m \quad \vdots\,\mathcal{F}_0}{\Xi_m \to F_m \quad F_{1..m}\Pi'' \to A}\,\text{Cut}}{F_{1..m-1}\Xi_m\Pi'' \to A}}{\Xi_{1..m}\Pi'' \to A}\,\text{Cut} \quad \dfrac{\vdots\,\mathcal{E}_1}{B\Upsilon \to E_1} \quad \dfrac{\dfrac{\vdots\,\mathcal{E}_r \quad \vdots\,\mathcal{E}_0}{\Theta_r \to E_r \quad \Gamma E_{1..r}\Delta'' \to C}\,\text{Cut}}{\Gamma E_{1..r-1}\Theta_r\Delta'' \to C} \quad \dfrac{\vdots\,\mathcal{E}_2}{\Theta_2 \to E_2} \quad \dfrac{\vdots}{\Gamma E_1 E_2\Theta_{3..r}\Delta'' \to C}}{\dfrac{\Xi_{1..m}\Pi''(A\backslash B)\Upsilon \to E_1 \qquad \Gamma E_1\Theta_{2..r}\Delta'' \to C}{\Gamma\Xi_{1..m}\Pi''(A\backslash B)\Upsilon\Theta_{2..r}\Delta'' \to C}\,\text{Cut}}\,\text{Cut}$$

$$\rightsquigarrow\ \text{(C3)}\quad \dfrac{\dfrac{\vdots\,\mathcal{F}_1}{\Xi_1 \to F_1} \quad \dfrac{\dfrac{\vdots\,\mathcal{F}_m \quad \vdots\,\mathcal{F}_0}{\Xi_m \to F_m \quad F_{1..m}\Pi'' \to A}\,\text{Cut}}{F_{1..m-1}\Xi_m\Pi'' \to A}}{\Xi_{1..m}\Pi'' \to A}\,\text{Cut} \quad \dfrac{\vdots\,\mathcal{E}_1}{B\Upsilon \to E_1} \quad \dfrac{\dfrac{\vdots\,\mathcal{E}_r \quad \vdots\,\mathcal{E}_0}{\Theta_r \to E_r \quad \Gamma E_{1..r}\Delta'' \to C}\,\text{Cut}}{\Gamma E_{1..r-1}\Theta_r\Delta'' \to C} \quad \dfrac{\vdots\,\mathcal{E}_2}{\Theta_2 \to E_2} \quad \dfrac{\vdots}{\Gamma E_1 E_2\Theta_{3..r}\Delta'' \to C}}{\dfrac{\Xi_{1..m}\Pi'' \to A \qquad \Gamma B\Upsilon\Theta_{2..r}\Delta'' \to C}{\Gamma\Xi_{1..m}\Pi''(A\backslash B)\Upsilon\Theta_{2..r}\Delta'' \to C}\,\text{Cut}}\,\text{Cut}$$

$$\rightsquigarrow^{*}\ \text{by I.H.}\quad \dfrac{\vdots\,\mathcal{F}}{\Xi_{1..m}\Pi'' \to A} \quad \dfrac{\vdots\,\mathcal{E}}{\Gamma B\Upsilon\Theta_{2..r}\Delta'' \to C}}{\Gamma\Xi_{1..m}\Pi''(A\backslash B)\Upsilon\Theta_{2..r}\Delta'' \to C}$$

The remaining cases are handled similarly. □

3.2 Pentus's Construction

Define

$$\|p\| = 1, \qquad \|A\backslash B\| = \|A\| + \|B\|, \qquad \|B/A\| = \|B\| + \|A\|,$$
$$\|A_1 \dots A_n\| = \|A_1\| + \dots + \|A_n\|.$$

An (m,q)-*type* is a type A such that $\|A\| \leq m$ and the atomic types that occur in A are among $p_1, \dots, p_q$. A sequent $A_1 \dots A_n \to C$ is an (m,q)-*sequent* if $A_1, \dots, A_n, C$ are all (m,q)-types. The class of $\mathrm{Lcut}(m,q)$-*derivations* are defined inductively as follows:

- A cut-free derivation of $A_1 \dots A_n \to C$ is an $\mathrm{Lcut}(m,q)$-derivation if $A_1 \dots A_n \to C$ is an (m,q)-sequent and $\|A_1 \dots A_n\| \leq 2m$.
- If $\mathcal{F}$ is an $\mathrm{Lcut}(m,q)$-derivation of $\Pi \to C$ and $\mathcal{E}$ is an $\mathrm{Lcut}(m,q)$-derivation of $\Gamma C\Delta \to A$, then

$$\dfrac{\dfrac{\vdots\,\mathcal{F}}{\Pi \to C} \quad \dfrac{\vdots\,\mathcal{E}}{\Gamma C\Delta \to A}}{\Gamma\Pi\Delta \to A}\,\text{Cut}$$

  is an $\mathrm{Lcut}(m,q)$-derivation.

Pentus uses his interpolation lemma to prove that every derivable (m,q)-sequent has an $\mathrm{Lcut}(m,q)$-derivation (Theorem 1 of [5]). With Lemma 1, we can strengthen this theorem to the following:

Lemma 2. *For every cut-free derivation $\mathcal{D}$ of an (m,q)-sequent, there is an* $\mathrm{Lcut}(m,q)$*-derivation $\mathcal{D}'$ of the same sequent such that $\mathcal{D}' \rightsquigarrow^* \mathcal{D}$.*

Let $G = (\mathcal{B}, \mathcal{T}, \Sigma, f, \mathcal{R}, S)$ be a Lambek grammar with Montague semantics. Let q be the least number such that $\mathcal{B} \subseteq \{p_1, \ldots, p_q\}$ and let $m = \max(\{\, \|B\| \mid (a, B, M) \in \mathcal{R} \,\} \cup \{\|S\|\})$. Construct a context-free grammar with Montague semantics $G_{\mathrm{cf}} = (\mathcal{N}, \mathcal{T}, \Sigma, f', \mathcal{P}, S)$, where

$$\begin{aligned}
\mathcal{N} &= \{\, B \in \mathrm{Tp}(\mathcal{B}, \backslash, /) \mid B \text{ is an } (m,q)\text{-type} \,\},\\
f'(B) &= f(h(B)) \quad \text{for all } B \in \mathcal{N},\\
\mathcal{P} &= \{\, C \to A_1 \ldots A_n : h(\mathcal{D}) \mid C, A_1, \ldots, A_n \in \mathcal{N}, \|A_1 \ldots A_n\| \le 2m,\\
&\qquad\qquad \mathcal{D} \text{ is a cut-free derivation of } A_1 \ldots A_n \to C \,\} \cup\\
&\quad \{\, B \to a : M \mid (a, B, M) \in \mathcal{R} \,\}.
\end{aligned}$$

Note that $\mathcal{N}$ and $\mathcal{P}$ are both finite.

Lemma 3. *Let $B_1, \ldots, B_n, C \in \mathcal{N}$.*

(i) *If $\mathcal{D}$ is an* $\mathrm{Lcut}(m,q)$*-derivation of $B_1 \ldots B_n \to C$, then there is a derivation tree context T of G_{cf} of sort C such that $\mathbf{y}(T) = B_1 \ldots B_n$ and $\mathbf{m}(T) = h(\mathcal{D})$.*

(ii) *If T is a derivation tree context of G_{cf} of sort C such that $\mathbf{y}(T) = B_1 \ldots B_n$, then there is an* $\mathrm{Lcut}(m,q)$*-derivation $\mathcal{D}$ of $B_1 \ldots B_n \to C$ such that $h(\mathcal{D}) = \mathbf{m}(T)$.*

Theorem 4. *For any Lambek grammar with Montague semantics G, $\mathbb{R}(G) = \mathbb{R}(G_{\mathrm{cf}})$.*

The grammar G_{cf} contains cycles $B \Rightarrow^+_{G_{\mathrm{cf}}} B$. The next lemma allows us to modify the construction to obtain a grammar G'_{cf} that has no rule of the form $B \to A : M[x_1]$.

Lemma 5. *For any* $\mathrm{Lcut}(m,q)$*-derivation $\mathcal{D}$, there is an* $\mathrm{Lcut}(m,q)$*-derivation $\mathcal{D}'$ of the same sequent such that $|h(\mathcal{D})|_\beta = |h(\mathcal{D}')|_\beta$ and no sequent of the form $A \to B$ appears in $\mathcal{D}'$ as a right premise of the Cut rule.*

Proof (sketch). Use the following rewriting to transform $\mathcal{D}$ into $\mathcal{D}'$.

$$\dfrac{\dfrac{\vdots\,\mathcal{F}_1 \quad \vdots\,\mathcal{F}_2}{\Pi \to C \quad \Gamma C \Delta \to A}\,\mathrm{Cut} \qquad \vdots\,\mathcal{E}}{\dfrac{\Gamma \Pi \Delta \to A \qquad A \to B}{\Gamma \Pi \Delta \to B}\,\mathrm{Cut}}\,\mathrm{Cut} \quad \dashrightarrow \quad \dfrac{\vdots\,\mathcal{F}_1 \quad \dfrac{\vdots\,\mathcal{F}_2 \qquad \vdots\,\mathcal{E}}{\Gamma C \Delta \to A \quad A \to B}\,\mathrm{Cut}}{\dfrac{\Pi \to C \qquad \Gamma C \Delta \to B}{\Gamma \Pi \Delta \to B}\,\mathrm{Cut}}\,\mathrm{Cut}$$

$$\dfrac{\vdots\,\mathcal{F} \quad \vdots\,\mathcal{E}}{\dfrac{\Gamma \to A \quad A \to B}{\Gamma \to B}\,\mathrm{Cut}}\,\mathrm{Cut} \quad \rightsquigarrow^* \text{ a cut-free derivation of } \Gamma \to B \quad \text{where } \|\Gamma\| \le 2m \qquad \square$$

4 From Context-Free to Lambek Grammars

4.1 From Greibach Normal Form Context-Free Grammars to Lambek Grammars

As with the case of context-free grammars without semantics, the conversion from context-free grammars with Montague semantics to Lambek grammars is based on the Greibach normal form. A context-free grammar with Montague semantics $G = (\mathcal{N}, \mathcal{T}, \Sigma, f, \mathcal{P}, S)$ is said to be in *Greibach normal form* if the associated grammar without semantics is in Greibach normal form, i.e., if each rule in $\mathcal{P}$ is of the form $B \to aC_1 \dots C_n : M[x_1, \dots, x_n]$, where $a \in \mathcal{T}$ and $C_i \in \mathcal{N}$. Such a grammar can be converted to a Lambek grammar $G' = (\mathcal{N}, \mathcal{T}, \Sigma, f, \mathcal{R}, S)$ by letting $\mathcal{R}$ consist of all triples

$$(a, (\dots (B/C_n)/\dots)/C_1, \lambda z_1 \dots z_n.M[z_1, \dots, z_n])$$

such that $B \to aC_1 \dots C_n : M[x_1, \dots, x_n]$ is a rule in $\mathcal{P}$. (Here, we assume that $\mathcal{N}$ is identified with some finite subset of Pr.)

4.2 Greibach Normal Form Transformation of Context-Free Grammars with Montague Semantics

We describe a procedure for converting a cycle-free context-free grammar with Montague semantics G with $\varepsilon \notin \mathbb{L}(G)$ into an equivalent one in Greibach normal form. This is done in five steps. The first step eliminates all ε-rules from the grammar. The second step eliminates all unit rules. The third step performs the left-corner transform, well-known from the work of Rosenkrantz and Lewis [7], but enriched with semantics. The fourth step takes the result of the previous step and converts it into extended Greibach normal form. The last step then converts it into Greibach normal form. The first four steps roughly mirror the procedure presented in the technical report by Kanazawa and Yoshinaka [2].

Suppose that $G = (\mathcal{N}, \mathcal{T}, \Sigma, f, \mathcal{P}, S)$ is a cycle-free grammar such that $\varepsilon \notin \mathbb{L}(G)$. Let us call a nonterminal B *nullable* if $B \Rightarrow_G^* \varepsilon$. By assumption, S is not nullable. Note that the binary relation $\Rightarrow_G^+$ restricted to $\mathcal{N}$ is a strict partial order. When $A \Rightarrow_G^+ B$ holds, we consider A "less than" B with respect to this partial order.

Elimination of ε-Rules. A rule of the form $B \to \varepsilon : M$ is called an ε-rule. Let C be a nullable nonterminal that is maximal with respect to the strict partial order $\Rightarrow_G^+$. Let $\mathcal{P}_0$ be the set of all ε-rules in $\mathcal{P}$ with C as the left-hand side nonterminal. For each rule π of the form

$$B \to w_0 B_1 w_1 \dots B_n w_n : M[x_1, \dots, x_n],$$

let $\pi \circ \mathcal{P}_0$ consist of all rules of the form

$$B \to w_0 \beta_1 w_1 \dots \beta_n w_n : M[Q_1, \dots, Q_n]$$

such that for some $k_1, \dots, k_n$, each $i \in \{1, \dots, n\}$ satisfies either

– $\beta_i = B_i$, $Q_i = x_{k_i}$, and $k_i = k_{i-1} + 1$, or
– $\beta_i = \varepsilon$, $B_i = C$, $\mathcal{P}_0$ contains the rule $C \to \varepsilon : Q_i$, and $k_i = k_{i-1}$,

where $k_0 = 0$. Let

$$\mathcal{P}' = \bigcup_{\pi \in \mathcal{P} - \mathcal{P}_0} \pi \circ \mathcal{P}_0,$$
$$G' = (\mathcal{N}, \mathcal{T}, \Sigma, \mathcal{P}', S).$$

Lemma 6. *For every $B \in \mathcal{N}$ and $w \in \mathcal{T}^+$, the following are equivalent:*

(i) $\vdash_G B(w, N)$.
(ii) *Either* $\vdash_{G'} B(w, N)$ *or* $B = C$, $w = \varepsilon$, *and* $\mathcal{P}_0$ *contains the rule* $C \to \varepsilon : N$.

Lemma 7. *For every $B \in \mathcal{N}$, B is nullable in G' if and only if $B \neq C$ and B is nullable in G.*

Lemma 8. *For every $B, B' \in \mathcal{N}$, $B \Rightarrow^+_{G'} B'$ if and only if $B \Rightarrow^+_G B'$.*

By Lemma 6, $\mathbb{R}(G') = \mathbb{R}(G)$, and by Lemmas 7 and 8, G' is a cycle-free grammar with one fewer nullable nonterminals than G. By repeating this procedure, we can turn G into an equivalent one that is cycle-free and contains no ε-rules.

Elimination of Unit Rules. A *unit rule* is a rule of the form $B \to B_1 : M[x_1]$. If $G = (\mathcal{N}, \mathcal{T}, \Sigma, f, \mathcal{P}, S)$ is a cycle-free grammar with no ε-rules, we can eliminate unit rules from G by a procedure similar to the one used for the previous step. Let C be a nonterminal in $\mathcal{N}$ that is maximal, but not minimal, with respect to the strict partial order $\Rightarrow^+_G$. This means that there is a unit rule with C as its right-hand side nonterminal, but there is no unit rule with C as its left-hand side nonterminal. Let $\mathcal{P}_{\text{left}}$ be the set of all rules in $\mathcal{P}$ with C as their left-hand side nonterminal, and let $\mathcal{P}_{\text{right}}$ be the set of all unit rules in $\mathcal{P}$ with C as their *right-hand* side nonterminal. Let $\mathcal{P}_{\text{right}} \circ \mathcal{P}_{\text{left}}$ consist of all rules of the form

$$B \to v_0 D_1 v_1 \dots v_{m-1} D_m v_m : N[M[x_1, \dots, x_m]]$$

such that $\mathcal{P}_{\text{right}}$ contains the rule

$$B \to C : N[x_1]$$

and $\mathcal{P}_{\text{left}}$ contains the rule

$$C \to v_0 D_1 v_1 \dots v_{m-1} D_m v_m : M[x_1, \dots, x_m].$$

Let

$$\mathcal{P}' = (\mathcal{P} - \mathcal{P}_{\text{right}}) \cup (\mathcal{P}_{\text{right}} \circ \mathcal{P}_{\text{left}}),$$
$$G' = (\mathcal{N}, \mathcal{T}, \Sigma, \mathcal{P}', S).$$

Lemma 9. $\vdash_{G'} B(w, M)$ *if and only if* $\vdash_G B(w, M)$.

Lemma 10. $B \Rightarrow^+_{G'} B'$ *if and only if* $B \Rightarrow^+_G B'$ *and* $B' \neq C$.

By Lemma 9, $\mathbb{R}(G') = \mathbb{R}(G)$. It is clear that G' is a cycle-free grammar with no ε-rules, and G' has one fewer nonterminals that appear on the right-hand side of unit rules than G. By repeating this procedure, we can obtain a grammar equivalent to G that has no ε- or unit rules.

Left-corner Transform. Let $G = (\mathcal{N}, \mathcal{T}, \Sigma, f, \mathcal{P}, S)$ be a grammar with no ε- or unit rules. Let

$$\mathcal{N}' = \mathcal{N} \cup \{\, [B\backslash C] \mid B, C \in \mathcal{N}\},$$

and define $f' : \mathcal{N}' \to \mathrm{Tp}(\mathcal{A})$ by

$$f'(B) = f(B), \qquad f'([B\backslash C]) = f(B) \to f(C).$$

Define $\mathcal{P}'$ as follows:

– For each rule in $\mathcal{P}$ of the form

$$B \to w_0 B_1 w_1 \dots B_n w_n : M[x_1, \dots, x_n]$$

($n \geq 0$) with $w_0 \neq \varepsilon$ and each $C \in \mathcal{N}$, $\mathcal{P}'$ contains the rules

$$B \to w_0 B_1 w_1 \dots B_n w_n : M[x_1, \dots, x_n],$$
$$C \to w_0 B_1 w_1 \dots B_n w_n [B\backslash C] : x_{n+1} M[x_1, \dots, x_n].$$

– For each rule in $\mathcal{P}$ of the form

$$B \to B_1 w_1 \dots B_n w_n : M[x_1, \dots, x_n]$$

($n \geq 1$) and each $C \in \mathcal{N}$, $\mathcal{P}'$ contains the rules

$$[B_1\backslash B] \to w_1 B_2 w_2 \dots B_n w_n : \lambda z. M[z, x_1, \dots, x_{n-1}],$$
$$[B_1\backslash C] \to w_1 B_2 w_2 \dots B_n w_n \, [B\backslash C] : \lambda z. x_n M[z, x_1, \dots, x_{n-1}],$$

(Note that here, either $n \geq 2$ or $w_1 \neq \varepsilon$, since G has no ε- or unit rules.)

Define $G' = (\mathcal{N}', \mathcal{T}, \Sigma, f', \mathcal{P}', S)$. The following lemma implies $\mathbb{R}(G') = \mathbb{R}(G)$.

Lemma 11. *For every $B, D \in \mathcal{N}$ and $w \in \mathcal{T}^+$, the following equivalences hold:*

(i) $\vdash_G B(w, M)$ *if and only if* $\vdash_{G'} B(w, M)$
(ii) $\vdash_G B(Dw, M[x_1])$ *if and only if* $\vdash_{G'} [D\backslash B](w, \lambda z. M[z])$.

Conversion to Extended Greibach Normal Form. Let G be a grammar with no ε- or unit rule, and let $G' = (\mathcal{N}', \mathcal{T}, \Sigma, f', \mathcal{P}', S)$ be the result of applying the left-corner transform to G. For each rule π of G', if the left-hand side nonterminal of π is some $B \in \mathcal{N}$, then the right-hand side of π starts with a terminal. If the left-hand side nonterminal of π is of the form $[B\backslash C]$, the right-hand side of π starts either with a terminal or with some nonterminal $B_2 \in \mathcal{N}$. Let $\mathcal{P}'_1$ be the set of all rules in $\mathcal{P}'$ that does not start with a terminal, and for each nonterminal $D \in \mathcal{N}$, let $\mathcal{P}'_D$ be the set of all rules in $\mathcal{P}'$ that has D as their left-hand side nonterminal. If $\pi \in \mathcal{P}'_1$ is of the form $\pi = [B\backslash C] \to D_\pi w_1 B_2 w_2 \dots B_n w_n : M[x_1, \dots, x_n]$, let $\pi \circ \mathcal{P}'_{D_\pi}$ consist of all rules

$$[B\backslash C] \to v_0 E_1 v_1 \dots E_m v_m w_1 B_2 w_2 \dots B_n w_n : M[P[x_1, \dots, x_m], x_{m+1}, \dots, x_{m+n-1}]$$

such that $D_\pi \to v_0 E_1 v_1 \dots E_m v_m : P[x_1, \dots, x_m]$ is a rule in $\mathcal{P}'_{D_\pi}$. Let

$$\mathcal{P}'' = (\mathcal{P}' - \mathcal{P}'_1) \cup \bigcup_{\pi \in \mathcal{P}'_1} \pi \circ \mathcal{P}'_{D_\pi},$$
$$G'' = (\mathcal{N}', \mathcal{T}, \Sigma, f', \mathcal{P}'', S).$$

It is easy to see that $\mathbb{R}(G'') = \mathbb{R}(G')$ and G'' is in *extended Greibach normal form* in the sense that the right-hand side of each rule starts with a terminal.

From Extended Greibach Normal Form to Greibach Normal Form. Let $G = (\mathcal{N}, \mathcal{T}, \Sigma, f, \mathcal{P}, S)$ be a grammar in extended Greibach normal form. Let

$$\mathcal{N}' = \mathcal{N} \cup \{\, [Ba] \mid B \in \mathcal{N}, a \in \mathcal{T} \,\},$$

and define $f' \colon \mathcal{N}' \to \mathrm{Tp}(\mathcal{A})$ by

$$f'(B) = f(B), \qquad f'([Ba]) = f(B) \to f(B).$$

If π is a rule of the form

$$C \to a X_1 \dots X_n : M[x_1, \dots, x_m]$$

in $\mathcal{P}$, where $X_i \in \mathcal{N} \cup \mathcal{T}$, and $k_1, \dots, k_m$ and $j_1, \dots, j_{n-m}$ list the elements of $\{\, i \mid X_i \in \mathcal{N} \,\}$ and $\{\, i \mid X_i \in \mathcal{T} \,\}$, respectively, in increasing order, then let π' be the rule

$$C \to a X'_1 \dots X'_n : x_{j_1}(\dots(x_{j_{n-m}} M[x_{k_1}, \dots, x_{k_m}])\dots),$$

where

$$X'_i = \begin{cases} X_i & \text{if } X_i \in \mathcal{N}, \\ [C X_i] & \text{if } X_i \in \mathcal{T}. \end{cases}$$

Let

$$\mathcal{P}' = \{\, [Ba] \to a : \lambda z.z \mid B \in \mathcal{N}, a \in \mathcal{T} \,\} \cup \{\, \pi' \mid \pi \in \mathcal{P} \,\}.$$

Let $G' = (\mathcal{N}', \mathcal{T}, \Sigma, f', \mathcal{P}', S)$. It is clear that $\mathbb{R}(G') = \mathbb{R}(G)$ and G' is in Greibach normal form.

The constructions in this and the previous subsection together give the second half of the main result of this paper:

Theorem 12. *Given any cycle-free context-free grammar with Montague semantics G such that $\varepsilon \notin \mathbb{L}(G)$, one can construct a Lambek grammar G_{L} such that $\mathbb{R}(G) = \mathbb{R}(G_{\mathrm{L}})$.*

References

1. Kanazawa, M.: Computing interpolants in implicational logics. Annals of Pure and Applied Logic 142, 125–201 (2006)
2. Kanazawa, M., Yoshinaka, R.: Lexicalization of second-order ACGs. NII Technical Report NII-2005-012E, National Institute of Informatics, Tokyo (2005)

3. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958)
4. Pentus, M.: Lambek grammars are context free. In: Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science, pp. 429–433 (1993)
5. Pentus, M.: Product-free Lambek calculus and context-free grammars. Journal of Symbolic Logic 62, 648–660 (1997)
6. Pentus, M.: Lambek calculus and formal grammars. In: Provability, Complexity, Grammars. American Mathematical Society Translations–Series 2, vol. (192), pp. 57–86. American Mathematical Society, Providence (1999)
7. Rosenkrantz, D.J., Lewis II, P.M.: Deterministic left corner parsing. In: IEEE Conference Record of the 11th Annual Symposium on Switching and Automata, pp. 139–152. IEEE (1970)

On the Complexity of Free Word Orders

Jérôme Kirman and Sylvain Salvati

LaBRI, CNRS/Université Bordeaux, INRIA, France

Abstract. We propose some extensions of mildly context-sensitive formalisms whose aim is to model free word orders in natural languages. We give a detailed analysis of the complexity of the formalisms we propose.

1 Introduction

Many natural languages present some free word order phenomena. In some sentences, certain words or phrases can be exchanged freely without changing their meanings in an essential way. It is rather usual to model free word order phenomena by means of dependency grammars [6,14]. We here take another approach that is pertaining to the tradition of generative grammars. We try to see how to enrich formalisms that are considered to be *mildly context sensitive* [8] so as to enable them to model free word order phenomena. Even though the capabilities of mildly context sensitive formalisms to model free word orders remains largely unknown [20,12], some evidence based on syntactic structures indicate that they are not well-suited for modeling free word orders in natural languages [2]. Our motivations are thus twofold: first we wish to close the gap between the dependency approaches and the generative approaches to natural language, following a research direction proposed by Kuhlmann [13]; second we wish to see how to increase the expressiveness of well-known mildly context sensitive formalisms so that they can model free word orders. A question related to the second motivation consists in understanding how robust mild context sensitivity is with respect to such extensions. In this paper, we focus on the computational difficulty of the extensions we propose and show that for certain of them the membership problem remains polynomial.

As is well-known and emphasized by formalisms like Abstract Categorial Grammars (ACGs) [5], mildly context sensitive formalisms have a natural counter-part in terms of tree languages, such as regular tree languages, non-duplicating context free tree languages, multiple regular tree languages, tree languages generated by hyperedge replacement grammars... Our approach consists in defining an algebra with letters as basic nullary operators and with two binary operators: (i) an associative and commutative operator that models the possibility of displacing phrases, (ii) an associative operator that models the usual concatenation. Then we use the *tree counterpart* of mildly context sensitive formalisms so as to generate terms over that algebra. Each of these terms, modulo the equational theory of the algebra denotes a finite set of strings. Then the tree languages we construct denote string languages that are the union of

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 209–224, 2013.

the finite languages denoted by the terms the grammar generates. This amounts to representing certain sentences up to the ordering of certain of their elements while the possible orderings still have the same syntactic tree; this models the fact that the semantics is preserved and is, in our opinion, crucial in modeling free word order phenomena. A side effect, is that the formalisms we define are naturally more expressive than their "string" counterparts.

The idea is rather similar to the one proposed by Muskens [15]; we nevertheless try to be more conservative with respect to formal language theory. In particular, we try to stay as close as possible to mildly context-sensitive formalisms. It is also related to the notion of ID/LP grammar [22], but the grammatical formalisms we propose are more expressive than context-free grammars.

The paper thus defines some extensions of well-known mildly context-sensitive formalisms using the techniques that are related to ACGs. We nevertheless adopt a presentation that is close to that of Multiple Context-Free Grammars (MCFG) so as to emphasize the relation with mildly context-sensitive formalisms. We then give an analysis of the complexity of the formalisms we obtain. Most of them have an NP-complete membership problem, but we manage to define two tractable subclasses, one being in NLOGSPACE and being an extension of regular languages, the other being LOGCFL-complete and being an extension of context-free languages. Concerning the universal membership problem, the complexity ranges from NP-hardness up to EXPTIME-completeness for the formalisms we consider.

2 Words Modulo Commutation

2.1 An Algebra for Representing Words Modulo Commutation

We write $[n]$ for the set $\{1, \ldots, n\}$ and given a finite set Σ, we write Σ^* for the set of strings over Σ, ε denoting the empty string.

The set of types *type* is the smallest set containing 0 and so that when α and β are in *type*, $(\alpha \to \beta)$ is also in *type*. As it is usual we consider that the operator $\to$ associates to the right and that $\alpha_1 \to \cdots \to \alpha_n \to 0$ denotes the type $(\alpha_1 \to (\cdots \to (\alpha_n \to 0) \ldots))$. The order $\mathrm{order}(\alpha)$ of a type α is defined by $\mathrm{order}(0) = 1$, $\mathrm{order}(\beta \to \gamma) = \max\{\mathrm{order}(\beta) + 1, \mathrm{order}(\gamma)\}$. The types that have order 2, or *second order types*, are all of the form $0 \to \cdots \to 0 \to 0$. In general, we shall write $0^k \to 0$ for the type defined as $0^0 \to 0 = 0$, $0^{k+1} \to 0 = 0 \to (0^k \to 0)$. Notice that when $k > 0$, $0^k \to 0$ denotes a second order type.

A *signature* Σ is a finite set of typed constants. These constants c^α come with their types written as superscripts. The order of a signature Σ is $\mathrm{order}(\Sigma) = \max\{\mathrm{order}(\alpha) \mid c^\alpha \in \Sigma\}$. In this paper we are going to work only with second order signatures. We assume that for each type α, we have an infinite countable set of λ-variables x^α, y^α, z^α, ... On a signature Σ, we can construct typed λ-terms, $(\Lambda^\alpha(\Sigma))_{\alpha \in type}$ as the smallest sets such that if c^α is in Σ, c^α is in $\Lambda^\alpha(\Sigma)$, if x^α is a variable of type α, x^α is in $\Lambda^\alpha(\Sigma)$, if M is in $\Lambda^{\beta \to \alpha}(\Sigma)$ and N is in $\Lambda^\beta(\Sigma)$, (MN) is in $\Lambda^\alpha(\Sigma)$, and if M is in $\Lambda^\alpha(\Sigma)$, $\lambda x^\beta.M$ is in $\Lambda^{\beta \to \alpha}(\Sigma)$. We let the order of a term be the order of its type and we are going to work mostly

with terms of order at most 2. Notice that variables and constants come with their types, nevertheless, we shall often omit type annotations; we shall also drop parenthesis following the usual conventions of λ-calculus. We write $FV(M)$ for the set of free variables of M. Given a term M, we write $|M|$ for the size of M defined as: $|c| = |x| = 1$, $|MN| = |M| + |N|$, $|\lambda x.M| = |M|$. For a constant a, we write $|M|_a$ for the number of occurrences of the constant a in M; for a set of constants $\mathcal{C}$, we write $|M|_{\mathcal{C}} = \sum_{a \in \mathcal{C}} |M|_a$. A term is linear when for each of its subterms, if it is of the form MN, then $FV(M) \cap FV(N) = \emptyset$ and if it is of the form $\lambda x.M$ then $x \in FV(M)$. Notice that in a linear terms each variable has at most one free occurrence.

We assume the usual notion of λ-calculus (β, η or $\beta\eta$-contraction/reduction/conversion, β-normal form...) and work up to α-conversion. The simultaneous capture avoiding substitution of $M_1, \ldots, M_n$ respectively for $x_1, \ldots, x_n$ in M is written $M[M_1/x_1, \ldots, M_n/x_n]$. Given a function σ that maps variables to terms, we write $M.\sigma$ for $M[\sigma(x_1)/x_1, \ldots, \sigma(x_n)/x_n]$ when $FV(M) = \{x_1, \ldots, x_n\}$ in M.

Given a first order signature (all constants in Σ have type 0) Σ, we define $\mathrm{com}(\Sigma)$ to be the signature $\Sigma \cup \{\varepsilon^0, \bullet^{0^2 \to 0}, \otimes^{0^2 \to 0}\}$ where ε^0 is a constant that is not in Σ. Given a closed term M in β-normal form and of type 0 built on $\mathrm{com}(\Sigma)$, we can map it to a string of Σ^* by simply taking the yield of M as follows:

1. $\mathrm{yield}(a) = a$,
2. $\mathrm{yield}(\varepsilon) = \varepsilon$,
3. $\mathrm{yield}(\bullet M_1\, M_2) = \mathrm{yield}(\otimes M_1\, M_2) = \mathrm{yield}(M_1)\mathrm{yield}(M_2)$.

We write $\overline{\Sigma}$ for $\Sigma \cup \{\varepsilon^0\}$. In particular, for a term M, $|M|_{\overline{\Sigma}}$ denotes the number of occurrences of the elements of $\Sigma \cup \{\varepsilon\}$ that occur in M.

Moreover, we define the least congruence over λ-terms built on $\mathrm{com}(\Sigma)$, written $\equiv_c$, that includes $\beta\eta$-conversion and so that:

$$\begin{aligned} \bullet(\bullet M_1\, M_2)\, M_3 &\equiv_c \bullet M_1\, (\bullet M_2\, M_3) && \text{(Assoc. } \bullet) \\ \otimes(\otimes M_1\, M_2)\, M_3 &\equiv_c \otimes M_1\, (\otimes M_2\, M_3) && \text{(Assoc. } \otimes) \\ \otimes M_1\, M_2 &\equiv_c \otimes M_2 M_1 && \text{(Com. } \otimes) \end{aligned}$$

In a nutshell $\bullet$ is associative while $\otimes$ is associative and commutative. We shall take a *flatten notation* for terms in β-normal form of type 0 whose free variables also have type 0. Given a linear λ-term M in normal form and of type 0 built only with $\otimes$ (*resp.* $\bullet$) and the free variables of type 0 $x_1, \ldots, x_n$ appearing in that order from left to right we write $\{N_1, \ldots, N_n\}$ (*resp.* $N_1 \ldots N_n$) for $M[N_1/x_1, \ldots, N_n/x_n]$. We shall also, given a term N, write N^k to denote $\underbrace{N \ldots N}_{k \text{ times}}$.

Now given a closed term M of type 0, it denotes a finite language $\mathcal{L}(M) = \{\mathrm{yield}(N) \mid M \equiv_c N\}$.

There are other algebras that can be considered so as to represent finite sets of strings. In particular, a rather natural choice consists in using partial orders

generated by series/parallel operations. This amounts to taking $\bullet$ and $\|$ as binary operators to represent the sets. Closed terms of type 0 in normal form are interpreted as languages using a homomorphism as follows:

1. $[\![a]\!] = \{a\}$,
2. $[\![\varepsilon]\!] = \{\varepsilon\}$,
3. $[\![\bullet M_1\, M_2]\!] = \{tu \mid t \in [\![M_1]\!] \wedge u \in [\![M_2]\!]\}$,
4. $[\![\| M_1\, M_2]\!] = \{t_1 u_1 \dots t_n u_n \mid t_1 \dots t_n \in [\![M_1]\!] \wedge u_1 \dots u_n \in [\![M_2]\!]\}$.

Such a choice gives some rather different notion of free word order languages:

Lemma 1. *The class of finite languages that can be described with $\bullet$ and $\otimes$ is incomparable with the one that can be described with $\bullet$ and $\|$.*

Proof. The language $\mathcal{L}(\otimes(\bullet\varepsilon\,(\otimes a\, b))\, c) = \{abc, bac, cab, cba\}$ cannot be described with $\bullet$ and $\|$.

The language $[\![\|(\bullet a\, b)\, c]\!] = \{cab, acb, abc\}$ cannot be described with $\bullet$ and $\otimes$. □

We will explain later on the reason why we prefer to take $\bullet$ and $\otimes$ instead of $\bullet$ and $\|$. For the moment we show that the problem of checking whether a word belongs to the finite language described by a term built on $\bullet$ and $\otimes$ is NP-complete.

Lemma 2. *Given a closed term M of type 0 in β-normal form and a word w, deciding whether w is in $\mathcal{L}(M)$ is NP-complete.*

Proof. We here only prove the NP-hardness; the proof that the problem is in NP relies on the fact that proving that two closed terms of type 0 in β-normal form are equivalent modulo $\equiv_c$ can be done in Ptime.

We are now going to see that the problem is NP-hard using a reduction of the 3-PART problem. An instance $\mathcal{P}$ of the 3-PART problem is given by a sequence $S = s_1, \dots, s_{3m}$ of natural numbers and a number k so that for $1 \leq i \leq 3m$, $\frac{k}{4} < s_i < \frac{k}{2}$. Such an instance has a solution when there is a partition S_1, …, S_m of S so that for every $1 \leq i \leq m$, $\sum_{s \in S_i} s = k$.

So given $\mathcal{P}$ an instance of the 3-PART problem, we construct M and w so that $w \in \mathcal{L}(M)$ iff $\mathcal{P}$ has a solution. The term M is constructed on $\mathrm{com}(\{a, \#\})$. To define M we need first to inductively define on k the terms A_k and H_k by: $A_0 = H_0 = \varepsilon$, $A_{k+1} = \bullet a\, A_k$ and $H_{k+1} = \otimes \#\, H_k$. We then let M and w be defined by (using the flatten notation):

$$M = \{A_{s_1}, \dots, A_{s_{3m}}, H_m\}$$
$$w = (a^k \#)^m.$$

It is rather obvious that w is in $\mathcal{L}(M)$ iff there is M' such that $M' \equiv_c M$, M' is of the form:

$$\{A_{s_{\sigma(1)}}, A_{s_{\sigma(2)}}, A_{s_{\sigma(3)}}, \#, \dots, A_{s_{\sigma(3m)}}\#, \varepsilon\}$$

and $w = \mathrm{yield}(M') = a^{k_0} \# \dots a^{k_{m-1}} \#$ where $k_i = k = s_{\sigma(3i+1)} + s_{\sigma(3i+2)} + s_{\sigma(4i)}$ (notice that the fact that $M' \equiv_c M$ implies that σ is a permutation of $[1, 3m]$). From this it easily follows that w is in $\mathcal{L}(M)$ iff $\mathcal{P}$ has a solution. □

2.2 Semilinearity

Given a finite set Σ, the set of vectors of dimension Σ is $\mathbb{N}^\Sigma$. Given v in $\mathbb{N}^\Sigma$, and a in Σ, we write $v.a$ for the value that v associates to a; we write $v_1 + v_2$ for the sum of two vectors; for a in Σ, we also denote 1_a the vector so that for b in Σ, $1_a.b = 1$ when $b = a$ and $1_a.b = 0$ otherwise. We also write $\|v\|$ for $\max\{v.a \mid a \in \Sigma\}$. We now introduce the notion of *linear* and *semilinear* sets.

Definition 1. *Given a finite set Σ, and v_0, v_1,..., v_n in $\mathbb{N}^\Sigma$, $\mathrm{lin}(v_0, \ldots, v_n)$ denotes the set:*

$$\mathrm{lin}(v_0, \ldots, v_n) = \{v_0 + \sum_{k=1}^{n} k_i v_i \mid k_1, \ldots, k_n \in \mathbb{N}\}$$

A subset V of $\mathbb{N}^\Sigma$ is said linear over Σ *either when it is empty or when there exists v_0, v_1,..., v_n in $\mathbb{N}^\Sigma$ so that $V = \mathrm{lin}(v_0, \ldots, v_n)$.*

A subset V of $\mathbb{N}^\Sigma$ is said semilinear over Σ *when it is a finite union of linear sets over Σ.*

We are now going to see that for a fixed linear set $V = \mathrm{lin}(v_0, \ldots, v_n)$, given k in $\mathbb{N}$ we can compute in $\textsc{nspace}(\log(\|v\|))$ whether a vector v is in V.

Lemma 3. *Given a finite set Σ, we fix a linear set $V = \mathrm{lin}(v_0, \ldots, v_n)$, the problem:*

INPUT. *v in $\mathbb{N}^\Sigma$*
OUTPUT. *yes when v is in V and no when v is not in V*

can be solved in $\textsc{nspace}(\log(\|v\|))$.

Proof. When a number p is smaller than $\|v\|$, it can be represented in space $\log(\|v\|)$. Thus a vector v' in $\mathbb{N}^\Sigma$ so that $\|v'\| \leq \|v\|$ can be represented in space $|\Sigma| \log(\|v\|)$. Now v is in V iff $v = v_0$ or there is i in $[1, n]$ so that $v - v_i$ is in V. When $\|v'\| \leq \|v\|$, we also have $\|v' - v_i\| \leq \|v\|$ so that the new vector can also be represented in space $|\Sigma| \log(\|v\|)$. This characterization thus yields a non-deterministic algorithm that executes in space $\mathcal{O}(\log(\|v\|))$. □

Corollary 1. *Given a finite set Σ, we fix a semilinear set V, the problem:*

INPUT. *v in $\mathbb{N}^\Sigma$*
OUTPUT. *yes when v is in V and no when v is not in V*

can be solved in $\textsc{nspace}(\log(\|v\|))$.

Proof. Since V is semilinear, it is a finite union of linear sets. The algorithm consists in choosing non-deterministically one of the linear sets composing V and then check whether v is in this linear set. From lemma 3 this can be done in $\textsc{nspace}(\log(\|v\|))$. □

Definition 2. *Given a word w in Σ^*, we write $\psi(w)$ for the* Parikh image of *w, i.e. the vector v of $\mathbb{N}^\Sigma$ such that $v.a$ is the number of a occurring in w.*

Given a language L included in Σ^*, we say that L is semilinear *when the set $\psi(L) = \{\psi(w) \mid w \in L\}$ is semilinear.*

3 Commutative λ-Grammars

We are now going to work with the usual mildly context sensitive grammatical formalisms as term generating devices that will produce sets of closed terms of type 0 over the signature $\mathrm{com}(\Sigma)$. Thus a grammar G is going to have a term language $\mathcal{T}(G)$ and a string language $\mathcal{L}(G)$ so that $\mathcal{L}(G) = \bigcup_{t\in\mathcal{T}(G)} \mathcal{L}(t)$. In the next sections, we are going to study the computational complexity of the universal membership and the membership problems for those grammars.

The presentation we are going to give of a grammar is going to be rather general and in line with the usual definition of MCFGs. We thus define the grammars as bottom-up generative devices. We will use some typed predicates A, B, C, ... as non-terminals. We shall write their types as a list of types $[\alpha_1, \ldots, \alpha_n]$. Given a first order signature Σ, we are going to build derivations by means of *Σ-inference rules* of the form:

$$A(M_1, \ldots, M_n) \leftarrow B_1(x_{1,1}, \ldots, x_{1,n_1}), \ldots, B_p(x_{p,1}, \ldots, x_{p,n_p})$$

where:

1. the $x_{i,j}$ are pairwise distinct λ-variables so that if B_i has type $[\beta_{i,1}, \ldots, \beta_{i,n_i}]$, then $x_{i,j}$ has type $\beta_{i,j}$,
2. if A has type $[\alpha_1, \ldots, \alpha_n]$ then M_1, ..., M_n are linear λ-terms in normal form built on $\mathrm{com}(\Sigma)$ and respectively of type α_1, ..., α_n
3. the variables $x_{i,j}$ have at most one occurrence in the M_k's (the rule is *non-deleting* when each variable has exactly one occurrences in the M_k's),
4. the free variables of the M_k's are the variables $x_{i,j}$.

Definition 3. *A* commutative λ-grammar *G is a tuple $(\mathcal{N}, \Sigma, R, S)$ so that:*

1. *$\mathcal{N}$ is a finite set of type* non-terminals*,*
2. *Σ is a first order signature of* terminals*,*
3. *R is a finite set of* Σ-inference rules*,*
4. *S is a non-terminal of $\mathcal{N}$ with type* $[0]$*.*

A grammar is non-erasing *when the rules in R are all non-erasing.*

A derivation judgment *is of the form $\Gamma \vdash_G A(M_1, \ldots, M_n)$ where*

$$\Gamma = B_1(x_{1,1}, \ldots, x_{1,n_1}), \ldots, B_p(x_{p,1}, \ldots, x_{p,n_p})$$

where the $x_{i,j}$ are pairwise distinct variables whose types are determined by the B_i. The derivation judgments are derived as follows; given a rule of R:

$$A(M_1, \ldots, M_n) \leftarrow B_1(x_{1,1}, \ldots, x_{1,n_1}), \ldots, B_p(x_{p,1}, \ldots, x_{p,n_p})$$

if for every $1 \leq i \leq p$, $\Gamma_i \vdash_G B_i(P_{i,1}, \ldots, P_{i,n_i})$ is derivable[1]*, then $\Gamma_1, \ldots, \Gamma_p \vdash_G A(N_1, \ldots, N_n)$ is derivable where N_k is obtained by substituting $P_{i,j}$ for $x_{i,j}$ in M_k and then normalizing the obtained term. The grammar G defines a term language $\mathcal{T}(G) = \{M \mid\vdash_G S(M)\}$ and a string language $\mathcal{L}(G) = \bigcup_{M\in\mathcal{T}(G)} \mathcal{L}(M)$.*

[1] We assume that the variables in the Γ_i are pairwise distinct. If they were not, a renaming would resolve the problem.

We are not going to study all possible commutative λ-grammars that definition 3 allows to define. We are going to use restrictions on the possible types that non-terminals may have and also on the shapes the rules may have. Nevertheless, those restrictions are harmless in terms of expressive power. Due to results in [11], for every term language $\mathcal{T}$ so that there is a λ-grammar G for which $\mathcal{T}(G) = \mathcal{T}$, there will be a λ-grammar G' satisfying the weakest restriction that we will consider and so that $\mathcal{T}(G')$ is also equal to $\mathcal{T}$. The reason why we take those restrictions into account is to make a precise study of the complexities of the membership and the universal membership problems for the grammars satisfying those restrictions. Moreover those restrictions shall allow us to make the connection with various classes of grammars that are used to capture mildly context sensitive languages.

The restrictions are as follows:

CREG every non-terminal has type [0] and the rules are of the form $A[\mathrm{op}\, a\, x] \leftarrow B[x]$ for $\mathrm{op} \in \{\bullet, \otimes\}$,
CCFG every non-terminal has type [0],
CMG every non-terminal has type $[0^k \to 0]$ for some k,
CMREG every non-terminal has type $[0, \ldots, 0]$,
CMCFG every non-terminal has type $[0^{k_1} \to 0, \ldots, 0^{k_p} \to 0]$, for some $k_1, \ldots, k_p$.

When we forbid the use of $\otimes$ for these restrictions we obtain as string languages for **CREG** exactly the class of regular languages, for **CCFG**, the class of context free languages, for **CMG**, the class of languages definable with non-duplicating macro grammars, for **CMREG** and **CMCFG** we obtain languages definable by multiple context-free grammars. If we are concerned with the term languages these classes of grammars define, we obtain for **CREG**, right-branching regular tree languages, for **CCFG**, regular tree languages, for **CMG**, non-duplicating context free tree languages, for **CMREG**, multiple regular tree languages, and for **CMCFG**, tree languages that are definable by hyperedge replacement grammars.

The reason why we do prefer to use the operators $\bullet$ and $\otimes$ rather than the pair $\bullet$ and $\|$ is that the class corresponding to **CREG** we would obtain in that context would coincide with the class of shuffle languages [21]. A problem is that the class obtained by intersecting shuffle languages with regular languages contains languages which are not semilinear [7]. More importantly the rational cone generated by shuffle languages (*i.e.* the class of languages obtained by rational transduction of shuffle languages) is the set of recursively enumerable languages [1]. These results seem to indicate that the class of languages based on the operator $\|$ are not suitable for modeling natural languages. We claim here that on the contrary the rational transductions of languages that are definable by commutative λ-grammars are not only recursive languages, and are also semilinear. We leave the proof of this claim for some future publication. We can nevertheless notice that the languages that those formalisms define are all semilinear.

Theorem 1. *For every commutative λ-grammar G, the language $\mathcal{L}(G)$ is semilinear.*

In the following sections we are going to be interested in two different problems related to the classes of grammars we have just defined.

Definition 4. *The* membership problem *is as follows; we fix a grammar G:*

INPUT. *a word w,*
OUTPUT. *yes when w is in $\mathcal{L}(G)$ and no otherwise.*

The universal membership problem *is as follows:*

INPUT. *a word w, and a grammar G,*
OUTPUT. *yes when w is in $\mathcal{L}(G)$ and no otherwise.*

The main difference between the membership and the universal membership problems is that the grammar G is part of the input of the latter while it is not for the former. Thus, constants coming from the grammar are considered as mere constants in the complexity of the membership problem while they are not in the universal membership problem. For instance, the membership problem for the languages defined by a formula of Monadic Second Order Logic is that of the recognition of a finite state machine and is in LOGSPACE while the universal membership is PSPACE-complete.

4 Universal Membership

4.1 Universal Membership Complexity of CMG CMREG CMCFG

We here give an algorithm that solves the recognition problem for **CMCFG** and as **CMCFG** subsume all the classes of grammars we are interested in, we obtain a recognition algorithm for all of them. This algorithm is going to work in EXPTIME in general and is closely related to the one presented in [9] for MCFGs. The idea behind the algorithm consists mainly in trying to find a derivation using directly the rules of the grammar. So given a grammar $G = (\mathcal{N}, \Sigma, R, S)$, and $w \in \Sigma^*$, the algorithm consists in guessing a term M so that $w = \text{yield}(M)$ and M is in $\mathcal{T}(G)$. One of the difficulties is that, due to the possibility of using ε, M may be of arbitrary size. In order to tackle this difficulty we introduce a simple rewrite system over terms built on $\text{com}(\Sigma)$. This rewrite system is based on the two rules

$$\bullet\varepsilon\,\varepsilon \to_\varepsilon \varepsilon \qquad \otimes\,\varepsilon\,\varepsilon \to_\varepsilon \varepsilon$$

This rewriting system is obviously terminating and confluent, moreover its union with β-contraction also yields a terminating and confluent relation. We may associate to a λ-term M a unique ε-normal form and a unique $\beta\varepsilon$-normal form. It is also easy to see that the following Lemma holds:

Lemma 4. *Given a closed term M of type 0 built on $\text{com}(\Sigma)$, we have:*

1. *if $M \overset{*}{\to}_\varepsilon N$ then $\mathcal{L}(M) = \mathcal{L}(N)$,*
2. *if $w = \text{yield}(M)$ and M is in $\beta\varepsilon$-normal form then $|M|_\varepsilon \leq |w|$ and $|M| \leq 4|w| - 1$.*

Proof. The first item of the Lemma is obvious, for the second statement, in case $w = \text{yield}(M)$ and M is in $\beta\varepsilon$-normal form , it can easily be established by induction on M that $|M|_\varepsilon \leq |w|$, then as M can be seen as a binary tree with at most $2|w|$ leaves, it follows that it contains at most $4|w| - 1$ nodes which establishes the second identity. □

Lemma 5. *Given a linear λ-term, if $M \xrightarrow{k}_\varepsilon N$, then $|M| = |N| + 2k$.*

Proof. It suffices to remark that when $M \to_\varepsilon N$ then $|M| = |N| + 2$ and iterate this identity. □

Lemma 6. *Given two second order linear terms M and N in $\beta\varepsilon$-normal form, if all the variables in $FV(M)$ are second order and σ is a substitution so that for every x, $\sigma(x)$ is linear and $M.\sigma =_{\beta\varepsilon} N$, then:*

1. $|M|_\Sigma + \sum_{x \in FV(M)} |\sigma(x)|_\Sigma = |N|_\Sigma$
2. *for every x, $|\sigma(x)|_\varepsilon \leq |N|_\varepsilon$.*

Proof. The first item of the Lemma is a simple consequence of the linearity of the terms. The second comes from the fact that when contracting a β-redex one may create at most one ε-redex, then analyzing reductions of second order redices we obtain the inequalities. □

Lemma 7. *Given N a term in $\beta\varepsilon$-normal form of type $0^k \to 0$, if $|N|_{\overline{\Sigma}} = l$ then $|N| = 2(k + l) - 1$.*

Proof. Here, $N = \lambda x_1 \ldots x_k.P$ and P can be seen as a binary tree with $k + l$ leaves; k of its leaves being the variables $x_1, \ldots, x_k$ and l other leaves coming from $\overline{\Sigma}$. □

Given a commutative λ-grammar $G = (\mathcal{N}, \Sigma, R, S)$, and a word w, the algorithm consists in:

1. guessing a term M in $\beta\varepsilon$-normal form so that $w = \text{yield}(M)$,
2. check whether there is N in $\mathcal{T}(G)$ so that $N \xrightarrow{*}_\varepsilon M$.

Using Lemma 4, it is easy to see that the first step can be achieved in NP.

For describing how the algorithm solves the second step we introduce fresh typed constants $\perp_\alpha$, and items of the form $\langle A, P_1, \ldots, P_n \rangle$ where A is a non-terminal of $\mathcal{N}$ of type $[\alpha_1, \ldots, \alpha_n]$ and for $1 \leq i \leq n$, either P_i is a linear λ-term in $\beta\varepsilon$-normal form of type α_i or $P_i = \perp_{\alpha_i}$. An item $\langle A, P_1, \ldots, P_n \rangle$ is *G-valid* iff there is a tuple of terms $N_1, \ldots, N_n$ so that $\vdash_G A(N_1, \ldots, N_n)$ is derivable, and for $1 \leq i \leq n$, if $P_i \neq \perp_{\alpha_i}$, then P_i is the ε-normal form of N_i. The algorithm relies on a procedure that decides whether an item is G-valid; it calls it to check whether $\langle S, M \rangle$ is G-valid. Notice that there is M in $\beta\varepsilon$-normal so that $w = \text{yield}(M)$ and $\langle S, M \rangle$ is G-valid iff w is in $\mathcal{L}(G)$.

We are going to present the procedure that establishes whether an item $\langle A, P_1, \ldots, P_n \rangle$ is G-valid by means of an alternating Turing machine that works

in PSPACE. This means that this algorithm can be implemented using a Turing machine that computes in EXPTIME [3]. The machine stores on its working tape the item $\langle A, P_1, \ldots, P_n\rangle$ it is trying to prove G-valid, then it chooses a rule in R:

$$A(M_1 \ldots M_n) \leftarrow B_1(x_{1,1} \ldots x_{1,n_1}) \ldots B_p(x_{p,1} \ldots x_{p,n_p})$$

Let, for $1 \leq k \leq n$, $X_k = \{x_{i,j} \mid 1 \leq j \leq n_i \wedge x_{i,j} \in FV(M_k)\}$, $X = \bigcup_{k\in[n]\wedge P_k\neq\perp} X_k$ and let $Y = \{x_{i,j} \mid 1 \leq i \leq p \wedge 1 \leq j \leq n_i\} - X$. The algorithm guesses a substitution σ such that:

1. for k in $[n]$, if $P_k \neq \perp$, $M_i.\sigma \xrightarrow{*}_{\beta\varepsilon} P_i$,
2. $\sigma(x_{i,j})$ is in $\beta\varepsilon$-normal form when $x_{i,j}$ is in X,
3. $\sigma(x_{i,j}) = \perp$ when $x_{i,j}$ is in Y.

From Lemmas 6 and 7, for every $x_{i,j}$ of type $0^{k_{i,j}} \to 0$, $|\sigma(x_{i,j})| \leq 2(k_{i,j} + |P_i|_{\overline{\Sigma}}) - 1$ from which we can conclude that when the algorithm tries to check whether $\langle S, M\rangle$ is G-valid, the item it stores on its working tape is always of the form $\langle A, P_1, \ldots, P_n\rangle$ with $\sum_{i=1}^{n} |P_i| \leq n|M| + \sum_{i=1}^{n} k_i \leq 4n|w| + \sum_{i=1}^{n} k_i$. This implies that the item can be stored in PSPACE and that moreover, the substitution σ can be found in PSPACE; the algorithm thus makes each transition in PSPACE. Obviously the algorithm is correct and complete, and therefore the membership problem can be solved in EXPTIME [3]. As MCFGs are as special kind of **CMCFG**, and as the universal membership problem for MCFGs is EXPTIME-hard [9], we obtain that the universal membership problem for **CMCFG** is EXPTIME-complete.

Theorem 2. *The universal membership problem for* ***CMCFG*** EXPTIME*-complete.*

It we restrict our attention to non-deleting **CMREG** we can show that this algorithm runs in alternating PTIME in a similar way as the case of LCFRSs is treated in in [9]. As non-deleting **CMREG** subsume non-deleting LCFRSs whose universal membership is PSPACE-complete, we obtain that:

Theorem 3. *The universal membership problem for non-deleting* ***CMREG*** PSPACE*-complete.*

For grammars in **CMG** we know from [9] that the universal membership problem is PSPACE-hard, but we do not know whether it is PSPACE-complete or EXPTIME-complete even for the non-deleting case.

4.2 Universal Memberhip Complexity for CREG and CCFG

We prove here that the universal membership problems for **CREG** and **CCFG** are NP-complete.

Lemma 8. *The universal membership problem for* ***CREG*** *is* NP*-hard.*

Proof. We use a reduction of the NP-complete Exact 3-cover problem, or X3C. An instance $\mathcal{P}$ of the X3C problem is a finite set $\mathcal{M} = \{a_1, \ldots, a_{3m}\}$ and a set $\mathcal{F} = \{S_1, \ldots, S_n\}$ so that for every i in $[1, n]$, $S_i \subseteq \mathcal{M}$ and $S_i = \{a_{i,1}, a_{i,2}, a_{i,3}\}$. The problem $\mathcal{P}$ has a solution iff there is a subset $\mathcal{F}'$ of disjoint elements of $\mathcal{F}$ so that their union is $\mathcal{M}$. Given an instance of X3C such as $\mathcal{P}$, we let $G_{\mathcal{P}} = (\{S\}, \mathcal{M}, R, S)$ be the **CREG** grammar so that the rules of R are precisely the rules of the form $S(\otimes a_{i,1}(\otimes a_{i,2}(\otimes a_{i,3}\, x))) \leftarrow S(x)$ for i in $[1, n]$ or the rule $S(\varepsilon) \leftarrow$. Now it is easy to see that $a_1 \ldots a_{3m}$ is in $\mathcal{L}(G)$ iff $\mathcal{P}$ has a solution. □

As an immediate corollary we obtain:

Corollary 2. *The universal membership problem for* ***CCFG*** *is* NP*-hard.*

Theorem 4. *The universal membership problem for* ***CREG*** *and* ***CCFG*** *is* NP*-complete.*

Proof. To prove this we only need to find an algorithm in NP that solves this problem. Let us suppose that we are given a **CCFG** $G = (\mathcal{N}, \Sigma, R, S)$, and a word w. Then given a word w, we guess a term M in $\beta\varepsilon$-normal form so that $\text{yield}(M) = w$ and there is M' in $\mathcal{T}(G)$ so that $M' \xrightarrow{*}_{\varepsilon} M$. Lemma 4 shows that $|M| \leq 4|w| - 1$, thus checking that $\text{yield}(M) = w$ can be done in polynomial time. It thus remains to prove that checking the existence of M' in $\mathcal{T}(G)$ so that $M' \xrightarrow{*}_{\varepsilon} M$ can also be done in polynomial time. For this, we remark that, for A in N, deciding whether there is a term P so that $P \xrightarrow{*}_{\varepsilon} \varepsilon$, and $\vdash_G A(P)$ is derivable can be decided in polynomial time. For each A in $\mathcal{N}$ for which there is P so that $P \xrightarrow{*}_{\varepsilon} \varepsilon$ and $\vdash_G A(P)$ we add a rule $A(\varepsilon) \leftarrow$ to R. The new grammar $G' = (\mathcal{N}, \Sigma, R', S)$ recognizes M iff there is M' in $\mathcal{T}(G)$ so that $M' \xrightarrow{*}_{\varepsilon} M$. Then checking whether M is in $\mathcal{T}(G')$ can obviously be done in polynomial time showing that the universal membership problem for **CCFG** is in NP. □

5 Membership Problem: The Polynomial Cases

Before we turn to proving that the membership problems for the grammars **CREG** and **CCFG** are tractable, we first introduce some technical notions that will be useful in both cases.

First of all we assume that the rules of the **CCFG**'s over the set of terminals Σ we consider are of one of the forms:

$$A(\text{op}\, x\, y) \leftarrow B(x),\, C(y) \qquad \text{where op} \in \{\bullet, \otimes\} \tag{1}$$

$$A(\text{op}\, x\, a) \leftarrow B(x) \qquad \text{where op} \in \{\bullet, \otimes\} \tag{2}$$

$$A(a) \leftarrow \qquad \text{where } a \in \Sigma \cup \{\varepsilon\} \tag{3}$$

Usual constructs allow to transform any **CCFG** G into a **CCFG** G' respecting this restriction and so that $\mathcal{T}(G) = \mathcal{T}(G')$ (and thus $\mathcal{L}(G) = \mathcal{L}(G')$). We can also make similar transformations that allow us to work without loss of generalization with **CCFG** whose rules are only of the forms (1) and (3).

Definition 5. *Given a* ***CCFG*** $G = (\mathcal{N}, \Sigma, R, S)$*, we define* $\mathbf{c}(G) = (\mathcal{N}, \Sigma, R', S)$ *so that* R' *is the set of rules of* R *that are like the rules (1) or (2) with* $\mathrm{op} = \otimes$*. We then define the language* $\psi(G, A)$ *to be the subset of* $\mathbb{N}^{\mathcal{N} \cup \Sigma}$ *so that*

$$\{\psi(\mathrm{yield}(M.\sigma)) \mid B_1(x_1), \ldots, B_n(x_n) \vdash_{G'} A(M) \wedge \sigma(x_i) = B_i\}$$

The set of vectors in $\psi(G, A)$ represent the commutative strings over $(\mathcal{N} \cup \Sigma)^*$ that can be constructed with G from the non-terminal A. It can intuitively be understood as the set of *commutative sentential forms* that are derivable from A.

A simple consequence of Theorem 1 is that the sets $\psi(G, A)$ are semilinear. Moreover, it is easy to get an actual representation those sets using Parikh's construction [16].

Lemma 9. *Given a* ***CCFG*** $G = (\mathcal{N}, \Sigma, R, S)$*, the sets* $\psi(G, A)$ *are semilinear.*

5.1 Membership Problem for CREG

In this section, we are going to see that the membership problem for **CREG** is in NLOGSPACE.

$$\frac{}{\langle S, 0, S, 0\rangle}\ \textsc{init} \qquad \frac{\langle A, 0, A, i\rangle \quad A(\bullet\, a_{i+1}\, B) \in R}{\langle B, 0, B, i+1\rangle}\ \textsc{scan}$$

$$\frac{\langle A, v, B, i\rangle \quad v + 1_B \in \psi(G, A) \quad v = \psi(w, i, j)}{\langle B, 0, B, j\rangle}\ \textsc{commutative scan}$$

$$\frac{\langle A, v, B, i\rangle \quad B(\otimes\, a\, x) \leftarrow C(x) \in R \quad \|v + 1_a\| \leq n}{\langle A, v + 1_a, C, i\rangle}\ \textsc{commutative guess}$$

$$\frac{\langle A, v, B, j\rangle \quad B(a) \leftarrow \quad \in R \quad v + 1_a = \psi(w, j, n)}{\top}\ \textsc{success}$$

Fig. 1. The NLOGSPACE recognition algorithm for **CREG**

Theorem 5. *The membership problem for* ***CREG*** *is in* NLOGSPACE.

Proof. Let us fix a **CREG** grammar $G = (\mathcal{N}, \Sigma, R, S)$, given a word $w = a_1 \ldots a_n$ we solve the problem whether $w \in \mathcal{L}(G)$ by using items either of the form $\top$, to denote that w has been recognized, or of the form $\langle A, v, B, i\rangle$ where A, B are in $\mathcal{N}$, v is in $\mathbb{N}^{\Sigma}$ and $\|v\| \leq |w|$, and $0 \leq i \leq |w|$. For describing the rules of the algorithm we are going to use the notation with $\psi(w, i, j)$ to denote the Parikh image of the string $a_{i+1} \ldots a_j$ when $i < j$ and the vector 0 otherwise. The algorithm is described by the inference rules of Figure 1. In the rule COMMUTATIVE SCAN we implicitly assume that in the sum $v + 1_B$ the vector v is injected in $\mathbb{N}^{\Sigma \cup \mathcal{N}}$ by giving it value 0 on the coordinates in $\mathcal{N}$ and 1_B is in $\mathbb{N}^{\Sigma \cup \mathcal{N}}$.

We are first going to see that deciding whether an item is derivable can be done in NSPACE($\log(|w|)$). First of all we know the items can all be represented in space $\mathcal{O}((|\Sigma|+1)\log(|w|))$; thus, to prove that the algorithm can be run in NSPACE($\log(|w|)$), it suffices that each rule can be executed within NSPACE($\log(|w|)$). The rules INIT and SCAN pose no problem. The rules COMMUTATIVE GUESS and SUCCESS require checking whether a certain vector has a norm smaller than $|w|$ which can be easily done in NSPACE($\log(|w|)$). Finally the rule COMMUTATIVE SCAN requires to checking the equality of two vectors whose norm is smaller than $|w|$, which can be done in NSPACE($\log(|w|)$), and also that one of those vectors is in the semilinear set $\psi(G, A)$ which according to corollary 1 can be done in NSPACE($\log(|w|)$). This finally shows that the algorithm described by the inference rules of figure 1 can be executed in NLOGSPACE.

It is then easy to see that $\langle A, v, B, i\rangle$ is derivable iff $A(x) \vdash_G S(M)$ so that $a_1 \dots a_i x$ is in $\mathcal{L}(M)$, and $B(x) \vdash_G A(N)$ so that $N = \otimes b_1(\cdots(\otimes b_r x)\cdots)$ with $b_1, \dots, b_r$ in $\Sigma \cup \{\varepsilon\}$ and $\psi(\text{yield}(N)) = v$. From this it easily follows that $\top$ is derivable iff w is in $\mathcal{L}(G)$. □

5.2 Membership Problem for CCFG

In this section we are going to see that the membership problem for **CCFG** is LOGCFL-complete. As every language definable by a context-free grammar can be seen as a **CCFG** we know that the membership problem for **CCFG** is LOGCFL-hard. We are thus going to see that this problem is actually in LOGCFL.

Theorem 6. *The membership problem for **CCFG** is* LOGCFL*-complete.*

Proof. We consider a **CCFG** $G = (\mathcal{N}, \Sigma, R, S)$ whose rules are of the forms (1) and (3). We are going to show that there is an alternating Turing machine (ATMs) that recognizes the word w in $\mathcal{L}(G)$ in a space $\mathcal{O}(\log(|w|))$ whose accepting computation trees have size $\mathcal{O}(|w|^{|\mathcal{N}|+1})$ (the size of a computation tree is the number of nodes that are visited by the machine together with all the possible successors of existential nodes). The results of Ruzzo [18], imply then that the problem is in LOGCFL. Before we describe the algorithm for a given word $w = a_1 \dots a_n$, for $0 \le i \le j \le n$ we write $w[i, j]$ for $a_{i+1} \dots a_j$ when $i < j$ and ε when $i = j$. We describe the machine by means of the inference system of figure 2 that works with items of the form $\langle v, i, j\rangle$ where v is in $\mathbb{N}^{\mathcal{N}}$ so that $\|v\| \le |w|$ and $0 \le i \le j \le |w|$. The machine accepts a word $w = a_1 \dots a_n$ if it can derive the item $\langle 1_S, 0, n\rangle$. The working tape of the machine contains the item that is the current goal of the inference system. The choice of a rule is made by an existential choice, and when a rule is chosen, all the premises are needed to be proven using alternation. For a given item we need to give an upper bound on the number of instances of the rules that can be applied to pursue the derivation that is polynomial in $|w|$. The key observation to prove this consists in noticing that there are $\mathcal{O}(\log(|w|)^{|\mathcal{N}|+2})$ possible items so that each rule may have only have polynomially many instances while the machine is trying to recognize w. Moreover each item can be represented in space $\mathcal{O}((|\mathcal{N}|+2)\log(|w|)) = \mathcal{O}(\log(|w|))$.

$$\frac{\langle v_1, i, j\rangle \quad \langle v_2, j, k\rangle \quad 0 < v_1 \quad 0 < v_2 \quad \|v_1 + v_2\| \leq |w|}{\langle v_1 + v_2, i, k\rangle}\ \text{COMMUTATIVE COMBINE}$$

$$\frac{\langle v, i, j\rangle \quad v \in \psi(G, A)}{\langle 1_A, i, j\rangle}\ \text{COMMUTATIVE REDUCTION}$$

$$\frac{\langle 1_B, i, j\rangle \quad \langle 1_C, j, k\rangle \quad A(\bullet x\, y) \leftarrow B(x), C(y)}{\langle 1_A, i, k\rangle}\ \text{COMBINE}$$

$$\frac{A(a) \quad a = w[i, j]}{\langle 1_A, i - 1, i\rangle}\ \text{CONSTANT}$$

Fig. 2. The LOGCFL recognition algorithm for **CCFG**

Finally it is easy to see that a tree that derives an item $\langle v, i, j\rangle$ with the inference system of figure 2 contains exactly $2(j - i - 1) - 1$ nodes, which implies that the accepting derivations of $\langle S, 0, n\rangle$ have size $2|w| - 1$. An accepting computation tree of the machine is thus made of a derivation tree of $\langle S, 0, n\rangle$ together with all the possible rules that can be applied at a given node of that tree; therefore such a tree has size $\mathcal{O}(\log(|w|)^{|N|+2}|w|)$. This finally shows that the machine is implementing a LOGCFL algorithm. It is rather straightforward to prove that w is in $\mathcal{L}(G)$ iff $\langle S, 0, n\rangle$ is derivable with the inference system of figure 2. For this it suffices to remark that $\langle v, i, j\rangle$ iff there is $u = A_1 \dots A_k$ in $\mathcal{N}^*$ so that $\psi(u) = v$, and $w[i, j] = u_1 \dots u_k$ with, for all $1 \leq i \leq k$, $\vdash_G A_i(u_i)$. This finally shows that the membership problem is in LOGCFL. □

6 Membership Problem: The Intractable Cases

In this section we are going to see that the membership problems for **CMG**, **CMREG**, **CMCFG** are NP-hard. As every commutative λ-grammar that is a **CMG** or a **CMREG** is also a **CMCFG**, we only need to prove that the membership problems for **CMG** and **CMREG** are NP-hard.

For this we shall use reductions from the 3-PART problem that we have already used to prove Lemma 2.

Lemma 10. *The membership problem for **CMREG** is* NP*-hard.*

Proof. We are going to construct a grammar **CMREG** $G = (\mathcal{N}, \Sigma, R, S)$ so that for every instance $\mathcal{P}$ of 3-PART there is a word $w_\mathcal{P}$ of size linear within the size of $\mathcal{P}$ so that $w_\mathcal{P}$ is in $\mathcal{L}(G)$ iff $\mathcal{P}$ has a solution. We construct G as follows: $\mathcal{N} = \{A, S\}$ with A of type $[0, 0, 0, 0]$ and S of type $[0]$; $\Sigma = \{a, b, \#\}$; then the rules in R (using the flatten notation) are: $S(\{x_1, x_2, x_3, x_4, y\}) \leftarrow A(x_1, x_2, x_3, x_4)\, S(y)$, $S(\varepsilon) \leftarrow$, $A(a\, x_1, x_2, x_3, b\, x_4) \leftarrow A(x_1, x_2, x_3, x_4)$, $A(x_1, a\, x_2, x_3, b\, x_4) \leftarrow A(x_1, x_2, x_3, x_4)$, $A(x_1, x_2, a\, x_3, b\, x_4) \leftarrow$

$A(x_1, x_2, x_3, x_4)$, and $A(\#, \#, \#, \#) \leftarrow$. It is easy to see that $A(M_1, M_2, M_3, N)$ is derivable iff $M_1 = a^{k_1}\#$, $M_2 = a^{k_2}\#$, $M_3 = a^{k_3}\#$ and $N = b^k\#$ with $k = k_1 + k_2 + k_3$. Then it follows that, $\vdash_G S(M)$ is derivable iff $M = \{a^{k_{1,1}}\#, a^{k_{1,2}}\#, a^{k_{1,3}}\#, b^{k_1}\#, \ldots, a^{k_{n,1}}\#, a^{k_{n,2}}\#, a^{k_{n,3}}\#, b^{k_n}\#, \varepsilon\}$ for some n and for every $1 \leq i \leq n$, $k_i = k_{i,1} + k_{i,2} + k_{i,3}$. Now, given an instance $\mathcal{P}$ of 3-PART that consists in the sequence $S = s_1, \ldots, s_{3m}$ of natural numbers, and a number k so that for $1 \leq i \leq 3m$, $\frac{k}{4} < s_i < \frac{k}{2}$, it is easy to see that $w_{\mathcal{P}} = a^{s_1}\# \ldots a^{s_{3m}}\#(b^k\#)^m$ is in $\mathcal{L}(G)$ iff $\mathcal{P}$ has a solution. □

Lemma 11. *The membership problem for **CMG** is* NP*-hard.*

Proof. The proof is based on the definition of the same language as in the proof of Lemma 10. □

Lemma 12. *The membership problem for **CMCFG** is in NP.*

Proof. Let us consider a grammar $G = (\mathcal{N}, \Sigma, R, S)$ and a word w on Σ^*. Using the results of [10] we know that there is a grammar G' that recognizes precisely the set $\{M \mid M \text{ in } \beta\varepsilon\text{-normal form and } \exists N \in \mathcal{T}(G).N \xrightarrow{*}_{\varepsilon} M\}$. Notice that $\mathcal{L}(G') = \mathcal{L}(G)$. We construct an algorithm in NP that checks whether w is in $\mathcal{L}(G')$. It guesses a term M in $\beta\varepsilon$-normal form so that $\text{yield}(M) = w$ and M is in $\mathcal{T}(G)$. Checking that M is in $\mathcal{T}(G)$ can be done in PTIME according to [19]. This completes the proof. □

Theorem 7. *The membership problems for **CMG**, **CMREG** and **CMCFG** are* NP*-complete.*

7 Conclusion

The classes of grammars we propose in the paper are quite close to the classes of grammars that are mildly context sensitive and should thus offer the same kind of ease in modeling natural languages. Moreover, if we put aside the informal condition *limited cross-serial dependencies*, the languages defined by **CCFG** satisfy the definition of mildly context sensitive languages. The uniform presentation that we have given to our formalisms together with their closeness to usual mildly context sensitive formalisms should allow the definition of some extensions of **CCFG** that are still mildly context sensitive.

Besides closing the picture concerning the universal memberhip problems for commutative λ-grammars, there are other research directions that we shall follow. A first one is to give the characterizations of certain natural families of languages that are generated by the families of languages that are definable with the grammars we proposed, such as the rational cones or the abstract families of languages they generate. A second one is to give a comparison of their expressive power with some known formalisms that share the same properties such as Unordered Vector Grammars [4], or Multiset-Valued LIG [17].

References

1. Araki, T., Tokura, N.: Flow languages equal recursively enumerable languages. Acta Informatica 15(3), 209–217 (1981)
2. Becker, T., Rambow, O., Niv, M.: The derivational generative power of formal systems or scrambling is beyond LCFRS. Technical Report IRCS-92-38, UPENN (1992)
3. Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. JACM 28(1), 114–133 (1981)
4. Dassow, J.: Grammars with regulated rewriting. In: Formal Languages and Applications, pp. 249–273. Springer (2004)
5. de Groote, P.: Towards abstract categorial grammars. In: ACL (ed.). Proceedings 39th Annual Meeting and 10th Conference of the European Chapter, pp. 148–155 (2001)
6. Gerdes, K., Kahane, S.: Word order in German: A formal dependency grammar using a topological hierarchy. In: ACL, pp. 220–227 (2001)
7. Jedrzejowicz, J., Szepietowski, A.: On the expressive power of the shuffle operator matched with intersection by regular sets. RAIRO 35(4), 379–388 (2001)
8. Joshi, A.K.: Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions? In: Natural Language Parsing, pp. 206–250. CUP (1985)
9. Kaji, Y., Nakanishi, R., Seki, H., Kasmi, T.: The universal recognition problems for multiple context-free grammars and for linear context-free rewriting systems. IEICE Trans. Inf. & Syst. E 75-D(1), 78–88 (1992)
10. Kanazawa, M.: Abstract families of abstract categorial languages. In: 13th WoLLIC, ENTCS, pp. 65–80 (2006)
11. Kanazawa, M.: Second-order abstract categorial grammars as hyperedge replacement grammars. Journal of Logic, Language and Information 19(2), 137–161 (2010)
12. Kanazawa, M., Salvati, S.: MIX is not a tree-adjoining language. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers, vol. 1, pp. 666–674. ACL (2012)
13. Kuhlmann, M., Möhl, M.: Mildly context-sensitive dependency languages. In: ACL (2007)
14. Lopatková, M., Plátek, M., Kuboň, V.: Modeling syntax of free word-order languages: Dependency analysis by reduction. In: Matoušek, V., Mautner, P., Pavelka, T. (eds.) TSD 2005. LNCS (LNAI), vol. 3658, pp. 140–147. Springer, Heidelberg (2005)
15. Muskens, R.: Separating syntax and combinatorics in categorial grammar. Research on Language and Computation 5(3), 267–285 (2007)
16. Parikh, R.: On context-free languages. JACM 13(4), 570–581 (1966)
17. Rambow, O.: Multiset-valued linear index grammars: imposing dominance constraints on derivations. In: Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics, pp. 263–270. ACL (1994)
18. Ruzzo, W.L.: Tree-size bounded alternation. JCSS 21(2), 218–235 (1980)
19. Salvati, S.: Problèmes de filtrage et problème d'analyse pour les grammaires catégorielles abstraites. PhD thesis, INPL (2005)
20. Salvati, S.: Mix is a 2-MCFL and the word problem in $\mathbb{Z}^2$ is solved by a third-order collapsible pushdown automaton. Technical report, INRIA (2011)
21. Shaw, A.: Software descriptions with flow expressions. IEEE Transactions on Software Engineering 4(3), 242–254 (1978)
22. Shieber, S.: Direct parsing of ID/LP grammars. Linguistics and Philosophy 2(7), 135–154 (1984)

Determiner Gapping as Higher-Order Discontinuous Constituency

Yusuke Kubota[1] and Robert Levine[2]

[1] Ohio State University
kubota@ling.ohio-state.edu
[2] Ohio State University
levine@ling.ohio-state.edu

Abstract. We argue that an approach to discontinuous constituency via prosodic lambda binding initiated by Oehrle (1994) and adopted by some subsequent authors (de Groote, 2001; Muskens, 2003; Pollard, 2011) needs to recognize higher-order prosodic variables to provide a fully systematic treatment of two recalcitrant empirical phenomena exhibiting discontinuity, namely, split gapping involving determiners and comparative subdeletion. Once we admit such higher-order prosodic variables, straightforward analyses of these phenomena immediately emerge. We take this result to provide strong support for recognizing such higher-order prosodic variables in this type of approach. We also touch on the more general issue of alternative approaches to discontinuity in categorial grammar, and suggest that an approach that recognizes (possibly higher-order) prosodic functors like the one we propose here leads to a more principled treatment of certain interactions between phenomena exhibiting complex types of discontinuity than competing approaches.

Keywords: Gapping, split gapping, split scope, comparative subdeletion, categorial grammar, discontinuous constituency.

1 Introduction

An approach that mediates a flexible mapping between the combinatoric component of syntax and the surface string component by recognizing functional expressions in the latter was initiated by Oehrle (1994), and was adopted in certain subsequent variants of categorial grammar (CG) mainly due to its theoretical elegance (whereby one can relegate word order from the combinatoric component entirely to the prosodic component; see, e.g., de Groote (2001); Muskens (2003); Pollard (2011)). The original empirical motivation for this approach came from a simple and systematic treatment of quantification (of generalized quantifiers), but recently a wider range of empirical facts have been adduced to it by Kubota and Levine (2012) and Pollard and Smith (2012), which respectively deal with Gapping and the semantics of symmetrical predicates and related phenomena (the latter via the notion of 'parasitic scope' *a la* Barker (2007)). We here extend this empirical investigation one step further. While previous approaches in this tradition recognize only variables over string-type expressions,

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 225–241, 2013.

we argue that certain linguistic phenomena call for recognizing prosodic variables of a higher type.[1] Recognizing such higher-order variables enables treating types of discontinuity that are much more complex than is possible by just allowing for string-type gaps. We suggest that at least some of the cases in which the phenomena we deal with below interact with one another call for the fully general treatment of discontinuity that is made possible by generalizing the approach this way. We will point out that alternative approaches to discontinuity that are essentially descriptively equivalent to an approach that recognizes only string-type variables in the Oehrle-style setup (of which the recent proposal by Morrill et al. (2011) is representative) cannot adequately deal with such cases.

2 Split Gapping with Determiners

2.1 Split Gapping Is Gapping

Our first case involves a somewhat odd version of gapping first noted by McCawley (1993), which is exemplified by the following sentence:

(1) Too many setters are named Kelly, and shepherds Fritz.

Here, in addition to the verb, the determiner is missing from the second conjunct. We call this construction *determiner gapping*.

McCawley (1993) also noted that in determiner gapping, the verb obligatorily undergoes gapping, together with the determiner. Thus:

(2) ??Too many setters are named Kelly, and shepherds are named Fritz.

Whatever its exact nature, the reduced acceptability of (2) suggests that determiner gapping is indeed a case of Gapping, since reduced acceptability of the verb non-gapped version is found in other types of discontinuous gapping as well:

(3) a. Robin wants Leslie to win, and Terry __ Peter __.
b.??Robin wants Leslie to win, and Terry wants Peter __.

In other respects too, determiner gapping parallels normal Gapping. Note first that, in both constructions, the relevant deletion operation (however one characterizes it theoretically) can target strings consisting of chains of verbs:

(4) a. Most professors [want to try to get] extra teaching, and most students, __ a summer job.
b. Too many professors want to try to get extra teaching, and students __ good-paying jobs, for us to cut the budget for summer.

[1] When we say variables of higher-order type, we mean variables posited in the calculus, which can enter into hypothetical reasoning in the derivation. This shouldn't be confused with metavariables for writing functional phonologies of linguistic expressions, for which higher-order types are already present in Oehrle (1994) (cf. below).

Second, not just (mono-)transitive verbs but verbs taking multiple arguments can undergo Gapping, and this carries over to determiner gapping as well.

(5) a. Robin sent a chess set to the King of Norway, and Leslie, a box of chocolates to the Queen of the Netherlands.
b. Too many men sent chess sets to the King of Norway, and women, boxes of chocolate to the Queen of the Netherlands.

Finally, there is one particularly striking parallel. In addition to examples like (1) (which may lend themselves to a simple deletion-based analysis), McCawley (1993) notes examples like the following for which simply recovering the missing material in the gapped conjunct does not yield a synonymous paraphrase:

(6) a. {No/Few/Hardly any} dog(s) eat(s) Whiskas or cat(s) Alpo.
b. $\neq$ {No/Few/Hardly any} dog(s) eat(s) Whiskas or {no/few/hardly any} cat(s) eat(s) Alpo.

To assign the right meaning to (6a), one has to somehow let the negation that is part of the negative quantifier scope over the disjunction. In other words, there is an apparent mismatch between the surface form and semantic scope.

This may look rather anomalous, but in fact, a precisely parallel scope mismatch is found in ordinary Gapping, as noted by Siegel (1984) and Oehrle (1987):

(7) a. Mrs. J can't live in LA and Mr. J in Boston.
b. Kim didn't play bingo or Sandy sit at home all evening.

The preferred reading for (7a) is one in which the negated modal scopes over the conjunction, i.e., the $\neg\Diamond(p \wedge q)$ interpretation. Similarly for (7b).

The following data provide further parallel between the two types of gapping:

(8) a. No positron can occupy the INner shell and electron the OUTer shell of the same atom.
b. A positron can't occupy the INner shell and some electron the OUTer shell of the same atom.

Both (8a) and (8b) correspond to $\neg\Diamond(\exists_x \psi(x) \wedge \exists_y \varrho(y))$ in meaning. So far as we are aware, Gapping is the only phenomenon in which an auxiliary scopes out of its local clause. The fact that this possibility is also realized in determiner gapping convincingly indicates that it is indeed a species of Gapping.

In view of this parallel between ordinary Gapping and determiner gapping, we propose an analysis which treats the latter as a special case of the former. For this purpose, we build on the treatment of Gapping by Kubota and Levine (2012), which is couched in a variant of categorial grammar called Hybrid Type-Logical Categorial Grammar (Hybrid TLCG). The central feature of this framework is that it recognizes both directional slashes (i.e. forward and backward

slashes) familiar from standard TLCG (going back to Lambek (1958)), and a non-directional slash tied to prosodic λ-binding in more recent variants of CG (Oehrle, 1994; de Groote, 2001; Muskens, 2003; Pollard, 2011). Kubota and Levine show how the apparently anomalous scoping pattern of auxiliaries falls out straightforwardly in such a setup. In what follows, building on this analysis of Gapping, we formulate an analysis of the determiner gapping data above.

Capturing the scopal relation between negative quantifiers and disjunction in examples like (6a) requires the so-called split scope analysis of negative quantifiers like *no* and *few*, where they are decomposed into a wide-scoping sentential negation and a non-negative quantifier meaning ($no = \neg + \exists$, $few = \neg + \mathbf{many}$; cf., e.g., Jacobs (1980); Johnson (2000); Penka (2011)). To the best of our knowledge, an exact implementation of split scope in CG is still an open question. In what follows, we suggest two alternatives for implementing split scope in Hybrid TLCG. The two approaches have more or less the same empirical coverage, but they differ in how exactly the decomposition of the two meaning components of negative quantifiers is mediated. The simpler approach that we present first involves an empty operator and a diacritic syntactic category, thereby directly separating the two meaning components in the combinatoric structure of the sentence, whereas the more sophisticated approach encodes the scope split directly within the lexicon, by treating negative determiners as lexically type-raised determiners. As we show below, the apparent scope anomaly of data like (6a) becomes a non-anomaly in both approaches, once the analysis of negative quantifiers is combined with an analysis of determiner gapping which is a straightforward extension of the Kubota-Levine analysis.

2.2 Kubota and Levine's (2012) Analysis of Gapping

The key analytic idea of Kubota and Levine's (2012) analysis of Gapping is that Gapping involves coordinating two (or more) sentences in which the verb is missing in the middle. This involves explicitly modelling such gapped sentences which essentially manifest discontinuous constituency as conjoinable categories. For this purpose, Kubota and Levine exploit the 'hybrid' nature of their calculus, which is equipped with rules for Elimination and Introduction for both directional slashes ($/$ and $\backslash$) and the order-insensitive non-directional slash (notated as $|$). The following is the complete set of inference rules of Hybrid TLCG:

(9)

Connective	Introduction	Elimination
$/$	$\dfrac{\dfrac{\vdots \quad [\varphi;\ x;\ A]^n \quad \vdots}{b \circ \varphi;\ \mathscr{F};\ B}}{b;\ \lambda x.\mathscr{F};\ B/A}\ /\mathrm{I}^n$	$\dfrac{a;\ \mathscr{F};\ A/B \quad b;\ \mathscr{G};\ B}{a \circ b;\ \mathscr{F}(\mathscr{G});\ A}\ /\mathrm{E}$

$$\backslash \qquad \dfrac{\dfrac{\vdots \quad \underline{[\varphi;\ x;\ A]^n} \quad \vdots}{\varphi \circ b;\ \mathscr{F};\ B}}{b;\ \lambda x.\mathscr{F};\ A\backslash B}\backslash\mathrm{I}^n \qquad \dfrac{b;\ \mathscr{G};\ B \quad a;\ \mathscr{F};\ B\backslash A}{b \circ a;\ \mathscr{F}(\mathscr{G});\ A}\backslash\mathrm{E}$$

$$| \qquad \dfrac{\dfrac{\vdots \quad \underline{[\varphi;\ x;\ A]^n} \quad \vdots}{b;\ \mathscr{F};\ B}}{\lambda\varphi.b;\ \lambda x.\mathscr{F};\ B|A}|\mathrm{I}^n \qquad \dfrac{a;\ \mathscr{F};\ A|B \quad b;\ \mathscr{G};\ B}{a(b);\ \mathscr{F}(\mathscr{G});\ A}|\mathrm{E}$$

The difference between /,\ and | is that while the rules for /,\ refer to the phonological forms of the input and output strings (so, for example, the applicability of the /I rule is conditioned on the presence of the phonology of the hypothesis φ on the right periphery of the phonology of the input $b \circ \varphi$), the rules for | are not constrained that way. For reasoning involving |, the phonological terms themselves fully specify the ways in which the output phonology is constructed from the input phonologies. Specifically, for |, the phonological operations associated with the Introduction and Elimination rules mirror exactly the semantic operations for these rules: function application and λ-abstraction. Thus, the order of the premises in the Elimination rules isn't relevant for any of these connectives; linear order is recorded in the phonological terms of the linguistic expressions (and not in the forms of the proofs) for reasoning involving / and \.

As shown by Oehrle (1994), hypothetical reasoning with a mode of implication associated with λ-binding enables a straightforward and formally explicit implementation of Montague's (1973) quantifying-in, as illustrated in (10):

(10)

$$\dfrac{\lambda\sigma.\sigma(\text{everyone});\ \forall_{\mathbf{person}};\ \mathrm{S}|(\mathrm{S}|\mathrm{NP}) \qquad \dfrac{\lambda\sigma.\sigma(\text{someone});\ \exists_{\mathbf{person}};\ \mathrm{S}|(\mathrm{S}|\mathrm{NP}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{talked} \circ \text{to};\ \mathbf{talk\text{-}to};\ (\mathrm{NP}\backslash\mathrm{S})/\mathrm{NP} \quad \left[\begin{smallmatrix}\varphi_1;\\ x;\ \mathrm{NP}\end{smallmatrix}\right]^1}{\text{talked} \circ \text{to} \circ \varphi_1;\ \mathbf{talk\text{-}to}(x);\ \mathrm{NP}\backslash\mathrm{S}} \quad \left[\begin{smallmatrix}\varphi_2;\\ y;\ \mathrm{NP}\end{smallmatrix}\right]^2}{\varphi_2 \circ \text{talked} \circ \text{to} \circ \varphi_1;\ \mathbf{talk\text{-}to}(x)(y);\ \mathrm{S}} \quad \begin{smallmatrix}\text{yesterday};\\ \mathbf{yest};\ \mathrm{S}\backslash\mathrm{S}\end{smallmatrix}}{\varphi_2 \circ \text{talked} \circ \text{to} \circ \varphi_1 \circ \text{yesterday};\ \mathbf{yest}(\mathbf{talk\text{-}to}(x)(y));\ \mathrm{S}}}{\lambda\varphi_2.\varphi_2 \circ \text{talked} \circ \text{to} \circ \varphi_1 \circ \text{yesterday};\ \lambda y.\mathbf{yest}(\mathbf{talk\text{-}to}(x)(y));\ \mathrm{S}|\mathrm{NP}}}{\text{someone} \circ \text{talked} \circ \text{to} \circ \varphi_1 \circ \text{yesterday};\ \exists_{\mathbf{person}}(\lambda y.\mathbf{yest}(\mathbf{talk\text{-}to}(x)(y)));\ \mathrm{S}}}{\lambda\varphi_1.\text{someone} \circ \text{talked} \circ \text{to} \circ \varphi_1 \circ \text{yesterday};\ \lambda x.\exists_{\mathbf{person}}(\lambda y.\mathbf{yest}(\mathbf{talk\text{-}to}(x)(y)));\ \mathrm{S}|\mathrm{NP}}}{\text{someone} \circ \text{talked} \circ \text{to} \circ \text{everyone} \circ \text{yesterday};\ \forall_{\mathbf{person}}(\lambda x.\exists_{\mathbf{person}}(\lambda y.\mathbf{yest}(\mathbf{talk\text{-}to}(x)(y))));\ \mathrm{S}}$$

Quantifiers are entered in the lexicon in the S|(S|NP) type, with the standard generalized quantifier meaning and a phonology that is a higher-order function over strings of type $(st \to st) \to st$ (with st the type of strings), which 'lowers' the quantifier string in the position in the sentence (bound by the λ-operator in the phonology) corresponding to the semantic variable bound. As in Montague's quantifying-in, the order in which the quantifier combines with the sentence that it lowers into determines its scope. Thus, the above derivation yields the inverse scope interpretation; if the object quantifier is introduced first in the derivation, we get the surface scope interpretation.

The treatment of discontinuous constituency by recognizing functional phonologies has wider empirical applications than just quantification. Kubota and Levine (2012) demonstrate this via an analysis of Gapping. Since expressions containing medial gaps can be modelled via hypothetical reasoning with the vertical slash |, expressions like *Robin* __ *a book* in (11) can be directly analyzed as a sentence missing a transitive verb, of category S|TV (with TV = (NP\S)/NP), as in (12).

(11) Leslie bought a CD, and Robin, a book.

(12)
$$\dfrac{\dfrac{\lambda\sigma_1.\sigma_1(\mathsf{a}\circ\mathsf{book});\ \exists_{\mathbf{book}};\ \mathrm{S|(S|NP)} \qquad \dfrac{\mathsf{robin};\ \mathbf{r};\ \mathrm{NP} \qquad \dfrac{[\varphi_1;P;\mathrm{VP/NP}]^1 \qquad [\varphi_2;x;\mathrm{NP}]^2}{\varphi_1\circ\varphi_2;\ P(x);\ \mathrm{VP}}}{\mathsf{robin}\circ\varphi_1\circ\varphi_2;\ P(x)(\mathbf{r});\ \mathrm{S}}}{\lambda\varphi_2.\mathsf{robin}\circ\varphi_1\circ\varphi_2;\ \lambda x.P(x)(\mathbf{r});\ \mathrm{S|NP}}}{\mathsf{robin}\circ\varphi_1\circ\mathsf{a}\circ\mathsf{book};\ \exists_{\mathbf{book}}(\lambda x.P(x)(\mathbf{r}));\ \mathrm{S}}}{\lambda\varphi_1.\mathsf{robin}\circ\varphi_1\circ\mathsf{a}\circ\mathsf{book};\ \lambda P.\exists_{\mathbf{book}}(\lambda x.P(x)(\mathbf{r}));\ \mathrm{S|(VP/NP)}}$$

By binding the hypothetically assumed TV at the last step of (12), we obtain an expression with a functional phonology (of type $st \to st$), where the phonological variable φ_1 keeps track of the position of the missing verb.

For coordinating such $st \to st$ functions (phonologically), Kubota and Levine introduce the following Gapping-specific lexical entry for the conjunction:

(13) $\lambda\sigma_2\lambda\sigma_1\lambda\varphi[\sigma_1(\varphi)\circ\mathsf{and}\circ\sigma_2(\varepsilon)];\ \lambda\mathscr{W}\lambda\mathscr{V}.\mathscr{V}\sqcap\mathscr{W};\ \mathrm{(S|TV)|(S|TV)|(S|TV)}$

Syntactically, (13) coordinates two sentences missing the main verb (i.e. S|TV) to produce a larger expression of the same type, instantiating the general like-category coordination schema; correspondingly, the semantics is that of generalized conjunction, again conforming to the general treatment of coordination. The only slight complication is in the phonology, where it is specified that the 'gap' position of the first conjunct is retained (so that the main verb can 'lower' into this position at a later step in the derivation) while the corresponding gap in the second conjunct is closed off by feeding an empty string ε to it.

With this conjunction lexical entry, (11) can be derived as in (14):

(14)

$$\dfrac{\dfrac{\dfrac{\vdots \quad \vdots}{\lambda\varphi_1.\mathsf{leslie}\circ\varphi_1\circ \mathsf{a}\circ\mathsf{CD};\ \lambda Q.\exists_{\mathbf{CD}}(\lambda y.Q(y)(\mathbf{l}));\ \mathrm{S}|\mathrm{TV}} \quad \dfrac{\lambda\sigma_2\lambda\sigma_1\lambda\varphi_0.\sigma_1(\varphi_0)\circ \mathsf{and}\circ\sigma_2(\varepsilon);\ \lambda\mathscr{W}\lambda\mathscr{V}.\mathscr{V}\sqcap\mathscr{W};\ (\mathrm{S}|\mathrm{TV})|(\mathrm{S}|\mathrm{TV})|(\mathrm{S}|\mathrm{TV}) \quad \dfrac{\vdots \quad \vdots}{\lambda\varphi_1.\mathsf{robin}\circ\varphi_1\circ\mathsf{a}\circ\mathsf{book};\ \lambda P.\exists_{\mathbf{book}}(\lambda x.P(x)(\mathbf{r}));\ \mathrm{S}|\mathrm{TV}}}{\lambda\sigma_1\lambda\varphi_0.\sigma_1(\varphi_0)\circ\mathsf{and}\circ\mathsf{robin}\circ\varepsilon\circ\mathsf{a}\circ\mathsf{book};\ \lambda\mathscr{V}.\mathscr{V}\sqcap\lambda P.\exists_{\mathbf{book}}(\lambda x.P(x)(\mathbf{r}));\ (\mathrm{S}|\mathrm{TV})|(\mathrm{S}|\mathrm{TV})}}{\begin{array}{l}\mathsf{bought};\\ \mathbf{buy};\\ \mathrm{TV}\end{array} \quad \begin{array}{l}\lambda\varphi_0[\mathsf{leslie}\circ\varphi_0\circ\mathsf{a}\circ\mathsf{CD}\circ\mathsf{and}\circ\mathsf{robin}\circ\varepsilon\circ\mathsf{a}\circ\mathsf{book}];\\ \lambda Q.\exists_{\mathbf{CD}}(\lambda y.Q(y)(\mathbf{l}))\sqcap\lambda P.\exists_{\mathbf{book}}(\lambda x.P(x)(\mathbf{r}));\mathrm{S}|\mathrm{TV}\end{array}}$$

$$\mathsf{leslie}\circ\mathsf{bought}\circ\mathsf{a}\circ\mathsf{CD}\circ\mathsf{and}\circ\mathsf{robin}\circ\varepsilon\circ\mathsf{a}\circ\mathsf{book};\ \exists_{\mathbf{CD}}(\lambda y.\mathbf{buy}(y)(\mathbf{l}))\wedge\exists_{\mathbf{book}}(\lambda x.\mathbf{buy}(x)(\mathbf{r}));\mathrm{S}$$

In this analysis by Kubota and Levine, the role of both directional and nondirectional implication is crucial: the gapped sentence S|TV (= S|((NP\S)/NP)), which is associated with the functional phonology $\lambda\varphi_1.\mathsf{robin}\circ\varphi_1\circ\mathsf{a}\circ\mathsf{book}$, explicitly keeps track of the position of the medial gap via |, and, since what's missing is a transitive verb (i.e. (NP\S)/NP, indicating explicitly the directions in which it looks for its two arguments via / and \), the subject and the object appear in the right order in the string part of this functional phonology. Note in particular that in a uni-implication systems like ACG and Lambda Grammar, keeping track of the right word order becomes a virtually intractable problem.[2]

We omit the analysis of scope interactions between auxiliaries and Gapping, but the key idea should already be clear from the above analysis: the auxiliary wide scope interpretations for sentences like those in (7) fall out in this analysis since auxiliaries are introduced in the tectogrammatical derivation essentially in the same way as main verbs in (14) above, at a point after the coordinate structure is built. The structure of the derivation determines the relative scope between the auxiliary and coordination, thus, the former scopes over the latter.[3] The mismatch between the surface form of the sentence and the semantic scope is due to the morpho-syntactic requirement of the Gapping construction that the

[2] A reviewer expressed a concern that this analysis would overgenerate examples such as (†) **Larry thinks Sue is nice and Sue ~~thinks~~ Larry is funny* and (‡) **John gave a book to Mary and Peter ~~gave a book to Mary~~*. However, independent processing-oriented explanations exist for such examples. The difficulty of interpreting the NP NP V sequence in (†) without being led to a garden path by taking just the NP NP substring to be a gapped constituent can be dramatically ameliorated with an explicit complementizer (*. . . and Sue that Larry is funny*). For (‡), an alternative parse *John gave a book to [Mary and Peter]* seems to create a practically irrecoverable garden path effect. (‡) additionally violates a functional felicity conditions on Gapping which requires at least two contrasting elements in the two clauses (Kuno, 1976).

[3] However, as noted by Oehrle (1987), in at least some cases a distributive, auxiliary narrow-scope reading is available in such examples, and in order to derive this reading, it becomes necessary to reduce the prosodic type of the auxiliary from $(st\to st)\to st$ to st. This type of proof crucially requires inferences involving directional and non-directional slashes to interact with one another (Kubota and Levine, 2012, 2013). Similar reduction of a (phonologically) higher-order scopal operator to a lower type is required for licensing distributive readings for generalized quantifiers as well (Kubota and Levine, 2013) (which can be found in the Gapping context as well, as in *Chris set a problem for her logic exam, and Terry for his cell anatomy class*), providing further empirical evidence for the present hybrid implication system.

verb (or the auxiliary) be pronounced only once and within the first conjunct, as specified in the lexical entry for the Gapping-type conjunction in (13).

2.3 Split Scope in Hybrid TLCG

We propose that determiner gapping is just a special case of discontinuous gapping in which both the verb and the determiner are gapped. The negation wide scope is obtained for examples like (6a) since the negative determiner, being gapped, takes scope over the whole coordinate structure. Thus, the apparently anomalous scoping pattern is a predicted consequence of the analysis, much in the same way that the wide scope auxiliary in (7) is immediately predicted in Kubota and Levine's (2012) like-category coordination analysis of Gapping.

To formulate an explicit analysis, we need to work out the relevant details of the mechanism for split scope. We first illustrate a more or less direct implementation of the 'LF decomposition' analysis widely entertained in the literature. The key assumption of this approach is that negative quantifiers are semantically decomposed into two meaning components at the level of representation relevant for semantic interpretation. For example, *no* decomposes into an existential quantifier and sentential negation that scopes above it. The challenge is how to treat the interdependence between these two meaning components and make sure that they together realize as one morpheme *no* in the overt string.

We here propose to model the existential quantifier part via a prosodically empty operator which is constrained to occur in the scope of the overt negation morpheme *no*. To capture the interdependence between the covert existential and the overt negation, we posit the syntactic category $\mathrm{S}_{\mathrm{neg}}$, which designates a sentence containing the covert existential somewhere inside and which is waiting to be 'scoped over' by the overt negation *no*. Thus, the covert existential has the following lexical entry, which is identical to overt existential quantifiers except that it returns $\mathrm{S}_{\mathrm{neg}}$ instead of S. Phonologically, expressions with syntactic category $\mathrm{S}_{\mathrm{neg}}$ have a $st \rightarrow st$ phonology which keeps track of the 'gap' position that the higher negation morpheme *no* lowers into.

(15) $\lambda\varphi_1\lambda\sigma\lambda\varphi_2.\sigma(\varphi_2 \circ \varphi_1)$; $\lambda P.\exists_P$; $\mathrm{S}_{\mathrm{neg}}|(\mathrm{S}|\mathrm{NP})|\mathrm{N}$

The negation morpheme has the following lexical entry:

(16) $\lambda\sigma.\sigma(\mathsf{no})$; $\neg$; $\mathrm{S}|\mathrm{S}_{\mathrm{neg}}$

It takes a $\mathrm{S}_{\mathrm{neg}}$ as argument and returns an ordinary S. Semantically, it contributes sentential negation. Phonologically, it lowers the phonology *no* into the determiner 'gap' position introduced by the empty existential.

A simple sentence containing a negative quantifier is then analyzed as follows:

(17)

$$\dfrac{\lambda\sigma.\sigma(\mathsf{no});\ \neg;\ \mathrm{S}|\mathrm{S}_{\mathrm{neg}} \qquad \dfrac{\dfrac{\lambda\varphi_1\lambda\sigma\lambda\varphi_2.\sigma(\varphi_2 \circ \varphi_1);\ \mathsf{fish};}{\lambda P.\exists_P;\ \mathrm{S}_{\mathrm{neg}}|(\mathrm{S}|\mathrm{NP})|\mathrm{N}\quad \mathbf{fish};\ \mathrm{N}} \qquad \dfrac{\vdots \quad \vdots}{\lambda\varphi.\varphi \circ \mathsf{walks};\ \mathbf{walk};\ \mathrm{S}|\mathrm{NP}}}{\lambda\sigma\lambda\varphi_2.\sigma(\varphi_2 \circ \mathsf{fish});\ \exists_{\mathbf{fish}};\ \mathrm{S}_{\mathrm{neg}}|(\mathrm{S}|\mathrm{NP})}}{\mathsf{no} \circ \mathsf{fish} \circ \mathsf{walks};\ \neg\exists_{\mathbf{fish}}(\mathbf{walk});\ \mathrm{S}}$$

As shown here, the covert existential takes scope just like ordinary quantifiers do, but returns the category S_{neg} instead. The overt negation then takes this category as an argument to semantically scope over the whole sentence and prosodically lower itself into the determiner 'gap' position introduced by the covert existential. This yields the right pairing of surface form and interpretation, embodying the idea of split scope directly in the combinatoric structure.

With this analysis of split scope, the determiner gapping example (6a) can be analyzed as coordination of expressions of category $S_{neg}|TV$, that is, sentences in which both the verb and the overt negative determiner are missing. With hypothetical reasoning, deriving such an expression is straightforward, as in (18):

(18)

$$\dfrac{\dfrac{\lambda\varphi_1\lambda\sigma\lambda\varphi_2.\sigma(\varphi_2\circ\varphi_1);\ \mathsf{dog};\quad \lambda P.\exists_P;\ S_{neg}|(S|NP)|N\quad \mathbf{dog};\ N}{\lambda\sigma\lambda\varphi_2.\sigma(\varphi_2\circ\mathsf{dog});\quad \exists_{\mathbf{dog}};\ S_{neg}|(S|NP)} \qquad \dfrac{[\varphi_2;x;NP]^2 \qquad \dfrac{[\varphi_1;P;TV]^1 \qquad \mathsf{whiskas};\ \mathbf{w};\ NP}{\varphi_1\circ\mathsf{whiskas};\ P(\mathbf{w});\ VP}}{\varphi_2\circ\varphi_1\circ\mathsf{whiskas};\ P(\mathbf{w})(x);\ S}}{\dfrac{\lambda\varphi_2.\varphi_2\circ\varphi_1\circ\mathsf{whiskas};\ \lambda x.P(\mathbf{w})(x);\ S|NP}{\lambda\varphi_2.\varphi_2\circ\mathsf{dog}\circ\varphi_1\circ\mathsf{whiskas};\ \exists_{\mathbf{dog}}(\lambda x.P(\mathbf{w})(x));\ S_{neg}}}$$

$$\dfrac{\lambda\varphi_1\lambda\varphi_2.\varphi_2\circ\mathsf{dog}\circ\varphi_1\circ\mathsf{whiskas};\ \lambda P.\exists_{\mathbf{dog}}(\lambda x.P(\mathbf{w})(x));\ S_{neg}|TV}{}$$

Or then takes two such expressions as arguments and retains only the gap of the first conjunct. This is only a slight generalization of the Gapping-type conjunction introduced above. The rest of the derivation is shown in (19).

(19)

$$\dfrac{\dfrac{\lambda\varphi_1\lambda\varphi_2.\sigma(\varphi_2\circ\varphi_1);\ \mathsf{no};\quad \neg;\ S|S_{neg}}{\lambda\sigma.\sigma(\mathsf{no});\quad \neg;\ S|S_{neg}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\lambda\varphi_1\lambda\varphi_2.\varphi_2\circ\mathsf{dog}\circ\varphi_1\circ\mathsf{whiskas};\ \lambda P.\exists_{\mathbf{dog}}(P(\mathbf{w}));\ S_{neg}|TV} \qquad \dfrac{\dfrac{\vdots}{\lambda\varphi_1\lambda\varphi_2.\varphi_2\circ\mathsf{cat}\circ\varphi_1\circ\mathsf{alpo};\ \lambda P.\exists_{\mathbf{cat}}(P(\mathbf{a}));\ S_{neg}|TV}}{\lambda\sigma_2\lambda\sigma_1\lambda\varphi_1\lambda\varphi_2.\ \sigma_1(\varphi_1)(\varphi_2)\circ\mathsf{or}\circ\sigma_2(\varepsilon)(\varepsilon);\ \lambda V\lambda W.W\sqcup V;\ X|X|X}}{\lambda\sigma_1\lambda\varphi_1\lambda\varphi_2.\sigma_1(\varphi_1)(\varphi_2)\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ \lambda W.W\sqcup\lambda P.\exists_{\mathbf{cat}}(P(\mathbf{a}));\ (S_{neg}|TV)|(S_{neg}|TV)}}{\lambda\varphi_1\lambda\varphi_2.\varphi_2\circ\mathsf{dog}\circ\varphi_1\circ\mathsf{whiskas}\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ \lambda P.\exists_{\mathbf{dog}}(P(\mathbf{w}))\sqcup\lambda P.\exists_{\mathbf{cat}}(P(\mathbf{a}));\ S_{neg}|TV}}{\lambda\varphi_2.\varphi_2\circ\mathsf{dog}\circ\mathsf{eats}\circ\mathsf{whiskas}\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ \exists_{\mathbf{dog}}(\mathbf{eat}(\mathbf{w}))\vee\exists_{\mathbf{cat}}(\mathbf{eat}(\mathbf{a}));\ S_{neg}}$$

$$\dfrac{\mathsf{eats};\quad \mathbf{eat};\quad TV}{}$$

$$\mathsf{no}\circ\mathsf{dog}\circ\mathsf{eats}\circ\mathsf{whiskas}\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\quad \neg[\exists_{\mathbf{dog}}(\mathbf{eat}(\mathbf{w}))\vee\exists_{\mathbf{cat}}(\mathbf{eat}(\mathbf{a}))];\ S$$

After the whole coordinate structure is built, the verb and *no* are lowered into their respective positions in the first conjunct. Since the negative morpheme scopes over the whole coordinate structure in the tectogrammatical structure reflecting the combinatorial order, the negation wide scope reading is obtained.

2.4 Lexical Treatment of Split Scope via Type-Raised Quantifiers

The analysis of split scope above is a fairly straightforward implementation of the 'LF decomposition' analysis. It works fine and extends straightforwardly to the treatment of apparent scope anomaly in determiner gapping, but note that it involves some ad-hoc assumptions. An empty operator like the covert existential

posited above should be avoided if possible, and the newly introduced syntactic category S_{neg} is a purely diacritic device, having no motivation other than to control the distribution of the overt and covert operators that are stipulated to correlate with one another. Moreover, without S_{neg}, there is a straightforward one-to-one mapping between syntactic and prosodic types such that the prosodic type of any syntactic category is transparently reflected in the level of embedding involving the vertical slash (so, for example, any expression of syntactic type X|(Y|Z), with X–Z all atomic or involving only the directional slashes, is of type $(st \to st) \to st$). The syntactically atomic category S_{neg} disrupts this neat correspondence between syntactic and prosodic types, since, despite being syntactically atomic, it has a functional, $st \to st$ phonological type.

Eliminating these ad-hoc assumptions would thus be desirable, and it is indeed possible to do away with the diacritic syntactic category S_{neg}, by lexically encoding the two meaning components of negative quantifiers within a single entry. This involves specifying the scope of the higher negation and the lower existential separately within the lexical entry for the negative determiner, and requires treating the determiners forming negative quantifiers as lexically type-raised determiners. In the present setup, determiners take their nominal arguments to become quantifiers, thus they are of type S|(S|NP)|N. Negative determiners are lexically type-raised over S on this category, thus, by taking Det to abbreviate S|(S|NP)|N, they are of type S|(S|Det). Semantically, this lexically type-raised determiner feeds an ordinary positive quantifier meaning to its argument, thus saturating the determiner-type variable position of its argument, and additionally contributes sentential negation which scopes over the whole sentence.

Thus, by lexically type-raising the determiner, the separate scoping positions of the two meaning components of negative quantifiers can be encoded fully lexically. What remains to be worked out is the phonology of the higher order determiner. Since ordinary quantificational determiners are of type $st \to ((st \to st) \to st)$, the prosodic type of this type-raised determiner is $((st \to ((st \to st) \to st)) \to st) \to st$. In other words, the phonology of the type-raised determiner has to be specified in such a way that, by binding the prosodic variable of type $st \to ((st \to st) \to st)$ of ordinary determiners in the S|Det category that it takes as an argument, we obtain the right surface string in which the string phonology of the negative determiner appears in the right position. The right form of this higher-order phonology of a type-raised determiner can be inferred from the phonological term that is assigned to a syntactically type-raised ordinary determiner. This is shown in the following derivation, where a determiner whose phonology is built from the string c is type-raised to the syntactic category S|(S|Det), with the corresponding higher-order phonology:

(20)
$$\dfrac{\dfrac{\lambda\varphi\lambda\sigma.\sigma(\mathsf{c} \circ \varphi);\ \gamma;\ \mathrm{Det} \quad [\rho;\ \mathscr{P};\ \mathrm{S|Det}]^1}{\rho(\lambda\varphi\lambda\sigma.\sigma(\mathsf{c} \circ \varphi));\ \mathscr{P}(\gamma);\ \mathrm{S}}}{\lambda\rho.\rho(\lambda\varphi\lambda\sigma.\sigma(\mathsf{c} \circ \varphi));\ \lambda\mathscr{P}.\mathscr{P}(\gamma);\ \mathrm{S|(S|Det)}}$$

By replacing the string c with no, we obtain the right phonology for the negative determiner. Thus, putting together the phonology, semantics and the syntactic category of negative determiners, we have the following lexical entry:

(21) $\lambda\rho.\rho(\lambda\varphi\lambda\sigma.\sigma(\mathsf{no} \circ \varphi))$; $\lambda\mathscr{P}.\neg\mathscr{P}(\exists)$; S|(S|Det)

The derivation for a sentence with a negative quantifier then goes as follows:

(22)
$$\dfrac{\lambda\rho.\rho(\lambda\varphi\lambda\sigma.\sigma(\mathsf{no} \circ \varphi)); \lambda\mathscr{P}.\neg\mathscr{P}(\exists); \mathrm{S}|(\mathrm{S}|\mathrm{Det}) \qquad \dfrac{[\tau; \mathscr{F}; \mathrm{Det}]^1 \quad \mathsf{fish}; \mathbf{fish}; \mathrm{N}}{\tau(\mathsf{fish}); \mathscr{F}(\mathbf{fish}); \mathrm{S}|(\mathrm{S}|\mathrm{NP})} \quad \lambda\varphi.\varphi \circ \mathsf{walks}; \mathbf{walk}; \mathrm{S}|\mathrm{NP}}{\lambda\tau.\tau(\mathsf{fish})(\lambda\varphi.\varphi \circ \mathsf{walks}); \lambda\mathscr{F}.\mathscr{F}(\mathbf{fish})(\mathbf{walk}); \mathrm{S}|\mathrm{Det}} \qquad \tau(\mathsf{fish})(\lambda\varphi.\varphi \circ \mathsf{walks}); \mathscr{F}(\mathbf{fish})(\mathbf{walk}); \mathrm{S}}{\mathsf{no} \circ \mathsf{fish} \circ \mathsf{walks}; \neg\exists_{\mathbf{fish}}\mathbf{walk}; \mathrm{S}}$$

The derivation proceeds by first assuming a hypothetical determiner in the position that the negative determiner lowers into later. After the whole sentence is built, this hypothesis is bound and the resultant expression is of the right type to be given as an argument to the negative determiner. Note in particular that the right surface string is obtained by applying the higher-order functional phonology of the negative determiner to its argument, itself of a functional phonological type looking for a determiner phonology to return a string.

Just as in the analysis in the previous section, determiner gapping is then treated as a case of multiple gapping involving both the verb and the determiner. The only complication here is that the 'gap' corresponding to the determiner is of a higher-order type prosodically, so an identity element of this higher-order phonological type needs to be fed to the second conjunct. This 'empty determiner phonology' can be modelled on the phonology of ordinary determiners by replacing the string part of the phonological term with an empty string. Thus:

(23) $\varepsilon_{\mathsf{d}} =_{def} \lambda\varphi\lambda\sigma.\sigma(\varepsilon \circ \varphi) = \lambda\varphi\lambda\sigma.\sigma(\varphi)$

The lexical entry for the conjunction word can then be written as in (24), generalizing the Gapping-type conjunction entry to the S|Det|TV type:

(24) $\lambda\rho_2\lambda\rho_1\lambda\varphi\lambda\sigma.\rho_1(\varphi)(\sigma) \circ \mathsf{and} \circ \rho_2(\varepsilon)(\varepsilon_{\mathsf{d}})$; $\sqcap$; $\mathbf{GC}(\mathrm{S}|\mathrm{Det}|\mathrm{TV})$
where $\mathbf{GC}(A) = A|A|A$ for any syntactic type A

Expressions that are of the right type to be coordinated by this conjunction category can be derived via hypothetical reasoning in the usual way:

(25)
$$\dfrac{\dfrac{\dfrac{[\tau; \mathscr{F}; \mathrm{Det}]^3 \quad \mathsf{dog}; \mathbf{dog}; \mathrm{N}}{\tau(\mathsf{dog}); \mathscr{F}(\mathbf{dog}); \mathrm{S}|(\mathrm{S}|\mathrm{NP})} \quad \dfrac{[\varphi_2; x; \mathrm{NP}]^2 \qquad \dfrac{[\varphi_1; P; \mathrm{TV}]^1 \quad \mathsf{whiskas}; \mathbf{w}; \mathrm{NP}}{\varphi_1 \circ \mathsf{whiskas}; P(\mathbf{w}); \mathrm{VP}}}{\varphi_2 \circ \varphi_1 \circ \mathsf{whiskas}; P(\mathbf{w})(x); \mathrm{S}}}{\lambda\varphi_2.\varphi_2 \circ \varphi_1 \circ \mathsf{whiskas}; \lambda x.P(\mathbf{w})(x); \mathrm{S}|\mathrm{NP}}}{\tau(\mathsf{dog})(\lambda\varphi_2.\varphi_2 \circ \varphi_1 \circ \mathsf{whiskas}); \mathscr{F}(\mathbf{dog})(\lambda x.P(\mathbf{w})(x)); \mathrm{S}}}{\lambda\tau.\tau(\mathsf{dog})(\lambda\varphi_2.\varphi_2 \circ \varphi_1 \circ \mathsf{whiskas}); \lambda\mathscr{F}.\mathscr{F}(\mathbf{dog})(\lambda x.P(\mathbf{w})(x)); \mathrm{S}|\mathrm{Det}}}{\lambda\varphi_1\lambda\tau.\tau(\mathsf{dog})(\lambda\varphi_2.\varphi_2 \circ \varphi_1 \circ \mathsf{whiskas}); \lambda P\lambda\mathscr{F}.\mathscr{F}(\mathbf{dog})(\lambda x.P(\mathbf{w})(x)); \mathrm{S}|\mathrm{Det}|\mathrm{TV}}$$

This is then conjoined with another expression of the same type via the determiner-gapping conjunction in (24) to yield the following coordinated S|Det|TV:

(26)

$$\dfrac{\dfrac{\vdots\quad\vdots}{\lambda\varphi_1\lambda\tau.\ \tau(\mathsf{dog})(\lambda\varphi_2.\varphi_2\circ\varphi_1\circ\mathsf{whiskas});\ \lambda P\lambda\mathscr{F}.\mathscr{F}(\mathbf{dog})(\lambda x.P(\mathbf{w})(x));\ \mathrm{S|Det|TV}} \qquad \dfrac{\dfrac{\vdots\quad\vdots}{\lambda\varphi_1\lambda\tau.\ \tau(\mathsf{cat})(\lambda\varphi_2.\varphi_2\circ\varphi_1\circ\mathsf{alpo});\ \lambda P\lambda\mathscr{F}.\mathscr{F}(\mathbf{cat})(\lambda x.P(\mathbf{a})(x));\ \mathrm{S|Det|TV}}{\lambda\rho_2\lambda\rho_1\lambda\varphi\lambda\tau.\rho_1(\varphi)(\tau)\circ\ \mathsf{or}\circ\rho_2(\varepsilon)(\varepsilon_{\mathrm{d}});\ \sqcup;\ \mathbf{GC}(\mathrm{S|Det|TV})}}{\lambda\rho_1\lambda\varphi\lambda\tau.\rho_1(\varphi)(\tau)\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ \lambda\mathscr{W}.\mathscr{W}\sqcup\lambda P\lambda\mathscr{F}.\mathscr{F}(\mathbf{cat})(\lambda x.P(\mathbf{a})(x));\ (\mathrm{S|TV|Det})|(\mathrm{S|TV|Det})}$$

$$\lambda\varphi_1\lambda\tau.\tau(\mathsf{dog})(\lambda\varphi_2.\varphi_2\circ\varphi_1\circ\mathsf{whiskas})\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ \lambda P\lambda\mathscr{F}.\mathscr{F}(\mathbf{dog})(\lambda x.P(\mathbf{w})(x))\sqcup\lambda P\lambda\mathscr{F}.\mathscr{F}(\mathbf{cat})(\lambda x.P(\mathbf{a})(x));\mathrm{S|Det|TV}$$

Note in particular that the right string *cat alpo* is obtained for the second conjunct. This is a straightforward result of a couple of β-reduction steps:

(27) $\lambda\varphi\lambda\tau[\tau(\mathsf{cat})(\lambda\varphi'.\varphi'\circ\varphi\circ\mathsf{alpo})](\varepsilon)(\varepsilon_{\mathrm{d}}) = \lambda\varphi\lambda\sigma[\sigma(\varphi)](\mathsf{cat})(\lambda\varphi_2.\varphi_2\circ\varepsilon\circ\mathsf{alpo})$
$= \lambda\varphi_2[\varphi_2\circ\varepsilon\circ\mathsf{alpo}](\mathsf{cat}) = \mathsf{cat}\circ\varepsilon\circ\mathsf{alpo} = \mathsf{cat}\circ\mathsf{alpo}$

The rest of the derivation just involves combining the main verb and the negative determiner with this S|Det|TV expression.

(28)

$$\dfrac{\lambda\rho.\rho(\lambda\varphi\lambda\sigma.\ \sigma(\mathsf{no}\circ\varphi));\ \lambda\mathscr{P}.\neg\mathscr{P}(\exists);\ \mathrm{S|(S|Det)} \qquad \dfrac{\begin{array}{ll}\mathsf{eats};\ \mathbf{eat};\ \mathrm{TV} & \lambda\varphi_1\lambda\tau.\tau(\mathsf{dog})(\lambda\varphi_2.\varphi_2\circ\varphi_1\circ\mathsf{whiskas})\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\\ & \lambda P\lambda\mathscr{F}.\mathscr{F}(\mathbf{dog})(\lambda x.P(\mathbf{w})(x))\sqcup\\ & \lambda P\lambda\mathscr{F}.\mathscr{F}(\mathbf{cat})(\lambda x.P(\mathbf{a})(x));\mathrm{S|Det|TV}\end{array}}{\begin{array}{l}\lambda\tau.\tau(\mathsf{dog})(\lambda\varphi_2.\varphi_2\circ\mathsf{eats}\circ\mathsf{whiskas})\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\\ \lambda\mathscr{F}.\mathscr{F}(\mathbf{dog})(\lambda x.\mathbf{eat}(\mathbf{w})(x))\sqcup\lambda\mathscr{F}.\mathscr{F}(\mathbf{cat})(\lambda x.\mathbf{eat}(\mathbf{a})(x));\mathrm{S|Det}\end{array}}$$

$$\begin{array}{l}\mathsf{no}\circ\mathsf{dog}\circ\mathsf{eats}\circ\mathsf{whiskas}\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\\ \neg[\exists_{\mathbf{dog}}(\lambda x.\mathbf{eat}(\mathbf{w})(x))\vee\exists_{\mathbf{cat}}(\lambda x.\mathbf{eat}(\mathbf{a})(x))];\mathrm{S}\end{array}$$

Crucially, just as in the analysis from the previous section, since the negative determiner scopes over the whole coordinated gapped sentence in this tectogrammatical derivation, the right semantic scope between the two operators is predicted. Thus, here again, the apparently anomalous scope relation between the negative quantifier and disjunction is a predicted consequence of the 'gapped' status of the former. The syntactic analysis of gapping requires the determiner to syntactically scope over the whole coordinate structure in the combinatoric structure, and the semantic scope between the two transparently reflects this underlying structural relationship.

3 Comparative Subdeletion

We now turn to comparative subdeletion, illustrated in (29):

(29) John ate more donuts than Mary bought bagels.

This construction is similar to Gapping in that there is apparent deletion of some material in one of the two clauses involved: a determiner is missing in the *than* clause in a position where *more* appears in the main clause.

In the early literature of transformational grammar, there was a debate as to whether comparative subdeletion involves ellipsis or *wh*-movement. We here assume, following Hendriks (1995), that comparative subdeletion is in fact neither *wh*-movement nor deletion, but is rather to be analyzed along lines similar to the treatment of split gapping above. The primary motivation for this analysis comes from the fact that it yields the right compositional semantics immediately. To see this, note that what the comparative subdeletion sentence (29) compares is sizes of the sets $\{x : \mathbf{donut}(x) \wedge \mathbf{eat}(x)(\mathbf{j})\}$ and $\{x : \mathbf{bagel}(x) \wedge \mathbf{buy}(x)(\mathbf{m})\}$. Such sets can be obtained by abstracting over the determiner positions in the two clauses and supplying some appropriate operator ($\lambda P \lambda Q \lambda x[P(x) \wedge Q(x)]$) in that semantic argument position. On the prosodic side, *more-than* fills in an empty determiner phonology and the type-raised string *more* in the determiner-type gap positions of the two clauses and concatenates them with the string *than* in between. Thus, the lexical entry for *more-than* can be formulated as follows:

(30) $\lambda\rho_1\lambda\rho_2.\rho_2(\lambda\varphi\lambda\sigma.\sigma(\mathsf{more} \circ \varphi)) \circ \mathsf{than} \circ \rho_1(\varepsilon_\mathsf{d}); \mathbf{more\text{-}than};$
S|(S|Det)|(S|Det)

where the constant **more-than** stands for the following logical term:

(31) $\lambda\mathscr{F}\lambda\mathscr{G}.|\mathscr{G}(\lambda P\lambda Q\lambda x[P(x) \wedge Q(x)])| > |\mathscr{F}(\lambda P\lambda Q\lambda x[P(x) \wedge Q(x)])|$

Note here that since the determiner-type gap involves a higher-order prosodic variable in the present approach, the same identity element that fills in that gap in the second conjunct of determiner gapping is involved in 'closing off' the gap position of the *than* clause, and the phonology of *more* is identical in form to the type-raised determiner *no* from the previous section.

With this lexical entry for *more-than*, the derivation for (29) goes as follows:

(32)

$\lambda\rho_1\lambda\rho_2.\rho_2(\lambda\varphi\lambda\sigma.$ $\sigma(\mathsf{more} \circ \varphi)) \circ$ $\mathsf{than} \circ \rho_1(\varepsilon_\mathsf{d})$; **more-than**; S|(S|Det)|(S|Det)	⋮ ⋮	⋮ ⋮
	$\lambda\tau.\tau(\mathsf{bagels})(\lambda\varphi.\mathsf{mary} \circ \mathsf{bought} \circ \varphi)$; $\lambda\mathscr{F}.\mathscr{F}(\mathbf{bagel})(\lambda x.\mathbf{buy}(x)(\mathbf{m}))$; S|Det	$\lambda\tau.\tau(\mathsf{donut})$ $(\lambda\varphi.\mathsf{john} \circ \mathsf{ate} \circ \varphi)$; $\lambda\mathscr{F}.\mathscr{F}(\mathbf{donut})$ $(\lambda x.\mathbf{eat}(x)(\mathbf{j}))$; S|Det
$\lambda\rho_2.\rho_2(\lambda\varphi\lambda\sigma.\sigma(\mathsf{more} \circ \varphi)) \circ \mathsf{than} \circ \mathsf{mary} \circ \mathsf{bought} \circ \mathsf{bagels}$; **more-than**$(\lambda\mathscr{F}.\mathscr{F}(\mathbf{bagel})(\lambda x.\mathbf{buy}(x)(\mathbf{m})))$; S|(S|Det)		
$\mathsf{john} \circ \mathsf{ate} \circ \mathsf{more} \circ \mathsf{donuts} \circ \mathsf{than} \circ \mathsf{mary} \circ \mathsf{bought} \circ \mathsf{bagels}$; **more-than**$(\lambda\mathscr{F}.\mathscr{F}(\mathbf{bagel})(\lambda x.\mathbf{buy}(x)(\mathbf{m})))(\lambda\mathscr{F}.\mathscr{F}(\mathbf{donut})(\lambda x.\mathbf{eat}(x)(\mathbf{j})))$; S		

The final translation can be unpacked as:

(33) $|\{x : \mathbf{donut}(x) \wedge \mathbf{eat}(x)(\mathbf{j})\}| > |\{x : \mathbf{bagel}(x) \wedge \mathbf{buy}(x)(\mathbf{m})\}|$

We finish our discussion with a somewhat complex interaction between the two phenomena we have analyzed above, exemplified by (34):

(34) No dog eats more whiskas than Leslie buys pizza, or cat alpo.

This sentence has an interpretation which can be paraphrased as 'No dog eats more whiskas than Leslie buys pizza *and* no cat eats more alpo than Leslie buys pizza'. That is, it involves determiner gapping where, together with the determiner and the main verb, the discontinuous constituent *more . . . than Leslie buys pizza* (of syntactic type S|(S|Det)) is gapped from the second conjunct. The analysis is straightforward, the only complication being that this example involves abstracting over a yet higher-order type category (of type S|(S|Det)) for the discontinuous type-raised quantifier *more . . . than Leslie buys pizza* in the two conjuncts. The identity element of this higher type to be fed to the second conjunct ($\varepsilon_{\mathsf{d}\uparrow} =_{def} \lambda\delta.\delta(\lambda\varphi\lambda\sigma.\sigma(\varphi))$) can be obtained by simply type-raising the determiner-type identity element ε_{d} over S. The derivation is given in (35).

(35)

$$\dfrac{\dfrac{\dfrac{\vdots\quad\vdots}{\lambda\varphi_0\lambda\omega\lambda\rho_1.\omega(\lambda\rho_2.\rho_2(\mathsf{whiskas})(\lambda\varphi_2.\rho_1(\mathsf{dog})(\lambda\varphi_1.\varphi_1\circ\varphi_0\circ\varphi_2)));\ \mathrm{S}|\mathrm{Det}|(\mathrm{S}|(\mathrm{S}|\mathrm{Det}))|\mathrm{TV}} \qquad \dfrac{\lambda\zeta_1\lambda\zeta_2\lambda\varphi\lambda\omega\lambda\rho.\zeta_2(\varphi)(\omega)(\rho)\circ \mathsf{or}\circ\zeta_1(\varepsilon)(\varepsilon_{\mathsf{d}\uparrow})(\varepsilon_{\mathsf{d}});\ \mathbf{GC}(\mathrm{S}|\mathrm{Det}|(\mathrm{S}|(\mathrm{S}|\mathrm{Det}))|\mathrm{TV}) \qquad \dfrac{\vdots\quad\vdots}{\lambda\varphi_0\lambda\omega\lambda\rho_1.\omega(\lambda\rho_2.\rho_2(\mathsf{alpo})(\lambda\varphi_2.\rho_1(\mathsf{cat})(\lambda\varphi_1.\varphi_1\circ\varphi_0\circ\varphi_2)));\ \mathrm{S}|\mathrm{Det}|(\mathrm{S}|(\mathrm{S}|\mathrm{Det}))|\mathrm{TV}}{\lambda\zeta_2\lambda\varphi\lambda\omega\lambda\rho.\zeta_2(\varphi)(\omega)(\rho)\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ (\mathrm{S}|\mathrm{Det}|(\mathrm{S}|(\mathrm{S}|\mathrm{Det}))|\mathrm{TV})|(\mathrm{S}|\mathrm{Det}|(\mathrm{S}|(\mathrm{S}|\mathrm{Det}))|\mathrm{TV})}}{\lambda\varphi\lambda\omega\lambda\rho.\omega(\lambda\rho_2.\rho_2(\mathsf{whiskas})(\lambda\varphi_2.\rho(\mathsf{dog})(\lambda\varphi_1.\varphi_1\circ\varphi\circ\varphi_2)))\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ \mathrm{S}|\mathrm{Det}|(\mathrm{S}|(\mathrm{S}|\mathrm{Det}))|\mathrm{TV}}$$

$$\dfrac{\lambda\rho.\rho(\lambda\varphi\lambda\sigma.\sigma(\mathsf{no}\circ\varphi));\ \mathrm{S}|(\mathrm{S}|\mathrm{Det}) \qquad \dfrac{\dfrac{\vdots\quad\vdots}{\lambda\rho_2.\rho_2(\lambda\varphi\lambda\sigma.\sigma(\mathsf{more}\circ\varphi))\circ \mathsf{than}\circ\mathsf{leslie}\circ\mathsf{buys}\circ\mathsf{pizza};\ \mathrm{S}|(\mathrm{S}|\mathrm{Det})} \qquad \dfrac{\dfrac{\mathsf{eats};\ \mathbf{eat};\ \mathrm{TV}}{\lambda\varphi\lambda\omega\lambda\rho.\omega(\lambda\rho_2.\rho_2(\mathsf{whiskas})(\lambda\varphi_2.\rho(\mathsf{dog})(\lambda\varphi_1.\varphi_1\circ\varphi\circ\varphi_2)))\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ \mathrm{S}|\mathrm{Det}|(\mathrm{S}|(\mathrm{S}|\mathrm{Det}))|\mathrm{TV}}\ \dfrac{\vdots\quad\vdots}{\lambda\omega\lambda\rho.\omega(\lambda\rho_2.\rho_2(\mathsf{whiskas})(\lambda\varphi_2.\rho(\mathsf{dog})(\lambda\varphi_1.\varphi_1\circ\mathsf{eats}\circ\varphi_2)))\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ \mathrm{S}|\mathrm{Det}|(\mathrm{S}|(\mathrm{S}|\mathrm{Det}))}}{\lambda\tau_1.\tau_1(\mathsf{dog})(\lambda\varphi_1.\varphi_1\circ\mathsf{eats}\circ\mathsf{more}\circ\mathsf{whiskas})\circ\mathsf{than}\circ\mathsf{leslie}\circ\mathsf{buys}\circ\mathsf{pizza}\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ \mathrm{S}|\mathrm{Det}}}{\mathsf{no}\circ\mathsf{dog}\circ\mathsf{eats}\circ\mathsf{more}\circ\mathsf{whiskas}\circ\mathsf{than}\circ\mathsf{leslie}\circ\mathsf{buys}\circ\mathsf{pizza}\circ\mathsf{or}\circ\mathsf{cat}\circ\mathsf{alpo};\ \mathrm{S}}$$

4 Conclusion

Two related conclusions emerge from the above discussion. First, if we adopt the analysis of quantifier scope due to Oehrle (1994) in which quantificational determiners are treated as higher-order functors prosodically, then the empirical phenomena considered in this paper show that we need to recognize not just functors of higher-order prosodic types but also variables ranging over such higher-order functors. So far as we are aware, this is the first time that the need for such higher-order prosodic variables has been noted in the literature. This

obviously raises the issue of how much complexity is needed in this domain, a question which we have to leave for another occasion.

Another, related point pertains to a comparison of the present proposal with related approaches to discontinuity. The most recent and well-developed framework for dealing with discontinuity within CG is the Displacement Calculus of Morrill et al. (2011) (which builds on the previous proposals by Morrill and Solias (1993); Morrill (1994), etc.). Though our Hybrid TLCG resembles Morrill et al.'s calculus in that both recognize directional slashes and non-directional syntactic connectives for dealing with discontinuity, there is one important difference between the two. In Hybrid TLCG, there is only one 'discontinuous' connective | (tied to lambda binding in phonology), whereas Morrill et al. recognize two counterparts of |, namely, ↑ and ↓, which respectively produce functors that wrap around their arguments and functors that are wrapped around by their arguments. With the distinction of these two syntactic connectives, certain aspects of the analysis can be simplified. Most notably, in the prosodic component, the only extension from the Lambek calculus is that 'separators' that keep track of gap positions are recognized as distinguished objects in the string algebra. Thus, in Morrill et al.'s calculus, no higher-order, functional entities are recognized in the prosodic component unlike in the Oehrle-style approach. Quantifiers and quantificational determiners are associated with strings, and their prosodic behaviors are encoded in the syntactic categories involving ↑ and ↓. Thus, in their approach, determiner gapping can simply be treated by abstracting over strings, without the complication of the higher-order treatment along lines we described above which is necessitated in the Oehrle-style treatment.

One might take this to be an advantage of the Morrill-style approach, but we believe that facts that bear on the comparison between the two types of approach come from more complex interactions between phenomena displaying discontinuity of the sort we sketched at the end of the previous section. The present approach, with a fully general lambda calculus in the prosodic component, straightforwardly extends to cases in which what is missing in a discontinuous constituent is itself a complex discontinuous constituent. By contrast, in the Morrill-style setup, there does not seem to be any straightforward way of treating the discontinuity exhibited by the gapped *more* + *than* S constituent in (34). Being of type S|(S|Det), this expression takes a determiner-gapped sentence and fills in the determiner *more* in the gap and concatenates the *than* clause to the resultant string. A lambda term for such a phonological functor is straightforward to write in the present approach, but in Morrill et al.'s setup, each functor is either a 'wrapper' or a 'wrappee', so a single expression cannot be both at the same time.[4] It thus seems reasonable to conclude that the present proposal

[4] There are of course ways around this problem, by mediating the interdependence between *more* and *than* via some syntactic mechanism (as indeed proposed by Morrill et al. (2011)). But such a solution seems to miss the point that the *more* + *than* clause in comparative subdeletion manifests discontinuous constituency.

offers the most general and empirically successful approach to discontinuity in the current CG literature.[5]

References

Barker, C.: Parasitic scope. Linguistics and Philosophy 30, 407–444 (2007)
Boyer, C.: The History of the Calculus and its Conceptual Development. Dover, New York (1949)
de Groote, P.: Towards abstract categorial grammars. In: Proceedings of ACL 39, pp. 148–155 (2001)
Hendriks, P.: Ellipsis and multimodal categorial type logic. In: Morrill, G.V., Oehrle, R.T. (eds.) Proceedings of Formal Grammar 1995, pp. 107–122 (1995)
Jacobs, J.: Lexical decomposition in Montague grammar. Theoretical Linguistics 7(1/2), 121–136 (1980)
Johnson, K.: Few dogs eat Whiskers or cats Alpo. In: Kusumoto, K., Villalta, E. (eds.) UMOP 23, pp. 47–60. GLSA, Amherst (2000)
Kubota, Y., Levine, R.: Gapping as like-category coordination. In: Béchet, D., Dikovsky, A. (eds.) LACL 2012. LNCS, vol. 7351, pp. 135–150. Springer, Heidelberg (2012)
Kubota, Y., Levine, R.: Against ellipsis: Arguments for the direct licensing of 'non-canonical' coordinations. MS., OSU (2013)
Kuno, S.: Gapping: A functional analysis. Linguistic Inquiry 7, 300–318 (1976)
Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958)
McCawley, J.D.: Gapping with shared operators. In: Peterson, D.A. (ed.) Berkeley Linguistics Society, pp. 245–253. University of California, Berkeley (1993)
Montague, R.: The proper treatment of quantification in ordinary English. In: Hintikka, J., Moravcsik, J.M., Suppes, P. (eds.) Approaches to Natural Language, pp. 221–242. D. Reidel, Dordrecht (1973)
Morrill, G.: Type Logical Grammar. Kluwer, Dordrecht (1994)
Morrill, G., Solias, T.: Tuples, discontinuity, and gapping in categorial grammar. In: Proceedings of EACL 6, pp. 287–296. ACL, Morristown (1993)
Morrill, G., Valentín, O., Fadda, M.: The displacement calculus. Journal of Logic, Language and Information 20, 1–48 (2011)

[5] We acknowledge here that there remains an important theoretical issue: the formal properties of our hybrid implication logic are currently unknown. But note that in the domain of empirical science (including linguistics), empirical considerations should always take precedence over purely formal issues. In this connection, the point made by Boyer in his detailed history of calculus is, we think, particularly relevant:

> Perhaps the most manifest deterring force [for the development of calculus] was the rigid insistence on the exclusion from mathematics of any idea not at the time allowing of strict logical interpretation ... it is clear that the indiscriminate use of methods and ideas which are without logical foundation is not to be condoned ... but pending the final establishment of this, the banishment of suggestive views is a serious mistake. (Boyer, 1949, 301–302)

Our system, so far as we can tell, makes systematic predictions which correspond to the empirically observed patterns exactly, and hence seems to achieve a level of 'suggestiveness' which entitles it to further investigation of its logical foundations.

Muskens, R.: Language, lambdas, and logic. In: Kruijff, G.-J., Oehrle, R. (eds.) Resource Sensitivity in Binding and Anaphora, pp. 23–54. Kluwer (2003)
Oehrle, R.T.: Boolean properties in the analysis of gapping. In: Huck, G.J., Ojeda, A.E. (eds.) Discontinuous Constituency, pp. 203–240. Academic Press (1987)
Oehrle, R.T.: Term-labeled categorial type systems. Linguistics and Philosophy 17(6), 633–678 (1994)
Penka, D.: Negative Indefinites. Oxford University Press, Oxford (2011)
Pollard, C.: Proof theoretic background for Linear Grammar. MS., OSU (2011)
Pollard, C., Allyn Smith, E.: A unified analysis of *the same*, phrasal comparatives and superlatives. In: Proceedings of SALT 2012, pp. 307–325 (2012)
Siegel, M.E.A.: Gapping and interpretation. Linguistic Inquiry 15, 523–530 (1984)

Conjunctive Grammars in Greibach Normal Form and the Lambek Calculus with Additive Connectives

Stepan Kuznetsov

Moscow State University
sk@lpcs.math.msu.su

Abstract. We prove that any language without the empty word, generated by a conjunctive grammar in Greibach normal form, is generated by a grammar based on the Lambek calculus enriched with additive ("intersection" and "union") connectives.

1 Conjunctive Grammars

Let Σ be an arbitrary finite alphabet, Σ^* is the set of all words, and Σ^+ is the set of all non-empty words over Σ.

We consider a generalisation of context-free grammars, introduced by Okhotin [9] (and earlier by Szabari [14]).

A *conjunctive grammar* is a quadruple $\mathcal{G} = \langle \Sigma, N, \mathcal{P}, S \rangle$, where Σ and N are two non-intersecting alphabets (Σ is the alphabet in which the language is being defined, its elements are called *terminal symbols*, and N is an auxiliary alphabet, consisting of *nonterminal symbols), $S \in N$* (the *start symbol)*, and $\mathcal{P}$ is a finite set of *rules* of the form

$$A \to \beta_1 \& \ldots \& \beta_m,$$

where $A \in N$, $m \geqslant 1$, $\beta_1, \ldots, \beta_m \in (\Sigma \cup N)^*$.

We define the language generated by this grammar in terms of a formal deduction system associated with the grammar [10]. This formal system derives pairs of the form $[X, w]$, where $X \in \Sigma \cup N$ and $w \in \Sigma^*$. Axioms are pairs $[a, a]$, for all $a \in \Sigma$, and for every rule $A \to B_{11} \ldots B_{1m_1} \& \ldots \& B_{k1} \ldots B_{km_k} \in \mathcal{P}$, $B_{ji} \in \Sigma \cup N$, and for all strings $u_{ji} \in \Sigma^*$, $j \in \{1, \ldots, k\}$, $i \in \{1, \ldots, m_j\}$, that satisfy $u_{11} \ldots u_{1m_1} = \ldots = u_{k1} \ldots u_{km_k} = w$, there is a deduction rule

$$\frac{[B_{11}, u_{11}] \quad \ldots \quad [B_{km_k}, u_{km_k}]}{[A, w]}.$$

The formal system, associated with the grammar $\mathcal{G}$, is also denoted by $\mathcal{G}$. Define $\mathfrak{L}_{\mathcal{G}}(X) \rightleftharpoons \{w \mid \mathcal{G} \vdash [X, w]\}$ and $\mathfrak{L}(\mathcal{G}) \rightleftharpoons \mathfrak{L}_{\mathcal{G}}(S)$ ("$\rightleftharpoons$" here and further means "equals by definition"). $\mathfrak{L}(\mathcal{G})$ is the *language generated by* $\mathcal{G}$.

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 242–249, 2013.

Example 1. Consider the following conjunctive grammar (here small letters stand for terminal symbols, capital stand for nonterminal ones; S is the start symbol):

$$S \to aAB \,\&\, aDC$$
$$A \to aA$$
$$A \to a$$
$$B \to bBc$$
$$B \to b$$
$$C \to cC$$
$$C \to c$$
$$D \to aDb$$
$$D \to b$$

This grammar generates the language $\{a^{n+1}b^{n+1}c^n \mid n \geq 1\}$ as an intersection of two context-free languages. For example, the word $aaabbbcc = a^3b^3c^2$ is generated in the following way: first we derive $[S, aaabbbcc]$ from $[a,a]$, $[A, aa]$, $[B, bbbcc]$, $[a,a]$, $[D, aabbb]$, and $[C, cc]$. The pair $[a,a]$ is an axiom; the others are derived as follows:

$$\dfrac{[a,a] \quad \dfrac{[a,a]}{[A,aa]}}{[A,aa]} \qquad \dfrac{[b,b] \quad \dfrac{[b,b] \quad \dfrac{[b,b]}{[B,b]} \quad [c,c]}{[B,bbc]} \quad [c,c]}{[B,bbbcc]}$$

$$\dfrac{[a,a] \quad \dfrac{[a,a] \quad \dfrac{[b,b]}{[D,b]} \quad [b,b]}{[D,abb]} \quad [b,b]}{[D,aabbb]} \qquad \dfrac{[c,c] \quad \dfrac{[c,c]}{[C,c]}}{[C,cc]}$$

For technical reasons we also consider an enlarged version of this deduction system, called $\mathcal{G}_{\text{cut}}$. We allow nonterminal symbols to appear in the second components of the pairs (derivable objects in it are of the form $[X, \omega]$, where $X \in \Sigma \cup N$ and $\omega \in (\Sigma \cup N)^*$) and add new axioms $[A, A]$ for all $A \in N$ and the cut rule:

$$\dfrac{[B, \tau] \quad [A, \omega_1 B \omega_2]}{[A, \omega_1 \tau \omega_2]}.$$

A trivial "cut elimination theorem" holds:

Lemma 1. *If $A \in N \cup \Sigma$, $w \in \Sigma^*$, then $\mathcal{G}_{\text{cut}} \vdash [A, w]$ if and only if $\mathcal{G} \vdash [A, w]$.*

Proof. The "if" part is obvious. For the "only if" part, we prove that every pair, derivable in $\mathcal{G}_{\text{cut}}$, is derivable without applying the cut rule (therefore, as w does not contain nonterminal symbols, they do not occur in the derivation, thus this derivation is valid in the original system). Let $[B, \tau]$ and $[A, \omega_1 B \omega_2]$ be

derivable without applying the cut rule. Prove that $[A, \omega_1\tau\omega_2]$ also has a cut-free proof. Proceed by induction on the derivation of $[A, \omega_1 B\omega_2]$. If it is an axiom, then ω_1 and ω_2 is empty, $B = A$, and our goal coincides with the left premise, $[B, \tau]$. If $[A, \omega_1 B\omega_2]$ is derived using an inference rule, then we can perform the substitution of τ for B in the premises of this rule, and apply the induction hypothesis.

2 Greibach Normal Form

Consider only languages without the empty word.

A conjunctive grammar is in *Greibach normal form* (a generalisation of Greibach normal form for context-free grammars [3]), if all the rules are of the form $A \to a\beta_1 \& \ldots \& a\beta_k$, $a \in \Sigma$, $\beta_j \in N^+$ or of the form $A \to a$, $a \in \Sigma$.

The question remains open, whether every conjunctive grammar can be transformed into this form. However, it is true for languages over the one-letter alphabet, as shown by Okhotin and Reitwießner [11]. Therefore, conjunctive grammars in Greibach normal form can capture some languages that are not context-free or even finite intersections of those, since the language $\{a^{4^n} \mid n \geq 1\}$ is generated by a conjunctive grammar found by Jeż [4].

Example 2. The grammar from Example 1 can be easily transformed into Greibach normal form:

$$
\begin{aligned}
S &\to aAB \,\&\, aDC\\
A &\to aA\\
A &\to a\\
B &\to bBU\\
B &\to b\\
U &\to c\\
C &\to cC\\
C &\to c\\
D &\to aDV\\
D &\to b\\
V &\to b
\end{aligned}
$$

3 Multiplicative-Additive Lambek Calculus

In this section we define an extension of the Lambek calculus (introduced in [7]) with two new connectives, *additive conjunction* and *disjunction*. The additive (intersective) conjunction was already introduced by Lambek [8], and the whole calculus was considered by Kanazawa [5]. We shall call this calculus **MALC**, as in [6], but use the Lambek-style notation for connectives.

A countable set $\mathrm{Pr} = \{p_1, p_2, p_3, \ldots\}$ is called the set of *primitive types. Types* of **MALC** are built from primitive types with five binary connectives: $\cdot$ (multiplication, product conjunction), $\backslash$ (left division), / (right division), $\cap$ (intersection,

additive conjunction), $\cup$ (union, additive disjunction). We denote types with capital Latin letters and their finite sequences (possibly empty) with capital Greek ones; Λ stands for the empty sequence. *Sequents* (derivable objects) of **MALC** are of the form $\Pi \to C$.

Axioms: $A \to A$.

Rules of inference:

$$\frac{A\,\Pi \to B}{\Pi \to A \setminus B}\ (\to \setminus),\ \Pi \neq \Lambda; \qquad \frac{\Pi \to A \quad \Gamma\,B\,\Delta \to C}{\Gamma\,\Pi\,(A \setminus B)\,\Delta \to C}\ (\setminus \to);$$

$$\frac{\Pi\,A \to B}{\Pi \to B \,/\, A}\ (\to /),\ \Pi \neq \Lambda; \qquad \frac{\Pi \to A \quad \Gamma\,B\,\Delta \to C}{\Gamma\,(B \,/\, A)\,\Pi\,\Delta \to C}\ (/ \to);$$

$$\frac{\Gamma \to A \quad \Delta \to B}{\Gamma\,\Delta \to A \cdot B}\ (\to \cdot); \qquad \frac{\Gamma\,A\,B\,\Delta \to C}{\Gamma\,(A \cdot B)\,\Delta \to C}\ (\cdot \to);$$

$$\frac{\Gamma \to A_1 \quad \Gamma \to A_2}{\Gamma \to A_1 \cap A_2}\ (\to \cap); \qquad \frac{\Gamma\,A_i\,\Delta \to C}{\Gamma\,(A_1 \cap A_2)\,\Delta \to C}\ (\cap \to)_i, i = 1, 2;$$

$$\frac{\Gamma \to A_i}{\Gamma \to A_1 \cup A_2}\ (\to \cup)_i, i = 1, 2; \qquad \frac{\Gamma\,A_1\,\Delta \to C \quad \Gamma\,A_2\,\Delta \to C}{\Gamma\,(A_1 \cup A_2)\,\Delta \to C}\ (\cup \to);$$

$$\frac{\Pi \to A \quad \Gamma\,A\,\Delta \to C}{\Gamma\,\Pi\,\Delta \to C}\ (\text{cut}).$$

The cut rule is eliminable using the standard technique [7].

The fragment without $\cap$ and $\cup$ is the ordinary (multiplicative) Lambek calculus, called **MLC** or **L**. We also consider fragments of **MALC** with other restrictions of the set of connectives: **MALC**$(/, \cap)$, **MALC**$(/, \cdot, \cap)$, **MLC**$(/)$.

4 Categorial Grammars

A **MALC**-*grammar* is a triple $\mathscr{G} = \langle \Sigma, H, \rhd \rangle$, where Σ is a finite alphabet, $H \in \mathrm{Tp}$, and $\rhd$ is a finite correspondence between Tp and Σ ($\rhd \subset \mathrm{Tp} \times \Sigma$). The *language generated by* $\mathscr{G}$ is the set of all nonempty words $a_1 \dots a_n$ over Σ for which there exist types $B_1, \dots, B_n$ such that $\mathbf{MALC} \vdash B_1 \dots B_n \to H$ and $B_i \rhd a_i$ for all $i \in \{1, \dots, n\}$. We denote this language by $\mathfrak{L}(\mathscr{G})$.

The notions of **MALC**$(/, \cap)$-, **MALC**$(/, \cdot, \cap)$-, **MLC**-, and **MLC**$(/)$-grammar are defined similarly.

As shown by Gaifman [1] and Buszkowski [2], any context-free language without the empty word is generated by an **MLC**$(/)$-grammar. On the other hand, any language generated by an **MLC**-grammar is context-free (Pentus [12]).

Kanazawa [5] proved that any finite intersection of context-free languages is generated by a **MALC**$(/, \cap)$-grammar (therefore such grammars go beyond context-free). No generalisation of Pentus' theorem for **MALC** is yet known.

Theorem 1. *If a language without the empty word is generated by a conjunctive grammar in Greibach normal form, then this language is generated by a* **MALC**$(/, \cdot, \cap)$*-grammar.*

5 The Construction

Given a conjunctive grammar $\mathcal{G} = \langle N, \Sigma, \mathcal{P}, S\rangle$ in Greibach normal form, we shall construct a $\mathbf{MALC}(/, \cdot, \cap)$-grammar $\mathscr{G}$, such that $\mathfrak{L}(\mathscr{G}) = \mathfrak{L}(\mathcal{G})$.

In order to avoid notation collisions, further we shall use the following naming convention (all these letters can also be decorated with numerical or other indices):

Letter	*Range*
A, B, S	N (nonterminal symbols of $\mathcal{G}$)
a	Σ (terminal symbols)
x	$N \cup \Sigma$
w, u	Σ^* (strings of terminal symbols)
β	N^+ (strings of nonterminal symbols)
τ, ω	$(N \cup \Sigma)^*$
p	Pr (primitive types of **MALC**)
E, F, G, P	Tp (types of **MALC**)
Γ, Φ, Ψ	Tp^* (sequences of types)

With every $A \in N$ we associate a distinguished primitive type p_A. For $\beta = B_1 \dots B_m$ let $P_\beta \leftrightharpoons p_{B_1} \cdot \ldots \cdot p_{B_m}$ (multiplication is associative, so we can omit the brackets).

Since intersection in **MALC** is commutative and associative, we can use intersections of nonempty sets of types, not bothering about order and brackets: $\bigcap_{j=1}^{k} E_j$ stands for $E_1 \cap \ldots \cap E_k$, and if $\mathcal{M} = \{E_1, \ldots, E_k\}$, then $\bigcap \mathcal{M} \leftrightharpoons E_1 \cap \ldots \cap E_k$. If $\mathcal{M} = \{E\}$, then $\bigcap \mathcal{M} \leftrightharpoons E$.

For every $a \in \Sigma$ let

$$\mathcal{M}_a \leftrightharpoons \{p_A \,/ \left(\bigcap_{j=1}^{k} P_{\beta_j}\right) \mid (A \to a\beta_1 \& \ldots \& a\beta_k) \in \mathcal{P}\} \cup \{p_A \mid (A \to a) \in \mathcal{P}\}.$$

Let $G_a \leftrightharpoons \bigcap \mathcal{M}_a$. For $A \in N$ let $G_A \leftrightharpoons p_A$. The following holds due to the $(\cap \to)$ rule:

Lemma 2. *If* $E \in \mathcal{M}_a$ *and* $\mathbf{MALC} \vdash \Phi\, E\, \Psi \to F$, *then* $\mathbf{MALC} \vdash \Phi\, G_a\, \Psi \to F$.

For $\omega = x_1 \dots x_n \in (N \cup \Sigma)^+$ let $\Gamma_\omega \leftrightharpoons G_{x_1} \dots G_{x_n}$.

Lemma 3. *If* $\mathcal{G} \vdash [A, w]$, *then* $\mathbf{MALC} \vdash \Gamma_w \to p_A$.

Proof. We proceed by induction on the length of w. The base case ($w = a$) corresponds to an application of a rule of the form $A \to a$ to the $[a, a]$ axiom (this is the only way to derive $[A, a]$). In this case we have $p_A \in \mathcal{M}_a$, therefore by Lemma 2 we get $\mathbf{MALC} \vdash G_a \to p_A$, and $\Gamma_w = G_a$.

Now let w contain at least two symbols and the last step of the derivation of $[A, w]$ be an application of the rule $A \to a\beta_1 \& \ldots \& a\beta_k$. Then $w = aw'$, and

for every $j \in \{1,\dots,k\}$, if $\beta_j = B_{j1}\dots B_{jm_j}$, then $w' = u_{j1}\dots u_{jm_j}$ and for every $i = \{1,\dots,m_j\}$ we have $\mathcal{G} \vdash [B_{ji}, u_{ji}]$. Therefore, by induction hypothesis, $\mathbf{MALC} \vdash \Gamma_{u_{ji}} \to p_{B_{ji}}$, whence $\mathbf{MALC} \vdash \Gamma_{w'} \to P_{\beta_j}$ for every j. Applying the $(\to \cap)$ rule k times we get

$$\mathbf{MALC} \vdash \Gamma_{w'} \to \bigcap_{j=1}^{k} P_{\beta_j},$$

and, finally, by $(/ \to)$,

$$\mathbf{MALC} \vdash p_A \,/ \left(\bigcap_{j=1}^{k} P_{\beta_j} \right) \Gamma_{w'} \to p_A.$$

Since $p_A \,/ (\bigcap_{j=1}^{k} P_{\beta_j}) \in \mathcal{M}_a$, by Lemma 2 we have $\mathbf{MALC} \vdash G_a\, \Gamma_{w'} \to p_A$, and $G_a\, \Gamma_{w'} = \Gamma_w$.

Before proving the inverse statement, we shall prove two technical lemmata:

Lemma 4. $\mathbf{MALC} \vdash \Phi \to \bigcap_{j=1}^{k} P_{\beta_j}$ *if and only if* $\mathbf{MALC} \vdash \Phi \to P_{\beta_j}$ *for every* $j \in \{1,\dots,k\}$.

Proof. The "if" part is just k applications of $(\to \cap)$. The "only if" part is proved using the cut rule (for every j_0):

$$\frac{\Gamma \to \bigcap_{j=1}^{k} P_{\beta_j} \quad \bigcap_{j=1}^{k} P_{\beta_j} \to P_{\beta_{j_0}}}{\Gamma \to P_{\beta_{j_0}}} \text{ (cut)}$$

Lemma 5. *If* $\omega \in (N \cup \Sigma)^+$, $\beta = B_1 \dots B_m \in N^+$, *and* $\mathbf{MALC} \vdash \Gamma_\omega \to P_\beta$, *then there exist such* $\tau_1,\dots,\tau_m \in (N \cup \Sigma)^+$, *that* $\omega = \tau_1 \dots \tau_m$ *and* $\mathbf{MALC} \vdash \Gamma_{\tau_i} \to p_{B_i}$ *for every* $i \in \{1,\dots,m\}$.

Proof. We can rearrange the derivation, so that the applications of $(\to \cdot)$ will be in the bottom (they are interchangeable with $(\cap \to)$ and $(/ \to)$, and these two are the only ones that can be applied below $(\to \cdot)$). Now the statement of the lemma is obvious.

Lemma 6. *If* $\mathbf{MALC} \vdash \Gamma_\omega \to p_A$, *then* $\mathcal{G}_{\mathrm{cut}} \vdash [A, \omega]$.

Proof. Induction by the length of ω. If $\omega = a$, then the only possible case is $p_A \in \mathcal{M}_a$. Then $(A \to a) \in \mathcal{P}$, and $\mathcal{G}_{\mathrm{cut}} \vdash [A, a]$.

Now let ω contain at least two letters. Consider the lowest application of $(/ \to)$ in the derivation of $\Gamma_\omega \to p_A$. Beneath this application there are only applications of $(\cap \to)$—the ones that open the type to which $(/ \to)$ is applied, and the ones that deal with other types in Γ_ω. We can transform the derivation so that the latter will be applied before the application of $(/ \to)$. Then we have $\omega = \omega_1 a \tau \omega_2$, $p_{A'} \,/ (\bigcap_{j=1}^{k} P_{\beta_j}) \in \mathcal{M}_a$, and the derivation step looks as follows:

$$\frac{\Gamma_\tau \to \bigcap_{j=1}^{k} P_{\beta_j} \quad \Gamma_{\omega_1}\, p_{A'}\, \Gamma_{\omega_2} \to p_A}{\Gamma_{\omega_1}\, p_{A'} \,/ (\bigcap_{j=1}^{k} P_{\beta_j})\; \Gamma_\tau\, \Gamma_{\omega_2} \to p_A} \; (/ \to)$$

Then, by Lemma 4, $\mathbf{MALC} \vdash \Gamma_\tau \to P_{\beta_j}$ for every $j \in \{1, \dots, k\}$. By Lemma 5, if $\beta_j = B_{j1} \dots B_{jm_j}$, $\tau = \tau_{j1} \dots \tau_{jm_j}$, and $\mathbf{MALC} \vdash \Gamma_{\tau_{ji}} \to p_{B_{ji}}$ (for every j and i in the ranges). By induction hypothesis, $\mathcal{G}_{\mathrm{cut}} \vdash [B_{ji}, \tau_{ji}]$, and, adding $[a, a]$, we can apply the rule for $A' \to a\beta_1 \& \dots \& a\beta_k$, therefore $\mathcal{G}_{\mathrm{cut}} \vdash [A', a\tau]$.

By induction hypothesis for the right premise of the $(/ \to)$ rule, $\mathcal{G}_{\mathrm{cut}} \vdash [A, \omega_1 A' \omega_2]$. Finally, applying the cut rule to $[A', a\tau]$ and $[A, \omega_1 A' \omega_2]$, we get $[A, \omega_1 a \tau \omega_2] = [A, \omega]$.

Now we are ready to define $\mathscr{G} = \langle \Sigma, \rhd, H \rangle$. Let $H = p_S$, and $E \rhd a$ if and only if $E = G_a$. If $w \in \mathfrak{L}(\mathcal{G})$, then $\mathcal{G} \vdash [S, w]$, and, by Lemma 3, $\mathbf{MALC} \vdash \Gamma_w \to p_S$, whence $w \in \mathfrak{L}(\mathscr{G})$. Conversely, if $w \in \mathfrak{L}(\mathscr{G})$, then $\mathbf{MALC} \vdash \Gamma_w \to p_S$. By Lemma 6 we get $\mathcal{G}_{\mathrm{cut}} \vdash [S, w]$, and by Lemma 1 $\mathcal{G} \vdash [S, w]$. Hence, $w \in \mathfrak{L}(\mathcal{G})$.

Note that in $\mathscr{G}$ every $a \in \Sigma$ is associated with only one type (such grammars are called *grammars with single type assignment* or *deterministic grammars).* Having the intersection connective, it is usually easy to make our grammar deterministic (cf. [5]); for the pure Lambek calculus the fact that any context-free language is generated by a deterministic **MLC**-grammar is not obvious, but still valid, as shown by Safiullin [13].

Example 3. This construction gives the following **MALC**-grammar equivalent to the grammar from Example 2:

$$a \rhd p_A \cap (p_A \,/\, p_A) \cap (p_D \,/ (p_D \cdot p_V)) \cap \big(p_S \,/ ((p_A \cdot p_B) \cap (p_D \cdot p_C))\big)$$
$$b \rhd p_B \cap p_D \cap p_V \cap (p_B \,/ (p_B \cdot p_U))$$
$$c \rhd p_C \cap p_U \cap (p_C \,/\, p_C)$$

Acknowledgements. I am grateful to Prof. Mati Pentus and Alexey Sorokin for fruitful discussions. I am also grateful to Ivan Zakharyashchev for bringing my attention to conjunctive grammars.

This research was supported by the Russian Foundation for Basic Research (grants 11-01-00281-a and 12-01-00888-a) and by the Presidential Council for Support of Leading Scientific Schools (grant NŠ 5593.2012.1).

References

1. Bar-Hillel, Y., Gaifman, C., Shamir, E.: On the categorial and phrase-structure grammars. Bull. of the Research Council of Israel, Sect. F 9F, 1–16 (1960)
2. Buszkowski, W.: The equivalence of unidirectional Lambek categorial grammars and context-free languages. Zeitschr. für math. Logik und Grundl. der Math. 31, 369–384 (1985)
3. Greibach, S.: A new normal-form theorem for context-free phrase structure grammars. Journal of the ACM 12, 42–52 (1965)
4. Jeż, A.: Conjunctive grammars can generate non-regular unary languages. International Journal of Foundations of Computer Science 19(3), 597–615 (2008)
5. Kanazawa, M.: The Lambek calculus enriched with additional connectives. Journal of Logic, Language and Information 1, 141–171 (1992)

6. Kanazawa, M.: Lambek calculus: recognizing power and complexity. In: Gerbrandy, J., Marx, M., de Rijke, M., Venema, Y. (eds.) JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday. Amsterdam University Press, Vossiuspers (1999)
7. Lambek, J.: The mathematics of sentence structure. American Math. Monthly 65(3), 154–170 (1958)
8. Lambek, J.: On the calculus of syntactic types. In: Jakobson, R. (ed.) Structure of Language and its Mathematical Aspects. Amer. Math. Soc. (1961)
9. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
10. Okhotin, A.: The dual of concatenation. Theor. Comput. Sci. 345(2-3), 425–447 (2005)
11. Okhotin, A., Reitwießner, C.: Conjunctive grammars with restricted disjunction. Theor. Comput. Sci. 411(26-28), 2559–2571 (2010)
12. Pentus, M.: Lambek grammars are context-free. In: 8th Annual IEEE Symposium on Logic in Computer Science, pp. 429–433. IEEE Computer Society Press, Los Alamitos (1993)
13. Safiullin, A.N.: Derivability of admissible rules with simple premises in the Lambek calculus. Moscow University Math. Bull. 62(4), 72–76 (2007)
14. Szabari, A.: Alternujúce Zásobníkové Automaty (Alternating Pushdown Automata), in Slovak, diploma work (M. Sc. thesis), University of Košice, Czechoslovakia, p. 45 (1991)

On the Generative Power of Discontinuous Lambek Calculus

Alexey Sorokin

Moscow State University, Faculty of Mechanics and Mathematics,
Moscow Institute of Physics and Technology

Abstract. We prove that the class of languages which are generated by discontinuous Lambek grammars and do not contain the empty word is closed under intersection with regular languages. The size of the grammar for the intersection is linear with respect to the size of the initial discontinuous Lambek grammar.

1 Introduction

Lambek calculus was created in 1958 by Joachim Lambek ([5]) for modelling the syntactic structure of natural language. During last decades it has found various applications in computational linguistics, nevertheless some of its deficiencies have emerged. One of that deficiencies is that the standard Lambek calculus has the same weak generative power as the context-free grammars ([8]), which is insufficient for adequate representation of several linguistic phenomena, such as discontinuous idioms, ellipsis or medial extraction. This problem is solved by adding the discontinuous operations to the traditional Lambek calculus, which was done by Morrill([6]). The grammars based on the formalism obtained, called the discontinuous Lambek calculus, present a natural treatment of these phenomena, the work [7] contain numerous examples.

But, on the contrary to the generative mildly context-sensitive systems, such as multiple context-free grammars or MCFGs ([9]), we know practically nothing about formal properties of discontinuous Lambek grammars. In fact there are only two results concerning lower bounds: the one of Morrill and Valentin ([6]) shows that all permutation closures of regular languages are generated by discontinuous Lambek grammars (and in fact by 1-discontinuous grammars). The result of the author ([10]) shows that all the languages generated by k-displacement context-free grammars (or, which is the same, by $k+1$-well-nested multiple context-free grammars) are also generated by k-discontinuous Lambek grammars. This fact gave us some hope to prove the converse result (the analogue of Pentus theorem) but this is not the case. As shown by Kanazawa and Salvati, the MIX language $\{w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c\}$ is not a TAG-language and therefore not a 1-well-nested multiple context-free language, but is obviously a permutation closure of a regular language. Moreover, it is argued that it is not well-nested at all (see [3] for the discussion) which implies that discontinuous Lambek grammars generate a wider family of languages

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 250–262, 2013.

than well-nested MCFGs. The explanation of this fact is that there is no way to model the split operation of discontinuous Lambek grammars by the means of well-nested MCFGs.

The author does not know any other nontrivial results concerning the generative power of discontinuous Lambek grammars. In our work we show that discontinuous Lambek grammars are closed under intersections with regular languages. At first we prove that result for usual Lambek grammars, where it is certainly the easy consequence of Pentus theorem and the well-known fact that context-free languages are closed under intersections with regular languages. The point is that the construction from the proof of Pentus theorem leads to the exponential growth of the size of the grammar, while our construction leads only to linear growth. Then we apply in fact the same algorithm to the discontinuous Lambek grammars.

2 Lambek Calculus

2.1 Axiomatics and Basic Definitions

In this section we describe the standard Lambek calculus and introduce the notion of a Lambek grammar. Let Tp be a countable set of primitive types which we denote by lower Latin letters p, q, r possibly with subscripts. The types are constructed from the primitive types with the help of binary connectives $\backslash, /, \cdot$ (left and right divisions and product). Formally, the set of types Tp is the smallest set containing Pr such that for any types $A, B \in \mathrm{Tp}$ the elements $(A\backslash B), (B/A)$ and $A \cdot B$ also belong to Tp. The sequents of Lambek calculus have the form $\Gamma \to A$, where Γ is a sequence of types and A is a type. The calculus obtained is denoted by L^* (the asterisk stands for the fact that we allow the antecedent to be empty).

Lambek calculus has the only axiom $A \to A$, $A \in \mathrm{Tp}$ and the following derivation rules:

$$\frac{\Pi A \to B}{\Pi \to B/A}(\to /) \qquad \frac{\Gamma B \Phi \to C \quad \Pi \to A}{\Gamma(B/A)\Pi\Phi \to C}(/ \to)$$

$$\frac{A\Pi \to B}{\Pi \to A\backslash B}(\to \backslash) \qquad \frac{\Gamma B \Phi \to C \quad \Pi \to A}{\Gamma \Pi (A\backslash B)\Phi \to C}(\to /)$$

$$\frac{\Gamma \to A \quad \Phi \to B}{\Gamma\Phi \to A \cdot B}(\to \cdot) \qquad \frac{\Gamma A B \Phi \to C}{\Gamma(A \cdot B)\Phi \to C}(\cdot \to)$$

The notation $\mathrm{L}^* \vdash \Gamma \to A$ denotes the derivability of the sequent $\Gamma \to A$ in the Lambek calculus with empty antecedents. It is possible to show by standard methods that Lambek calculus possesses the subformula property and cut-elimination, which means that cut rule $\dfrac{\Gamma B \Phi \to C \qquad \Pi \to B}{\Gamma \Pi \Phi \to C}$ is admissible in this calculus. The substitution rule is also admissible.

Note that the connectives $\backslash, /, \cdot$ form a residuated triple. Also note that the product operation is implicitly associative. These statements together imply that the types $(A\backslash B)/C$ and $A\backslash(B/C)$ are equivalent and consequently can be interchanged. Therefore we will omit the brackets in such types.

Example 1. The sequent $(p/q)q(p\backslash r) \to r$ is derivable in Lambek calculus. The derivation is given below.

$$\dfrac{\dfrac{p \to p \quad q \to q}{(p/q)q \to p} \qquad r \to r}{(p/q)q(p\backslash r) \to r}$$

The lemma below contains several statements, which can be easily proved by induction on cut-free derivation length.

Lemma 1.

1. *If* $\mathrm{L}^* \vdash \Gamma_1(B \cdot C)\Gamma_2 \to A$ *then the sequent* $\Gamma_1 BC\Gamma_2 \to A$ *is also derivable.*
2. *If* $\mathrm{L}^* \vdash \Gamma \to (B/C)$ *then the sequent* $\Gamma C \to B$ *is also derivable.*
3. *If* $\mathrm{L}^* \vdash \Gamma \to (C\backslash B)$ *then the sequent* $C\Gamma \to B$ *is also derivable.*

Lambek grammar is a triple $\mathcal{G} = \langle \Sigma, Lex, H \rangle$ where Σ is a finite alphabet, Lex is a finite subset of the set $\Sigma \times \mathrm{Tp}$ called lexicon and H is a distinguished element of Tp. A language of the grammar consists of all the words $a_1 \dots a_k$ such that there are types $A_1, \dots A_k$ such that all the pairs $\langle a_i, A_i \rangle$ are in Lex and the sequent $A_1 \dots A_k \to H$ is derivable.

Example 2. The grammar $\mathcal{G} = \langle \{a, b\}, Lex, p/p \rangle$ where Lex contains the pairs $\langle a, (p/p)/p \rangle$ and $\langle b, p \rangle$ generates the language $L = \{w \in \{a, b\}^* \mid |w|_a = |w|_b, \forall u \sqsubset w |u|_a > |u|_b\}$ which is isomorphic to the language of correct bracket sequences over one type of brackets. Note that the empty word is also in L because the sequent $\to p/p$ is derivable in L^*.

In general, as proved by Gaifman ([2]), all context-free languages without an empty word are generated by some Lambek grammar, based on the calculus L. In the work [4] Stepan Kuznetsov strengthened this result and proved that any context-free language is generated by some L^*-grammar. The opposite result (that Lambek grammars based on both the calculi L and L^* generate only context-free languages) is proved by Pentus in [8]. Hence the class of languages generated by L^*-grammars coincides with the class of context-free languages.

So it is trivial to prove that the languages generated by L^*-grammars are closed under intersections with regular languages. To construct the grammar for the intersection one should at first transform the initial Lambek grammar to the equivalent context-free grammar by the algorithm from Pentus work, then obtain the context-free grammar for the intersection (the algorithm can be found in any textbook on formal languages, for example [1]) and then transform the context-free grammar back to L^*-grammar by the method from Gaifman's work. However, this algorithm is impractical since the construction of Pentus leads to exponential growth of the grammar size.

Our goal is to construct directly the Lambek grammar for the intersection, in a way which doesn't extend the size of the grammar exponentially. Also we want to describe the construction which can be further adopted to the case of discontinuous calculus.

2.2 Lambek Grammars and Regular Languages

In this section we give a direct proof of the fact that the class of languages generated by L*-grammars is closed under intersection with regular languages. Actually all the theorems of that type for different families of generative grammars are proved in the same way. We adopt this algorithm for the case of Lambek grammar which generates the language without the empty word.

Let R be a particular regular language. By the Kleene theorem we assume that this language is generated by some nondeterministic finite automaton M with one-letter transitions. Moreover, since we consider only the languages without the empty word we can assume that the language R also does not contain the empty word, so it can be generated by a nondeterministic finite automaton with only one terminal state. Let M have the set of states $Q = \{q_1, \ldots, q_n\}$, $\Delta \in Q \times \Sigma \times Q$ be the set of transitions and q_1, q_f be the initial and the final state respectively.

To clarify the further we recall the sketch of the standard proof of the fact that context-free languages are closed under intersection with regular languages. We expect the reader to be familiar with the notions of context-free grammar and Chomsky normal form for them. So let L be a context-free language not containing the empty word and $G = \langle N, \Sigma, P, S\rangle$ be the grammar in Chomsky normal form generating L. Here N is a set of nonterminals, Σ is an alphabet, P is a set of productions and S is an initial nonterminal. Let R be the regular language defined above. Then the intersection is generated by the grammar $G_R = \langle Q \times N \times Q, \Sigma, P_R, \langle q_1, S, q_f\rangle\rangle$ where for any rule $B \to CD \in P$ and arbitrary states $q', q'', q''' \in Q$ the set P_R contains the rule $\langle q', B, q''\rangle \to \langle q', C, q'''\rangle\langle q''', D, q''\rangle$ and for any rule $A \to a \in P$ and any transition $\langle q', a, q''\rangle \in \Delta$ the set P_R contains the rule $\langle q', A, q''\rangle \to a$. Actually, the nonterminal $\langle q', A, q''\rangle$ generates exactly the words with are generated from A in the old grammar and are the labels of some path from q' to q''. We use the same construction for the Lambek calculus.

Let $\mathcal{G} = \langle \Sigma, Lex, H\rangle$ be a L*-grammar generating the language L. We want to prove that the language $L \cap R$ is also generated by some Lambek grammar G'. Let $T(G) = \{A \mid \langle a, A\rangle \in Lex\} \cup \{H\}$ be the set of types used in the grammar and $\mathcal{T}$ be the set of all the subtypes of the types in $T(G)$. Without the loss of generality we assume that the types of $\mathcal{T}$ are constructed from primitive types $p_1, p_2, \ldots$. We extend the set of primitive types with new types $q_1, \ldots, q_n$ corresponding to the states from Q. For every type T from $\mathcal{T}$ we denote by T_{ij} the type $(q_i \backslash T) \cdot q_j$, $i, j > 0$. We also denote by T_{0j} the type $T \cdot q_j, j > 0$, by T_{i0} the type $q_i \backslash T, i > 0$ and set $T_{00} = T$. We denote by $\mathcal{T}'$ the set $\{T_{ij} \mid T \in \mathrm{Tp}, i, j \in \overline{0, n}\} \cup \{q_i \mid i \in \overline{1, n}\}$. Note that $\mathcal{T}'$ is closed under subtypes.

Let $\overline{Q} = Q \cup \{\overline{q} \mid q \in Q\}$. For every $i \in \overline{1, n}$ we set $\Theta_i = q_i$, $\overline{\Theta}_i = \overline{q}_i$, $\Theta_0 = \overline{\Theta}_0 = \Lambda$, where Λ is an empty sequence. For every type T_{ij} we define its q-image as the string $\overline{\Theta}_i \Theta_j$, also we define the q-image of q_j as q_j. For every sequence $\Gamma \in \mathcal{T}'^*$ its q-image Γ_q equals the concatenation of q-images of the types in Γ. We continue the mapping to sequents of Lambek calculus, setting $(\Gamma \to T_{ij})_q = \Theta_i \Gamma_q \overline{\Theta}_j$ and $(\Gamma \to q_i)_q = \Gamma_q q_i$

The elements of $\overline{Q}$ in a natural way correspond to brackets of n different types. Namely, the elements of Q are opening brackets and the overlined elements are closing brackets. This representation is in fact a light simulation of a proof-net.

Recall that if $\overline{Q} = \{q_i \mid 1 \leq i \leq n\} \cup \{\overline{q}_i \mid 1 \leq i \leq n\}$ is a set of brackets of n different types, then correct bracket sequences are exactly the words generated by the context-free grammar with the rules $S \to \epsilon$, $S \to q_i S \overline{q}_i S$. Informally, the brackets of a correct bracket sequence can be partitioned to pairs satisfying the following conditions: 1) in each pair the left element is q_i and the right element is $\overline{q}_i$ for some $i \leq n$, 2) we can connect the brackets of each pair by an arc in upper semiplane in a way such that these arcs do not intersect.

We want to prove that the q-images of derivable sequents are correct bracket sequences. To achieve this goal we need some proposition which also plays a significant role in the following.

Proposition 1. *Let $\Gamma \in \mathcal{T}'^*$ and $\mathrm{L}^* \vdash \Gamma \to q_i$. Then $\Gamma = q_i$.*

Proof. Induction on derivation length. Since the succedent includes a single primitive type, then only the application of rules $\setminus \to, / \to$ or $\cdot \to$ is possible on the last step. But then one of the preceding sequents also had q_i in the succedent and due to the definition of $\mathcal{T}'$ and the induction hypothesis the application of the mentioned rules is impossible.

Lemma 2. *If $\Gamma \in \mathcal{T}'^*$, $A \in \mathcal{T}'$ and $\mathrm{L}^* \vdash \Gamma \to A$ then $(\Gamma \to A)_q$ is a correct bracket sequence.*

Proof. Induction on derivation length. There are several cases depending on the last rule in derivation.

Let the last rule be $\dfrac{\Gamma_1 \to A_1 \quad \Gamma_2 \to A_2}{\Gamma_1\Gamma_2 \to A_1 \cdot A_2}$. Then there are two subcases. In the first $A_2 \in \mathcal{T}$, then $A_1 \cdot A_2 \in \mathcal{T}$ by the definition of the system $\mathcal{T}'$, so $(\Gamma_1\Gamma_2 \to A_1 \cdot A_2)_q = (\Gamma_1\Gamma_2)_q = (\Gamma_1)_q(\Gamma_2)_q = (\Gamma_1 \to A_1)_q(\Gamma_2 \to A_2)_q$ and $(\Gamma \to A)_q$ is a correct bracket sequence as a concatenation of two correct ones. In the second case $A_2 = q_j$, then $A_1 = T_{i0}$ for some i, so $(\Gamma_1\Gamma_2 \to A_1 \cdot A_2)_q = \Theta_i(\Gamma_1)_q(\Gamma_2)_q\overline{\Theta}_j = (\Gamma_1 \to A_1)_q(\Gamma_2 \to A_2)_q$ and $(\Gamma \to A)_q$ is correct by the same arguments.

Let the last rule be $\cdot \to$. Obviously it does not change the q-image of the sequent, so we can use the induction hypothesis. Let the last rule be $\dfrac{\Gamma A_2 \to A_1}{\Gamma \to (A_1/A_2)}$ then by construction $A_1/A_2 \in \mathcal{T}$, so $(\Gamma \to A_1/A_2)_q = (\Gamma)_q = (\Gamma A_2 \to A_1)_q$ and $(\Gamma \to A)_q$ is a correct bracket sequence by the induction hypothesis.

Let the last rule be $\dfrac{A_2\Gamma \to A_1}{\Gamma \to (A_2\backslash A_1)}$. In the case $A_2 \notin Q$ the proof is the same as for the opposite division. Otherwise $A_2 = q_i$ for some i and $(\Gamma \to (A_2\backslash A_1))_q = q_i(\Gamma)_q = (A_2\Gamma \to A_1)_q$ and $(\Gamma \to A)_q$ is correct by induction hypothesis.

Let the last rule be $\dfrac{\Gamma_1 B_1 \Gamma_2 \to T_{ij} \quad \Phi \to B_2}{\Gamma_1(B_1/B_2)\Phi\Gamma_2 \to T_{ij}}$. Then by definition of $\mathcal{T}'$ it holds that $(B_1/B_2) \in \mathcal{T}$, so $(\Gamma_1(B_1/B_2)\Phi\Gamma_2 \to T_{ij})_q = \Theta_i(\Gamma_1)_q\Phi_q(\Gamma_2)_q\overline{\Theta}_j$. Since $\Phi_q = (\Phi \to B_2)_q$ and $\Theta_i(\Gamma_1)_q\Phi_q(\Gamma_2)_q\overline{\Theta}_j = (\Gamma_1 B_1 \Gamma_2 \to T_{ij})_q$ then the q-image of the sequent under consideration is obtained by inserting the q-image of one of its

premises into the q-image of its another premise. Obviously, inserting a correct bracket sequence inside another correct sequence yields a correct sequence again, so the statement of the lemma holds. The case of the other division is analogous. We should consider two subcases, in one of them there is a q-type under division. We leave the detailed analysis because of its simplicity. The lemma is proved.

Corollary 1. *If* $\mathrm{L}^* \vdash T^{(1)}_{i_1 j_1} \dots T^{(r)}_{i_r j_r} \to T_{ij}$ *then* $i_1 = i$, $i_{l+1} = j_l$ *for every* $l \in \overline{1, r-1}$ *and* $j_r = j$.

The corollary has an easy consequence which we will use in the further: if in the notation used above $i_1 > 0$ then it also holds that $i > 0$ which means that there is a left division on some q_i in the succedent.

Corollary 2. *If* $\Gamma = T^{(1)}_{i_1 j_1} \dots T^{(r)}_{i_r j_r}, i_1 > 0$ *and* $\mathrm{L}^* \vdash \Gamma \to T_{ij}$ *then* $i > 0$.

Proposition 2. *If* $T \in \mathcal{T}'$, $\Gamma \in \mathcal{T}'^*$ *and* $\mathrm{L}^* \vdash \Gamma q_i \to T \cdot q_j$ *then* $i = j$ *and* $\mathrm{L}^* \vdash \Gamma \to T$.

Proof. Induction on derivation length. If the last rule is $\dfrac{\Gamma_1 \to T \quad \Gamma_2 q_i \to q_j}{\Gamma_1 \Gamma_2 q_i \to T \cdot q_j}$ then by Proposition 1 it follows that $\Gamma_2 = \Lambda$ and $q_i = q_j$, then $\Gamma = \Gamma_1$ and the second statement is also proved. If the last rule was $\setminus \to$ then q_i and q_j were in the same premise and we can use the induction hypothesis and apply the last rule. In the case of the rule $\to \cdot$ we also straightforwardly apply the induction hypothesis.

It remains to consider the case when the last rule application has the form $\dfrac{\Gamma_1 B \Gamma_2 \to T \cdot q_j \quad \Pi \to C}{\Gamma_1 (B/C) \Pi \Gamma_2 \to T \cdot q_j}$. If Γ_2 includes q_i then we proceed just like in the case of the rule $\setminus \to$. The remaining case is when Γ_2 is empty and $\Pi = \Pi_1 q_i$. By the definition of the set $\mathcal{T}'$ the type C does not contain any q-s, so the q-image of the sequent $\Pi \to C$ must end with q_i. But it contradicts with the Lemma 2. All the cases are inspected and the proposition is proved.

The next lemma is the most important step of the proof.

Lemma 3. *If* $B \in \mathcal{T}$ *then for all* $\Gamma_1, \Gamma_2 \in \mathcal{T}'^*, q_i \in Q, C \in \mathcal{T}'$ *the statements* $\mathrm{L}^* \vdash \Gamma_1 B \Gamma_2 \to C$ *and* $\mathrm{L}^* \vdash \Gamma_1 q_i (q_i \setminus B) \Gamma_2 \to C$ *are equivalent .*

Proof. Let the sequent $\Gamma_1 B \Gamma_2 \to C$ be derivable. Then the derivation $\dfrac{q \to q \quad \Gamma_1 B \Gamma_2 \to C}{\Gamma_1 q_i (q_i \setminus B) \Gamma_2 \to C}$ shows that another sequent is also derivable. Now let the sequent $\Gamma_1 q_i (q_i \setminus B) \Gamma_2 \to C$ be derivable. Then we proceed by induction on derivation length. If the types q_i and $q_i \setminus B$ are in the same premise of the last rule, then we can use the induction hypothesis and then apply the last rule to prove the derivability of the sequent $\Gamma_1 B \Gamma_2 \to C$. It remains to inspect the three cases when they are in different premises of the last rule. We may assume that the antecedent does not contain a type of the form $A_1 \cdot A_2$ since in this case a rule $\cdot \to$ could be applied.

The first case: the last rule application was $\dfrac{\Gamma_1 q_i \to C_1 \quad (q_i\backslash B)\Gamma_2 \to C_2}{\Gamma_1 q_i (q_i\backslash B)\Gamma_2 \to C_1 \cdot C_2}$. But in this case by Corollary 2 the type C_2 must have the form $q_i\backslash C_3$ which makes a contradiction with the condition that $C_1 \cdot C_2 \in \mathcal{T}'$.

The second case: the last rule application was $\dfrac{\Gamma_1 q_i D\Gamma_3 \to C \quad (q_i\backslash B)\Phi \to E}{\Gamma_1 q_i (q_i\backslash B)\Phi(E\backslash D)\Gamma_3 \to C}$ where $\Gamma_2 = \Phi(E\backslash D)\Gamma_3$. By the Corollary 2 the type E must have the form $E = q_i\backslash E_1$ which is impossible by the definition of $\mathcal{T}'$.

The third case: the last rule application was $\dfrac{\Gamma_3 B\Gamma_2 \to C \quad \Phi q_i \to q_i}{\Gamma_3 \Phi q_i (q_i\backslash B)\Gamma_2 \to C}$. By Lemma 1 $\Phi = \Lambda$ and then we obtain that the first premise of the rule is exactly $\Gamma_1 B\Gamma_2 \to C$. That was required.

Now we can construct the grammar for the language $L \cap R$. The new grammar is $G' = \langle \Sigma, Lex', H' \rangle$ where $H' = (q_1\backslash H) \cdot q_f$ (recall that q_1 and q_f are the initial and the only terminal state in the automaton recognizing R). We set $Lex' = \{\langle a, T_{ij}\rangle \mid \langle a, T\rangle \in Lex,\ \langle q_i, a, q_j\rangle \in \Delta\}$. Now we should prove that this grammar generates exactly the desirable language.

Lemma 4. *The grammar $G' = \langle \Sigma, Lex', H' \rangle$ generates the language $L \cap R$.*

Proof. At first let us prove that all the words from $L \cap R$ are in $L(G')$. Let $w = a_1 \dots a_r$ be such a word and $T^{(1)}, \dots, T^{(r)}$ be the types such that for every i it holds that $\langle a_i, T^{(i)}\rangle \in Lex$ and the sequent $T^{(1)}, \dots, T^{(r)} \to H$ is derivable in Lambek calculus. Let $q_{i_0} = q_1, q_{i_1}, q_{i_2}, q_{i_r} = q_f$ be the path leading in M from the initial state to the final (recall that M is the automaton recognizing R).

Consider the types $T^{(1)}_{i_0 i_1}, T^{(2)}_{i_1 i_2}, \dots, T^{(r)}_{i_{r-1} i_r}$, it is not difficult to see that for every l the pair $\langle a_l, T^{(l)}_{i_{l-1} i_l}\rangle$ is in Lex'. Therefore it remains to prove that the sequent $T^{(1)}_{i_0 i_1} \dots, T^{(r)}_{i_{r-1} i_r} \to H'$ is derivable in L*. By the properties of product connective it suffices to show that the sequent $T^{(1)}_{i_0 0} q_{i_1} T^{(2)}_{i_1 0} q_{i_2} \dots T^{(r)}_{i_{r-1} 0} q_{i_r} \to (q_{i_0}\ H) \cdot q_{i_r}$ is derivable.

Applying the Lemma 3 $r-1$ times we obtain that it suffices to show that the sequent $(q_{i_0}\backslash T^{(1)})T^{(2)} \dots T^{(r-1)} T^{(r)} q_{i_r} \to (q_1\backslash H) \cdot q_{i_r}$ is derivable. Using the rules $\to\backslash$ and $\to\cdot$ we reduce this problem to the derivability of the sequent $T^{(1)} \dots T^{(r)} \to H$. But it is derivable since the word $a_1 \dots a_r$ was in L.

Now we should prove the lemma in the opposite direction. Let the word $a_1 \dots a_r$ belong to $L(G')$, it means that there are such types $T^{(1)}_{i_1 j_1} \dots T^{(r)}_{i_r j_r}$ that $\langle a_l, T^{(i)}_{i_l j_l}\rangle \in Lex'$ for every l and L$^* \vdash T^{(1)}_{i_1 j_1} \dots T^{(r)}_{i_r j_r} \to (q_1\backslash H) \cdot q_f$. The q-image of the last sequent equals $q_1 \overline{q}_{i_1} q_{j_1} \overline{q}_{i_2} \dots q_{j_r} \overline{q}_f$. Then by the Corollary 1 it holds that $i_1 = 1, i_{l+1} = j_l$ for $l = 1, 2, \dots, r-1$ and $q_{j_r} = q_f$. So the sequent has the form $T^{(1)}_{1 i_2} \dots T^{(r)}_{i_r f} \to (q_1\backslash H) \cdot q_f$. Applying the Lemma 3 $r-1$ times we obtain that the sequent $(q_1\backslash T^{(1)})T^{(2)} \dots T^{(r)} \cdot q_f \to (q_1\backslash H) \cdot q_f$ is also derivable. By the Proposition 2 the sequent $(q_1\backslash T^{(1)})T^{(2)} \dots T^{(r)} \to q_1\backslash H$ is also derivable, it implies the derivability of the sequent $q_1(q_1\backslash T^{(1)})T^{(2)} \dots T^{(r)} \to H$ which by Lemma 3 yields the derivability of the sequent $T^{(1)}T^{(2)} \dots T^{(r)} \to H$ which was required. The lemma is proved.

From this lemma directly follows the theorem below.

Theorem 1. *The class of languages which do not contain the empty word and are generated by a Lambek grammar is closed under intersections with regular languages.*

The proof of the theorem can be significantly simplified by using the technique of proof-nets. But since the proof-nets of discontinuous Lambek calculus are not so convenient for the same purpose we have decided to prove the theorem above directly. In the next section we slightly modify the proof to obtain the analogous result for discontinuous Lambek calculus.

3 Discontinuous Lambek Calculus

3.1 Axiomatics

Various linguistic phenomena cannot be captured by context-free grammars and, hence, by usual Lambek grammars. Therefore there were several attempts to extend the vocabulary of the calculus but preserve its essential features. One of the most successful extensions of Lambek calculus is the discontinuous Lambek calculus of Morrill ([6]). In this calculus the discontinuous product operation $\odot$ and its residuals $\downarrow$ and $\uparrow$ were added to the standard set of connectives. The discontinuous product operation is just a replacement of the distinguished separator in its first argument by its second argument.

We start by defining the sort function $s()$ from the set of primitive types to the set of natural numbers. We also extend the set of primitive types with two constants I and J such that $s(I) = 0$ and $S(J) = 1$. The type I corresponds to the empty word (the neutral element under concatenation) and J corresponds to the separator (the neutral element under intercalation). The types of discontinuous Lambek calculus are constructed from the primitive types with the help of continuous connectives $\backslash, /, \cdot$ and discontinuous connectives $\downarrow_k, \uparrow_k, \odot_k, k \in \mathbb{N}^+$. The notion of sort is extended to complex types in the following way:

1. If $A, B \in \mathrm{Tp}$ and $s(A) \geq s(B)$ then $(A/B) \in \mathrm{Tp}, (B\backslash A) \in \mathrm{Tp}$ and $s(A/B) = s(B\backslash A) = sA - sB$.
2. If $A, B \in \mathrm{Tp}$ then $(A \cdot B) \in \mathrm{Tp}$ and $s(A \cdot B) = sA + sB$.
3. If $A, B \in \mathrm{Tp}$ and $s(A) \geq s(B) - 1$ then for every $k \leq s(B)$ $B \downarrow_k A \in \mathrm{Tp}$ and $s(B \downarrow_k A) = s(A) - s(B) + 1$.
4. If $A, B \in \mathrm{Tp}$ and $s(A) \geq s(B)$ then for every $k \leq sA - sB + 1$ $(A \uparrow_k B) \in \mathrm{Tp}$ and $s(A \uparrow_k B) = s(A) - s(B) + 1$.
5. If $A, B \in \mathrm{Tp}$ and $s(A) \geq 1$ then for every $k \leq sA$ $(A \odot_k B) \in \mathrm{Tp}$ and $s(A \odot B) = sA + sB - 1$.

Let $\mathcal{F}_i$ denote the set of the types of the sort i, Λ be the empty string and $[]$ be a metalinguistic separator. Then the set of hyperconfigurations is defined by the following grammar: $\mathcal{O} ::== \Lambda | [] | \mathcal{F}_0 | \mathcal{F}_i\{\underbrace{\mathcal{O}, \ldots, \mathcal{O}}_{i\mathcal{O}'s}\} | \mathcal{O}, \mathcal{O}$.

The sort of a hyperconfiguration $s(\Gamma)$ is the number of separators it contains and is defined inductively: $s(\Lambda) = 0$, $s([]) = 1$; $s(A) = 0$ for $A \in \mathcal{F}_0$; $s(A\{\Gamma_1, \ldots, \Gamma_{s(A)}\}) = s(\Gamma_1) + \ldots + s(\Gamma_{s(A)})$; $s(\Gamma, \Phi) = s(\Gamma) + s(\Phi)$. The sequents are of the form $\Gamma \to A$, where $s(\Gamma) = s(A)$. For every type A we define its vector $\overrightarrow{A}$, which equals A if $sA = 0$ and $A\{\underbrace{[], \ldots, []}_{sA[]'s}\}$ in the other case. For any two configurations Γ, Φ we denote by $\Gamma|_k\Phi$ the result of replacing the k-th separator in Γ by Φ (it is valid only if $k \leq s(\Gamma)$). If Γ is a configuration of sort i then we denote by $\Gamma \otimes \langle \Phi_1, \ldots, \Phi_i \rangle$ the result of simultaneous replacement of all the successive separators in Γ by the hyperconfigurations $\Phi_1, \ldots, \Phi_i$.

To formulate the rules of discontinuous Lambek calculus we need to introduce the notion on a hypercontext. For the standard Lambek calculus a context is just a sequence of types with a gap in it. If Γ is a context and A is a type, we denote by $G[A]$ the result of filling the gap in Γ with A. The usual lefthand rules of Lambek calculus can be reformulated more concisely using the context notation,for example, the rule $/ \to$ takes the form $\dfrac{\Gamma[B] \to A \quad \Pi \to C}{\Gamma[(B/C)\Pi] \to A}$. In the discontinuous Lambek calculus we have to generalize the notion of a context, which leads to a hypercontext. Namely, if Φ_0 is a hyperconfiguration with a distinguished placeholder instead of one of its subhyperconfigurations and $\Phi_1, \ldots, \Phi_k$ are configurations, then the notation $\Phi\langle\Gamma\rangle$ stands for $\Phi_0[\Gamma \otimes \langle \Phi_1, \ldots, \Phi_k \rangle]$.

The axiomatics of discontinuous Lambek calculus DL is given below.

$$\frac{}{A \to A}(ax) \qquad \frac{\Gamma \to A \quad \Phi\langle\overrightarrow{A}\rangle \to B}{\Phi\langle\Gamma\rangle \to B}(cut)$$

$$\frac{\overrightarrow{A}, \Gamma \to C}{\Gamma \to A\backslash C}(\to \backslash) \qquad \frac{\Gamma \to A \quad \Phi\langle\overrightarrow{C}\rangle \to D}{\Phi\langle\Gamma, \overrightarrow{A\backslash C}\rangle \to D}(\backslash \to)$$

$$\frac{\Gamma, \overrightarrow{A} \to C}{\Gamma \to C/A}(\to /) \qquad \frac{\Gamma \to A \quad \Phi\langle\overrightarrow{C}\rangle \to D}{\Phi\langle\overrightarrow{C/A}, \Gamma\rangle \to D}(/ \to)$$

$$\frac{\Gamma \to A \quad \Phi \to B}{\Gamma, \Phi \to A \cdot B}(\to \cdot) \qquad \frac{\Phi\langle\overrightarrow{A}, \overrightarrow{B}\rangle \to D}{\Phi\langle\overrightarrow{A \cdot B}\rangle \to D}(\cdot \to)$$

$$\frac{}{\Lambda \to I}(\to I) \qquad \frac{\Phi\langle\Lambda\rangle \to A}{\Phi\langle I\rangle \to A}(I \to)$$

$$\frac{\overrightarrow{A}|_k\Gamma \to C}{\Gamma \to A \downarrow_k C}(\to\downarrow) \qquad \frac{\Gamma \to A \quad \Phi\langle\overrightarrow{C}\rangle \to D}{\Phi\langle\Gamma|_k\overrightarrow{A \downarrow_k C}\rangle \to D}(\downarrow\to)$$

$$\frac{\Gamma|_k\overrightarrow{A} \to C}{\Gamma \to C \uparrow_k A}(\to\uparrow) \qquad \frac{\Gamma \to A \quad \Phi\langle\overrightarrow{C}\rangle \to D}{\Phi\langle\overrightarrow{C \uparrow_k A}|_k\Gamma\rangle \to D}(\uparrow\to)$$

$$\frac{\Gamma \to A \quad \Phi \to B}{\Gamma|_k\Phi \to A \odot_k B}(\to \odot) \qquad \frac{\Phi\langle\overrightarrow{A}|_k\overrightarrow{B}\rangle \to D}{\Phi\langle\overrightarrow{A \odot_k B}\rangle \to D}(\odot \to)$$

$$\frac{}{[] \to J}(\to J) \qquad \frac{\Phi\langle[]\rangle \to A}{\Phi\langle J\rangle \to A}(I \to)$$

As well as the basic Lambek calculus, discontinuous Lambek calculus possess such useful properties as cut-elimination and the subformula property. Hence it is a conservative extension of the calculus L^*. If we bound the sort of all types and hyperconfigurations allowed in derivations by some natural number

k, we obtain the calculus DL_k which also is a conservative extension of Lambek calculus. Note that the sequent is derivable in DL iff it is derivable in some DL_k.

3.2 Discontinuous Lambek Grammars

Just like the types of standard Lambek calculus, the types of discontinuous Lambek calculus can be interpreted as sets of strings. Let 1 be a distinguished separator and $w \in (\Sigma \cup 1)^*$, then we can define a sort of the word $s(w) = |w|_1$. The types of the sort i are interpreted as sets of i-sorted words.

Now we extend a notion of Lambek grammar to discontinuous calculi. We give a definition, that slightly differs from one of Morrill ([6]). Our definition is less general, but is more traditional in the framework of Lambek grammars. In fact, the definition of Morrill can be simulated in our terms.

So, a DL-grammar is a again a triple $G = \langle \Sigma, Lex, H \rangle$. All the definitions remain the same up to natural changes. Lex is now a relation in $\Sigma \times \mathrm{Tp}$, where Tp is a set of discontinuous types. Also H must be a discontinuous type of the sort 0. The word $w = a_1 \dots a_r$ is in the language $L(G)$ generated by the grammar if there are such types $A^{(1)}, \dots, A^{(r)}$ that for every i the pair $\langle a_i, A^{(i)} \rangle$ is in Lex and the sequent $A^{(1)}, \dots, A^{(r)} \to H$ is derivable in DL. Note that commas in the last sequent are the elements of configuration.

Example 3. The grammar with the target type S and the lexicon $a : S/D, T/E$, $b : E \downarrow D, J\backslash(F \downarrow E), c : J\backslash E, T\backslash F$ generates the language $\{a^n b^n c^n \mid n \geq 1\}$.

Let L be a language, which does not contain the empty word and is generated by some DL-grammar, and R be a regular language. Our aim is to prove that the language $L \cap R$ is also generated by a DL-grammar. Again we assume that R does not contain the empty word and is generated by finite automaton M with the set of states $Q = \{q_1, \dots, q_n\}$, the transition relation Δ, the initial state q_1 and the final state q_f. We define the sets $\mathcal{T}$ and $\mathcal{T}'$ in the same way as we did for the calculus L^*. Again T_{ij} denotes the type $(q_i\backslash T) \cdot q_j$.

The general scheme of the proof is the same as in the case of the calculus L^*. We start from the following lemma:

Lemma 5.

1. *If* $\mathrm{DL} \vdash \Gamma_1, (B \cdot C), \Gamma_2 \to A$ *then the sequent* $\Gamma_1, B, C, \Gamma_2 \to A$ *is also derivable.*
2. *If* $\mathrm{DL} \vdash \Gamma \to (B/C)$ *then the sequent* $\Gamma, C \to B$ *is also derivable.*
3. *If* $\mathrm{DL} \vdash \Gamma \to (C\backslash B)$ *then the sequent* $C, \Gamma \to B$ *is also derivable.*

We denote by $C(\mathcal{T}')$ the set of hyperconfigurations which contain only the types from $\mathcal{T}'$. For every hyperconfiguration $\Gamma \in C(\mathcal{T}')$ we want to define its q-image. We use the same notation as in the previous section. Then q-image Γ_q of Γ satisfies the following inductive conditions: $\Lambda_q = []_q = \Lambda$, $(q_i)_q = q_i$, $(\overrightarrow{T_{ij}})_q = \overline{\Theta}_i \Theta_j$, $(T_{ij}\langle \Gamma_1, \dots, \Gamma_m \rangle)_q = \overline{\Theta}_i (\Gamma_1)_q \dots (\Gamma_m)_q \Theta_j$, $(\Gamma_1, \Gamma_2)_q = (\Gamma_1)_q (\Gamma_2)_q$. We also extend the definition of q-images to the sequents of discontinuous Lambek calculus, setting $(\Gamma \to T_{ij})_q = \Theta_i (\Gamma)_q \overline{\Theta}_j$. We prove that

the q-image of a derivable sequent satisfies the same conditions as in the case of calculus L*:

Proposition 3. *Let* $\Gamma \in C(\mathcal{T}')$ *and* $\mathrm{DL} \vdash \Gamma \to q_i$. *Then* $\Gamma = q_i$.

Lemma 6. *If* $\Gamma \in C(\mathcal{T}')$, $A \in \mathcal{T}'$ *and* $\mathrm{DL} \vdash \Gamma \to A$ *then* $(\Gamma \to A)_q$ *is a correct bracket sequence.*

Proof. Induction on the length of derivation. Consider the last rule in the derivations. We will not inspect the cases of the rules that introduce continuous connectives since the proof is the same as in the case of L*. We will omit the vector notation when writing the hyperconfigurations which contain a single type.

Let the last rule be $\odot \to$ in the form $\dfrac{\Gamma\langle B_1|_k B_2\rangle \to A}{\Gamma\langle B_1 \odot B_2\rangle \to A}$. Then by definition of the set $\mathcal{T}'$ the type $B_1 \odot B_2$ does not contain any occurrences of q. So the application of this rule does not change the q-image and we can use the induction hypothesis.

Let the last rule be $\to \odot$ in the form $\dfrac{\Gamma_1 \to A \quad \Gamma_2 \to B}{\Gamma_1|_k\Gamma_2 \to A \odot_k B}$. By the construction the succedent does not contain occurrences of q, so $(\Gamma_1|_k\Gamma_2 \to A \odot_k B)_q = (\Gamma_1|_k\Gamma_2)_q$. Then the q-image of the sequence given is obtained by inserting the q-image of Γ_2 into the middle of the q-image of Γ_1. But both these q-images are correct bracket sequences by induction hypothesis and then the result of the insertion is also a correct bracket sequence.

If the last rule is $\to\downarrow$ or $\to\uparrow$ then the antecedent also contains no q occurrences, so the reverse application of this rule does not change the q-image of the sequent and we can use the induction hypothesis. In the case when discontinuous connectives are introduced in the antecedent it is not difficult to see that the q-image of the sequent under consideration is again a result of insertion of q-image of its premise into the q-image of its another premise. By induction both these q-images are correct bracket sequences and is correct also. The lemma is proved.

Corollary 3. *If* $\mathrm{DL} \vdash T^{(1)}_{i_1 j_1} \dots T^{(r)}_{i_r j_r} \to T_{ij}$ *then* $i_1 = i$, $i_{l+1} = j_l$, $l \in \overline{1, r-1}$, $j = j_r$.

Corollary 4. *If* $\Gamma = T^{(1)}_{i_1 j_1} \dots T^{(r)}_{i_r j_r}, i_1 > 0$ *and* $\mathrm{L}^* \vdash \Gamma \to T_{ij}$ *then* $i > 0$.

Proposition 4. *If* $T \in \mathcal{T}'$, $\Gamma \in C(\mathcal{T}')$ *and* $\mathrm{DL} \vdash \Gamma, q_i \to T \cdot q_j$ *then* $i = j$ *and* $\mathrm{DL} \vdash \Gamma \to T$.

Proof. Induction on derivation length. If the last rule introduces a "continuous" connective, then the proof copies the proof of Proposition 4. Otherwise some discontinuous connective is introduced in the antecedent, namely in Γ. It is not difficult to see that in this case q_i and q_j were in the same premise before the last step, so we can use the induction hypothesis and then apply the last rule of the derivation to the sequent without q_i and q_j.

Now we prove the main technical lemma, which allows us to "contract" the q_i-s.

Lemma 7. *If $B \in \mathcal{T}$ then for all contexts Γ that contain only the types from $\mathcal{T}'$, $q_i \in Q$ and $C \in \mathcal{T}'$ the statements* $\mathrm{DL} \vdash \Gamma\langle B\rangle \to C$ *and* $\mathrm{DL} \vdash \Gamma\langle q_i, q_i\backslash B\rangle \to C$ *are equivalent.*

Proof. In one direction the proof repeats the one in Lemma 3. So we need to prove the converse statement: the derivability of the sequent $\Gamma\langle q_i, q_i\backslash B\rangle \to C$ implies the derivability of the sequent $\Gamma\langle B\rangle \to C$. We use the induction on derivation length. Consider the last rule application. Again if q_i and $q_i\backslash B$ were in the same premise of the rule, then we could use induction hypothesis to the premises and then combine them by the last rule of the derivation. So it suffices to consider the cases when q_i and $q_i\backslash B$ were in different premises. That could happen when applying $\to\cdot, \to\odot, \backslash\to, /\to$ or $\downarrow\to$ rules.

The case when the last rule was $\to\cdot$ is verified in the same way as before. If the last rule was $\odot\to$ and the occurrences of q_i and $q_i\backslash B$ under consideration were in different premises, then the rule must have the form $\dfrac{\Gamma_1\langle q_i, []\rangle \to C_1 \quad (q_i\backslash B), \Pi \to C_2}{\Gamma_1\langle q_i, (q_i\backslash B), \Pi\rangle \to C_1 \odot C_2}$. In this case by Corollary 4 the type C_2 should contain some q_i which is impossible due to definition of $\mathcal{T}'$.

A similar argument works in the case of the rules $\to /$ and $\to \backslash$ (when it is not the division in $q_i\backslash B$ which is introduced). If the division in $q_i\backslash B$ is introduced, then the Proposition 3 implies that the last rule should be $\dfrac{\Gamma\langle B\rangle \to C \quad q_i \to q_i}{\Gamma\langle q_i, q_i\backslash B\rangle \to C\rangle}$, but the first premise is exactly the statement we need.

It remains to inspect the case of the rule $\downarrow\to$. There are two variants of its application, depending on which premise contains the type q_i. The first form of this rule is $\dfrac{\Gamma_1\langle q_i, D\rangle \to C \quad q_i\backslash B, \Pi \to A}{\Gamma_1\langle q_i, (q_i\backslash B)|_k(A \downarrow_k D)\rangle \to C}$. It is impossible because by Corollary 4 the type A should contain q_i which contradicts the definition of $\mathcal{T}'$. The second form is $\dfrac{\Pi, q_i \to A \quad \Gamma_1\langle D, q_i\backslash B\rangle \to C}{\Gamma_1\langle(\Pi, q_i)|_k(A \downarrow_k D), q_i\backslash B\rangle \to C}$. In this case the derivability of the sequent $\Pi, q_i \to A$ by Lemma 6 implies that A contains an occurrence of q_i which again contradicts the definition of $\mathcal{T}'$. All the cases are inspected and the lemma is proved.

We construct the grammar for the language $L \cap R$ just in the way we did it before. The grammar is $G' = \langle \Sigma, Lex', H'\rangle$ where $H' = (q_1\backslash H) \cdot q_f$ and $Lex' = \{\langle a, T_{ij}\rangle \mid \langle a, T\rangle \in Lex,\ \langle q_i, a, q_j\rangle \in \Delta\}$. The fact that this grammar generates exactly the language $L \cap R$ is proved exactly like in standard Lambek calculus. That leads us to the following theorem:

Theorem 2. *The class of languages which do not contain the empty word and are generated by a discontinuous Lambek grammar is closed under intersections with regular languages.*

Actually, the proof can be applied to DL_k-grammars as well. Note that the number of types in the lexicon of the grammar constructed does not exceed $M|Q|^2$, where M is the number of types in the initial Lambek grammar.

4 Conclusion

We have proved that the class of languages, generated by discontinuous Lambek grammars, is closed under intersections with regular languages. That is not the strange fact and the proof is quite simple, though there are not many results of such type for Lambek grammars. The proof works only for the grammars that do not generate the empty word, but the author supposes that extending the construction to them is just a matter of technicalities. The interesting direction is to develop similar constructions to generate the particular examples of non-context-free languages by the means of discontinuous Lambek grammars. This could help us to understand better the comparative generative power of DL-grammars and other formalisms, for example different variants of multiple context-free grammars or conjunctive grammars of Okhotin.

References

1. Aho, A., Ullman, J.: The theory of parsing, translation, and compiling. Prentice-Hall, Inc. (1972)
2. Bar-Hillel, Y., Gaifman, H., Shamir, E.: On categorial and phrase-structure grammars. Bull. Res. Council Israel Sect. F 9F, 1–16 (1960)
3. Kanazawa, M., Salvati, S.: MIX is not a tree-adjoining language. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, pp. 666–674 (2012)
4. Kuznetsov, S.: Lambek grammars with one division and one primitive type. Logic Journal of the IGPL 20(1), 207–221 (2012)
5. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958)
6. Morrill, G., Valentín, O.: On calculus of displacement. In: Proceedings of the 10th International Workshop on Tree Adjoining Grammars and Related Formalisms, pp. 45–52 (2010)
7. Morrill, G., Valentín, O., Fadda, M.: The displacement calculus. Journal of Logic, Language and Information 20(1), 1–48 (2011)
8. Pentus, M.: Lambek grammars are context-free. In: Logic in Computer Science, Proceedings of the LICS 1993, pp. 429–433 (1993)
9. Seki, H., Matsumoto, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science 88(2), 191–229 (1991)
10. Sorokin, A.: Normal Forms for Multiple Context-Free Languages and Displacement Lambek Grammars. In: Artemov, S., Nerode, A. (eds.) LFCS 2013. LNCS, vol. 7734, pp. 319–334. Springer, Heidelberg (2013)

A Count Invariant for Lambek Calculus with Additives and Bracket Modalities*

Oriol Valentín[1], Daniel Serret[2], and Glyn Morrill[1]

[1] Universitat Politècnica de Catalunya
oriol.valentin@gmail.com, morrill@lsi.upc.edu
http://www.lsi.upc.edu/~morrill/
[2] Universitat de Barcelona
daniel.serret@gmail.com

Abstract. The count invariance of van Benthem (1991[16]) is that for a sequent to be a theorem of the Lambek calculus, for each atom, the number of positive occurrences equals the number of negative occurrences. (The same is true for multiplicative linear logic.) The count invariance provides for extensive pruning of the sequent proof search space. In this paper we generalize count invariance to categorial grammar (or linear logic) with additives and bracket modalities. We define by mutual recursion two counts, minimum count and maximum count, and we prove that if a multiplicative-additive sequent is a theorem, then for every atom, the minimum count is less than or equal to zero and the maximum count is greater than or equal to zero; in the case of a purely multiplicative sequent, minimum count and maximum count coincide in such a way as to together reconstitute the van Benthem count criterion. We then define in the same way a bracket count providing a count check for bracket modalities. This allows for efficient pruning of the sequent proof search space in parsing categorial grammar with additives and bracket modalities.

1 Introduction

Van Benthem (1991[16]) showed that a necessary condition for a sequent to be a theorem of the Lambek calculus is that it satisfies a simple count check. Let $\mathcal{P}$ be the set of atoms. Where $P \in \mathcal{P}$, the P-count $\#_P(A)$ of a type A is defined by:

$$\begin{aligned}
&\#_P(P) = 1\\
&\#_P(Q) = 0 \text{ for } Q \in \mathcal{P} - \{P\}\\
&\#_P(A \bullet B) = \#_P(A) + \#_P(B)\\
&\#_P(A \backslash C) = \#_P(C) - \#_P(A)\\
&\#_P(C / B) = \#_P(C) - \#_P(B)
\end{aligned}$$

* Research partially supported by an ICREA Acadèmia 2012 to the third author, and by BASMATI MICINN project (TIN2011-27479-C04-03) and SGR2009-1428 (LARCA). Many thanks to Josefina Sierra and to three Formal Grammar referees for comments and suggestions. Particular thanks to the referee who pointed towards the simplification of the proposal in the appendix which we have used in the main text. Any errors are our own.

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 263–276, 2013.

Let the P-count be extended to configurations by the following, where Λ is the empty configuration:

$$\#_P(A, \Gamma) = \#_P(A) + \#_P(\Gamma)$$
$$\#_P(\Lambda) = 0$$

The count-invariance property is that

$$\vdash \Gamma \Rightarrow A \Longrightarrow \forall P \in \mathcal{P}, \#_P(\Gamma) = \#_P(A)$$

This is proved by induction on sequent proofs.

The count invariance forms the basis of an extensive pruning of the sequent proof search space in categorial parsing-as-deduction/theorem-proving. Every time a subgoal $\Gamma \Rightarrow A$ is generated it can be quickly checked whether $\forall P$, $\#_P(\Gamma) = \#_P(A)$; if not, the subgoal can be discarded at once. Informal experimentation shows that such count-checking, together with focusing normalization (Andreoli 1992[1]; König 1989[6]; Hepple 1990[4]; Hendriks 1993[3]; Morrill 2011[11]) makes a critical difference in efficiency of Lambek calculus sequent proof search parsing-as-deduction.

In this paper we consider the case of the Lambek calculus extended with additives (Lambek 1961[7]; Girard 1987[2]; van Benthem 1991[16]; Morrill 1990[9]; Kanazawa 1992[5]). For linguistic motivation see Morrill (1994[14], ch. 6) or Morrill (2011[15], ch. 7). Our contribution is to define two new counts, $\#_{min}$ and $\#_{max}$, which, for a multiplicative-additive sequent to be a theorem, must be less than or equal to zero and greater than or equal to zero respectively. In the special case that a sequent has no additives, $\#_{min} = \#_{max}$ = the van Benthem count # so that the two inequations together impose the van Benthem criterion, i.e. our generalisation preserves the particular case of the van Benthem count for pure multiplicative sequents. We go on to formulate in addition count invariance for bracket modalities (Morrill 1992[10]; Moortgat 1995[8]).

2 Count Invariance for Additives

Let us consider the sequent formulation of **LA**, the Lambek calculus with additive connectives. We will denote the additive conjunction as $\wedge$ and the additive disjunction as $\vee$. The sequent rules are shown in Figure 1. Cut, of course, is eliminable.

Where m = max or min, we define recursively counts $\#_{max/min,P}$ on types as follows:[1]

$$\#_m(P) = 1$$
$$\#_m(Q) = 0 \text{ for } Q \in \mathcal{P} - \{P\}$$
$$\#_m(A \bullet B) = \#_m(A) + \#_m(B)$$
$$\#_m(A \backslash C) = \#_m(C) - \#_m(A)$$
$$\#_m(C / B) = \#_m(C) - \#_m(B)$$
$$\#_m(A \wedge B) = \overline{m}(\#_m(A), \#_m(B))$$
$$\#_m(A \vee B) = m(\#_m(A), \#_m(B))$$

We extend the counts $\#_{min/max}$ to configurations by the following:

$$\#_m(A, \Gamma) = \#_m(A) + \#_m(\Gamma)$$
$$\#_m(\Lambda) = 0$$

[1] $\overline{max} = min; \overline{min} = \max$. We leave the parameter $P \in \mathcal{P}$ implicit.

$$\frac{}{A \Rightarrow A}\, id \qquad \frac{\Gamma \Rightarrow A \qquad \Delta(A) \Rightarrow B}{\Delta(\Gamma) \Rightarrow B}\, Cut$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta(B) \Rightarrow C}{\Delta(\Gamma, A\backslash B) \Rightarrow C}\, \backslash L \qquad \frac{A, \Delta \Rightarrow B}{\Delta \Rightarrow A\backslash B}\, \backslash R$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta(B) \Rightarrow C}{\Delta(B/A, \Gamma) \Rightarrow C}\, /L \qquad \frac{\Delta, A \Rightarrow B}{\Delta \Rightarrow B/A}\, /R$$

$$\frac{\Delta(A, B) \Rightarrow C}{\Delta(A \bullet B) \Rightarrow C}\, \bullet L \qquad \frac{\Delta \Rightarrow A \qquad \Gamma \Rightarrow B}{\Delta, \Gamma \Rightarrow A \bullet B}\, \bullet R$$

$$\frac{\Delta(B) \Rightarrow C}{\Delta(A \wedge B) \Rightarrow C}\, \wedge L1 \qquad \frac{\Delta(A) \Rightarrow C}{\Delta(A \wedge B) \Rightarrow C}\, \wedge L2$$

$$\frac{\Delta \Rightarrow B \qquad \Delta \Rightarrow C}{\Delta \Rightarrow B \wedge C}\, \wedge R$$

$$\frac{\Delta(A) \Rightarrow C \qquad \Delta(B) \Rightarrow C}{\Delta(A \vee B) \Rightarrow C}\, \vee L$$

$$\frac{\Delta \Rightarrow C}{\Delta \Rightarrow B \vee C}\, \vee R1 \qquad \frac{\Delta \Rightarrow B}{\Delta \Rightarrow B \vee C}\, \vee R2$$

Fig. 1. Rules for the additives and multiplicatives

And we define the counts $\#_{min/max}$ of a sequent by:

$$\#_m(\Delta \Rightarrow A) = \#_m(A) - \#_m(\Delta)$$

Given an arbitrary atomic type P we have the following theorem:

(1) **Theorem** (*Soundness of* **LA** *w.r.t.* $\#^p_{m,P}(A)$)

If $\Delta \Rightarrow A$ is an **LA** provable sequent then:

$$\#_{min}(\Delta \Rightarrow A) \leq 0 \leq \#_{max}(\Delta \Rightarrow A)$$

Proof. By induction on the length of Cut-free **LA** derivations. In the following, i.h. will abbreviate induction hypothesis.

– Axiom case. If $A = P$ then:

$$\begin{array}{ll} \#_{min}(A \Rightarrow A) & = \\ \#_{min}(A) - \#_{min}(A) = 1 - 1 = 0 = 1 - 1 & = \\ \#_{max}(A) - \#_{max}(A) & = \\ \#_{max}(A \Rightarrow A) & \end{array}$$

Otherwise, if A is an atomic type Q different from P we have:

$$\#_{min}(A \Rightarrow A) = 0 - 0 = \#_{max}(A \Rightarrow A)$$

– $\wedge$ left rule:

$$\frac{\Delta(A) \Rightarrow C}{\Delta(A \wedge B) \Rightarrow C} \wedge L$$

We have that:

$$\begin{array}{ll} \#_{min}(\Delta(A \wedge B) \Rightarrow C) & = \\ \#_{min}(C) - (\#_{min}(A \wedge B) + \#_{min}(\Delta(\Lambda))) & = \\ \#_{min}(C) - max(\#_{min}(A), \#_{min}(B)) - \#_{min}(\Delta(\Lambda)) & \leq \\ \#_{min}(C) - \#_{min}(A) - \#_{min}(\Delta(\Lambda)) & = \\ \#_{min}(\Delta(A) \Rightarrow C) & \overset{\text{i.h.}}{\leq} 0 \end{array}$$

On the other hand:

$$\begin{array}{rl} 0 \overset{\text{i.h.}}{\leq} & \#_{max}(\Delta(A) \Rightarrow C) = \\ & \#_{max}(C) - \#_{max}(A) - \#_{max}(\Delta(\Lambda)) \leq \\ & \#_{max}(C) - min(\#_{max}(A), \#_{max}(B)) - \#_{max}(\Delta(\Lambda)) = \\ & \#_{max}(C) - \#_{max}(A \wedge B) - \#_{max}(\Delta(\Lambda)) = \\ & \#_{max}(\Delta(A \wedge B) \Rightarrow C) \end{array}$$

– $\wedge$ right rule:

$$\frac{\Delta \Rightarrow B \qquad \Delta \Rightarrow C}{\Delta \Rightarrow B \wedge C} \wedge R$$

Suppose that $max(\#_{min}(B), \#_{min}(C)) = \#_{min}(B)$. We have that,

$$\begin{array}{ll} \#_{min}(\Delta \Rightarrow B \wedge C) & = \\ \#_{min}(B \wedge C) - \#_{min}(\Delta) & = \\ max(\#_{min}(B), \#_{min}(C)) - \#_{min}(\Delta) & = \\ \#_{min}(B) - \#_{min}(\Delta) & = \\ \#_{min}(\Delta \Rightarrow B) & \overset{\text{i.h.}}{\leq} 0 \end{array}$$

If $max(\#_{min}(B), \#_{min}(C)) = \#_{min}(C)$ we get similarly:

$$\begin{array}{ll} \#_{min}(\Delta \Rightarrow B \wedge C) & = \\ \#_{min}(B \wedge C) - \#_{min}(\Delta) & = \\ max(\#_{min}(B), \#_{min}(C)) - \#_{min}(\Delta) & = \\ \#_{min}(C) - \#_{min}(\Delta) & = \\ \#_{min}(\Delta \Rightarrow C) & \overset{\text{i.h.}}{\leq} 0 \end{array}$$

On the other hand, suppose that $min(\#_{max}(B), \#_{max}(C)) = \#_{max}(B)$

$$\begin{array}{rl} 0 \overset{\text{i.h.}}{\leq} & \#_{max}(\Delta \Rightarrow B) = \\ & \#_{max}(B) - \#_{max}(\Delta) = \\ & min(\#_{max}(B), \#_{max}(C)) - \#_{max}(\Delta) = \\ & \#_{max}(B \wedge C) - \#_{max}(\Delta) = \\ & \#_{max}(\Delta \Rightarrow B \wedge C) \end{array}$$

Similarly, if we have that $min(\#_{max}(B), \#_{max}(C)) = \#_{max}(C)$ we obtain:

$$0 \leq \#_{max}(\Delta \Rightarrow B \wedge C)$$

– $\vee$ left rule:

$$\frac{\Delta(A) \Rightarrow C \qquad \Delta(B) \Rightarrow C}{\Delta(A \vee B) \Rightarrow C} \vee L$$

Suppose that $min(\#_{min}(A), \#_{min}(B)) = \#_{min}(A)$. We have that:

$$\begin{array}{ll} \#_{min}(\Delta(A \vee B) \Rightarrow C) & = \\ \#_{min}(C) - \#_{min}(A \vee B) - \#_{min}(\Delta(\Lambda)) & = \\ \#_{min}(C) - min(\#_{min}(A), \#_{min}(B)) - \#_{min}(\Delta(\Lambda)) & = \\ \#_{min}(C) - \#_{min}(A) - \#_{min}(\Delta(\Lambda)) & = \\ \#_{min}(\Delta(A) \Rightarrow C) & \overset{\text{i.h.}}{\leq} 0 \end{array}$$

Similarly, if we have that $min(\#_{min}(A), \#_{min}(B)) = \#_{min}(B)$ we obtain:

$$\begin{array}{ll} \#_{min}(\Delta(A \vee B) \Rightarrow C) & = \\ \#_{min}(C) - \#_{min}(A \vee B) - \#_{min}(\Delta(\Lambda)) & = \\ \#_{min}(C) - min(\#_{min}(A), \#_{min}(B)) - \#_{min}(\Delta(\Lambda)) & = \\ \#_{min}(C) - \#_{min}(B) - \#_{min}(\Delta(\Lambda)) & = \\ \#_{min}(\Delta(B) \Rightarrow C) & \overset{\text{i.h.}}{\leq} 0 \end{array}$$

On the other hand, if we have $max(\#_{max}(A), \#_{max}(B)) = \#_{max}(A)$:

$$\begin{array}{ll} 0 \overset{\text{i.h.}}{\leq} & \#_{max}(\Delta(A) \Rightarrow C) = \\ & \#_{max}(C) - \#_{max}(A) - \#_{max}(\Delta(\Lambda)) = \\ & \#_{max}(C) - max(\#_{max}(A), \#_{max}(B)) + \#_{max}(\Delta(\Lambda)) \\ & \#_{max}(\Delta(A \vee B) \Rightarrow C) \end{array}$$

Similarly, if we have $max(\#_{max}(A), \#_{max}(B)) = \#_{max}(B)$ we get the desired result.

– $\vee$ right rule:

$$\frac{\Delta \Rightarrow B}{\Delta \Rightarrow B \vee C} \vee R$$

We have that:

$$\begin{array}{ll} \#_{min}(\Delta \Rightarrow B \vee C) & = \\ \#_{min}(B \vee C) - \#_{min}(\Delta) & = \\ min(\#_{min}(B), \#_{min}(C)) - \#_{min}(\Delta) & \leq \\ \#_{min}(B) - \#_{min}(\Delta) & = \\ \#_{min}(\Delta \Rightarrow B) & \overset{\text{i.h.}}{\leq} 0 \end{array}$$

On the other hand:

$$\begin{array}{ll} 0 \overset{\text{i.h.}}{\leq} & \#_{max}(\Delta \Rightarrow B) = \\ & \#_{max}(B) - \#_{max}(\Delta) \leq \\ & max(\#_{max}(B), \#_{max}(C)) - \#_{max}(\Delta) = \\ & \#_{max}(\Delta \Rightarrow B \vee C) \end{array}$$

– / left rule:

$$\frac{\Gamma \Rightarrow A \qquad \Delta(B) \Rightarrow C}{\Delta(B/A, \Gamma) \Rightarrow C}\, /L$$

We have that:

$$\frac{\#_{min}(\Gamma \Rightarrow A) \overset{\text{i.h.}}{\leq} 0 \qquad \#_{min}(\Delta(B) \Rightarrow C) \overset{\text{i.h.}}{\leq} 0}{\#_{min}(C) \underbrace{-(\#_{min}(B) - \#_{min}(A))}_{= -\#_{min}(B/A)} - \#_{min}(\Gamma) - \#_{min}(\Delta(\Lambda)) \leq 0}\ \text{Adding both inequations}$$

Where the last inequation corresponds to

$$\#_{min}(\Delta(B/A, \Gamma) \Rightarrow C) \leq 0$$

On the other hand:

$$\frac{0 \overset{\text{i.h.}}{\leq} \#_{max}(\Gamma \Rightarrow A) \qquad 0 \overset{\text{i.h.}}{\leq} \#_{max}(\Delta(B) \Rightarrow C)}{0 \leq \#_{max}(C) - \#_{max}(\Delta(\Lambda)) - \#_{max}(\Gamma) \underbrace{-(\#_{max}(B) - \#_{max}(A))}_{= -\#_{max}(B/A)}}\ \text{Adding both inequations}$$

Where the last inequation corresponds to:

$$0 \leq \#_{max}(\Delta(B/A, \Gamma) \Rightarrow C)$$

– / right rule

$$\frac{\Delta, A \Rightarrow B}{\Delta \Rightarrow B/A}\, /R$$

We have that:

$$\#_{min}(\Delta, A \Rightarrow B) = \underbrace{\#_{min}(B) - \#_{min}(A)}_{= \#_{min}(B/A)} - \#_{min}(\Delta) \overset{\text{i.h.}}{\leq} 0$$

Where the last inequation corresponds to:

$$\#_{min}(\Delta \Rightarrow B/A) \leq 0$$

On the other hand:

$$0 \overset{\text{i.h.}}{\leq} \#_{max}(\Delta, A \Rightarrow B) = \#_{max}(B) - \#_{max}(A) - \#_{max}(\Delta) = \#_{max}(B/A) - \#_{max}(\Delta)$$

Where the last inequation corresponds to:

$$0 \leq \#_{max}(\Delta \Rightarrow B/A)$$

– \L rule: as /L

– \R rule: as /R

– • left rule:

$$\frac{\Delta(A, B) \Rightarrow C}{\Delta(A \bullet B) \Rightarrow C} \bullet L$$

We have that:

$$\#_{min}(C) - \#_{min}(\Delta(\Lambda)) - \#_{min}(A) - \#_{min}(B) \overset{\text{i.h.}}{\leq} 0$$

Where the last inequation corresponds to:

$$\#_{min}(\Delta(A \bullet B) \Rightarrow C) \leq 0$$

On the other hand:

$$0 \overset{\text{i.h.}}{\leq} \#_{max}(C) - \#_{max}(\Delta(\Lambda)) - \#_{max}(A) - \#_{max}(B)$$

Where the last inequation corresponds to:

$$0 \leq \#_{max}(\Delta(A \bullet B) \Rightarrow C)$$

– • right rule:

$$\frac{\Delta \Rightarrow A \qquad \Gamma \Rightarrow B}{\Delta, \Gamma \Rightarrow A \bullet B} \bullet R$$

We have that:

$$\frac{\#_{min}(A) - \#_{min}(\Delta) \overset{\text{i.h.}}{\leq} 0 \qquad \#_{min}(B) - \#_{min}(\Gamma) \overset{\text{i.h.}}{\leq} 0}{\#_{min}(A \bullet B) - \#_{min}(\Delta) - \#_{min}(\Gamma) \leq 0} \text{ Adding both inequations}$$

Where the last inequation corresponds to:

$$\#_{min}(\Delta, \Gamma \Rightarrow A \bullet B) \leq 0$$

On the other hand:

$$\frac{0 \overset{\text{i.h.}}{\leq} \#_{max}(A) - \#_{max}(\Delta) \qquad 0 \overset{\text{i.h.}}{\leq} \#_{max}(B) - \#_{max}(\Gamma)}{0 \leq \#_{max}(A \bullet B) - \#_{max}(\Delta) - \#_{max}(\Gamma)} \text{ Adding both inequations}$$

Where the last inequation corresponds to:

$$0 \leq \#_{max}(\Delta, \Gamma \Rightarrow A \bullet B)$$

This completes the proof. □

2.1 Exemplification

In this section, by way of example we give some underivable sequents which are falsified by the count check. Let P and Q be two atomic types:

1) $\nvdash_{\mathbf{LA}} P \Rightarrow P \wedge Q$

Consider the count check with respect to P. We have then that:

$$\begin{array}{ll} \#_{max}(P \Rightarrow P \wedge Q) = \\ \#_{max}(P \wedge Q) - 1 & = \\ min(1,0) - 1 & = -1 \not\geq 0 \end{array}$$

Therefore we falsify sequent 1).

2) $\nvdash_{\mathbf{LA}} P \vee Q \Rightarrow P$

Consider the count check with respect to Q. We have then that:

$$\begin{array}{ll} \#_{max}(P \vee Q \Rightarrow P) & = \\ \#_{max}(P) - \#_{max}(P \vee Q) & = \\ 0 - max(0,1) & = -1 \not\geq 0 \end{array}$$

Therefore we falsify sequent 2).

3) $\nvdash_{\mathbf{LA}} P \vee Q \Rightarrow P \wedge Q$

Consider the count check with respect to P. We have then that:

$$\begin{array}{ll} \#_{max}(P \vee Q \Rightarrow P \wedge Q) & = \\ \#_{max}(P \wedge Q) - \#_{max}(P \vee Q) & = \\ min(1,0) - max(1,0) & = 0 - 1 = -1 \not\geq 0 \end{array}$$

Therefore we falsify sequent 3).

4) $\nvdash_{\mathbf{LA}} P \Rightarrow P \bullet P$

Consider the count check with respect to P. We have then that:

$$\begin{array}{ll} \#_{min}(P \Rightarrow P \bullet P) & = \\ \#_{min}(P \bullet P) - \#_{min}(P) & = \\ 2 - 1 & = 1 \not\leq 0 \end{array}$$

Therefore we falsify sequent 4).

3 Count Invariance for Bracket Modalities

In the Lambek calculus with bracket modalities (Morrill 1992[10]; Moortgat 1995[8]) configurations are bracketed; for linguistic applications see Morrill (1994[14], ch. 7)

$$\frac{\Gamma(A) \Rightarrow B}{\Gamma([[]^{-1}A]) \Rightarrow B}\,[]^{-1}L \qquad \frac{[\Gamma] \Rightarrow B}{\Gamma \Rightarrow []^{-1}B}\,[]^{-1}R$$

$$\frac{\Gamma([A]) \Rightarrow B}{\Gamma(\langle\rangle A) \Rightarrow B}\,\langle\rangle L \qquad \frac{\Gamma \Rightarrow B}{[\Gamma] \Rightarrow \langle\rangle B}\,\langle\rangle R$$

Fig. 2. Logical rules for bracket modalities

or Morrill (2011[15], ch. 5). We extend the logic **LA** with bracket modalities, and we denote it **LAb**; configurations may now include brackets. The logical rules for bracket modalities are as shown in Figure 2.

We can define bracket counts $\#_{min/max,[]}$ as follows:[2]

$$\begin{aligned}\#_m(P) &= 0 \text{ for } P \in \mathcal{P}\\ \#_m(\langle\rangle A) &= \#_m(A) + 1\\ \#_m([]^{-1}A) &= \#_m(A) - 1\end{aligned}$$

The clauses for the multiplicative and additive connectives are the same as those given in the previous section. We extend the bracket count to configurations thus:

$$\begin{aligned}\#_m(A, \Gamma) &= \#_m(A) + \#_m(\Gamma)\\ \#_m([\Gamma]) &= \#_m(\Gamma) + 1\\ \#_m(\Lambda) &= 0\end{aligned}$$

(Naturally for an atom P, $\#_{m,P}([\Gamma]) = \#_{m,P}(\Gamma)$.) Where $m \in \{min, max\}$, the min/max-count of a sequent is again:

$$\#_m(\Delta \Rightarrow A) = \#_m(A) - \#_m(\Delta)$$

The soundness theorem (1) extends to bracket modalities.

Proof. Extending the proof of (1) to bracket modalities.

– $\langle\rangle$ left rule:

$$\frac{\Delta([A]) \Rightarrow B}{\Delta(\langle\rangle A) \Rightarrow B}\,\langle\rangle L$$

We have that for $m \in \{min, max\}$:

$$\#_m([A]) = \#_m(\langle\rangle A)$$

It follows that for $m \in \{min, max\}$:

$$\begin{aligned}&\#_m(\Delta([A]) \Rightarrow B) =\\ &\#_m(\Delta(\langle\rangle A) \Rightarrow B)\end{aligned}$$

[2] We leave implicit the reference to [].

And therefore, by i.h.:

$$\#_{min}(\Delta(\langle\rangle A) \Rightarrow B) \leq 0$$
$$0 \leq \#_{max}(\Delta(\langle\rangle A) \Rightarrow B)$$

– $\langle\rangle$ right rule:

$$\frac{\Delta \Rightarrow A}{[\Delta] \Rightarrow \langle\rangle B}\,\langle\rangle R$$

We have that for $m \in \{min, max\}$:

$$\begin{aligned}&\#_m([\Delta] \Rightarrow \langle\rangle A) = \#_m(\langle\rangle A) - \#_m([\Delta]) = \\ &(\#_m(A) + 1) - \#_m(\Delta) - 1 = \\ &\#_m(A) - \#_m(\Delta) = \\ &\#_m(\Delta \Rightarrow A)\end{aligned}$$

It follows that by i.h:

$$\#_{min}([\Delta]) \Rightarrow \langle\rangle B) \leq 0$$
$$0 \leq \#_{max}([\Delta] \Rightarrow \langle\rangle B)$$

– $[]^{-1}$ left rule:

$$\frac{\Delta(A) \Rightarrow B}{\Delta([[]^{-1}A]) \Rightarrow B}\,[]^{-1}L$$

We have that for $m \in \{min, max\}$:

$$\#_m([[]^{-1}A]) = (\#_m(A) - 1) + 1 = \#_m(A)$$

It follows that for $m \in \{min, max\}$:

$$\#_m(\Delta([[]^{-1}A]) \Rightarrow B) = \#_m(\Delta(A) \Rightarrow B)$$

And therefore by i.h.:

$$\#_{min}(\Delta([[]^{-1}A]) \Rightarrow B) \leq 0$$
$$0 \leq \#_{max}(\Delta([[]^{-1}A] \Rightarrow B)$$

– $[]^{-1}$ right rule:

$$\frac{[\Delta] \Rightarrow A}{\Delta \Rightarrow []^{-1}A}\,[]^{-1}R$$

We have that for $m \in \{min, max\}$:

$$\begin{aligned}&\#_m([\Delta] \Rightarrow A) = \\ &\#_m(A) - \#_m(\Delta) - 1 = \\ &(\#_m(A) - 1) - \#_m(\Delta) = \\ &\#_m(\Delta \Rightarrow []^{-1}A)\end{aligned}$$

It follows that by i.h.:

$$\#_{min}(\Delta \Rightarrow []^{-1}A) \leq 0$$
$$0 \leq \#_{max}(\Delta \Rightarrow []^{-1}A)$$

This completes the proof.

□

3.1 Exemplification

We consider some examples of underivable sequents which are falsified by the count invariant extended to bracket modalities. Let N and S be two atomic types.

1) $\nvdash_{\mathbf{LAb}} N, (\langle\rangle N)\backslash S \Rightarrow S$

We have that the following count check with respect to []:

$$\begin{aligned}\#_{min}(N, (\langle\rangle N)\backslash S \Rightarrow S) &= \#_{min}(S) - \#_{min}(N) - \#_{min}((\langle\rangle N)\backslash S)\\ &= \#_{min}(S) - \#_{min}(N) - \#_{min}(S) + \#_{min}(\langle\rangle N)\\ &= \#_{min}(S) - \#_{min}(N) - \#_{min}(S) + (\#_{min}(N) + 1)\\ &= 0 - 0 - 0 + (0 + 1) = 1 \not\leq 0\end{aligned}$$

Therefore we falsify sequent 1).

2) $\nvdash_{\mathbf{LAb}} [[N]], (\langle\rangle N)\backslash S \Rightarrow S$

Consider the count check with repect to []:

$$\begin{aligned}\#_{max}([[N]], (\langle\rangle N)\backslash S \Rightarrow S) &= \#_{max}(S) - \#_{max}([[N]]) - \#_{max}((\langle\rangle N)\backslash S)\\ &= 0 - (1 + 1 + 0) - (0 - (1 + 0))\\ &= -2 + 1 = -1 \not\geq 0\end{aligned}$$

Therefore we falsify sequent 2).

3) $\nvdash_{\mathbf{LAb}} [S, (S\backslash([]^{-1}[]^{-1}S))/S] \Rightarrow S$

Consider the count check with repect to []:

$$\begin{aligned}\#_{min}([S, (S\backslash([]^{-1}[]^{-1}S))/S] \Rightarrow S) &= \#_{min}(S) - (1 + \#_{min}(S) + (\#_{min}(S) - 2) - 2 \cdot \#_{min}(S))\\ &= 0 - (1 - 2) = 1 \not\leq 0\end{aligned}$$

Therefore we falsify sequent 3).

4 Conclusion: Discriminatory Power

Our proposal for count invariance comprises two inequations. These are parameterised by atoms or brackets. If we assume that the likelyhood of satisfying one arbitrary inequation by chance is $1/2$, the likelyhood of satisfying one inequation for n atoms or brackets is $1/2^n$. But if there are two inequations, as in our case, the chance of satisfying the two is $1/2 \times 1/2 = 1/4$, and the probability of satisfying the two equations for n atoms or brackets is $1/4^n$. Thus the discriminatory capacities of one or both of our count invariants together grow with the number of atoms as follows:

n	2^n	4^n
1	2	4
2	4	16
3	8	64
4	16	256
5	32	1024
6	64	4096
7	128	16384
8	256	65536

Clearly the count invariant is sound for multiplicative-additive linear logic since it is a criterion sensitive to occurences and in no way depends on commutativity or non-commutativity. In the same way it extends immediately to the deterministic connectives of the (dis)placement calculus of Morrill, Valentín and Fadda (2011[13]) since these form residuated families like the Lambek connectives. Furthermore we think it is possible to extend it to the nondeterministic discontinuous connectives since these are defined using additives. Finally, we have begun experimenting with implementation of the new count invariant in the context of the categorial parser/theorem-prover CatLog (Morrill 2012[12]).

Appendix

Another count invariant could be defined using mutual recursion with respect to polarities (of types) and $m = max$ or min. Where polarity $p = \bullet$ or $\circ$ represents antecedent (input) and succedent (output) respectively and $m =$ max or min, we define by mutual recursion as follows counts $\#^{\bullet/\circ}_{max/min,P}$ on types, leaving the parameter $P \in \mathcal{P}$ implicit:[3]

$$\begin{aligned}
&\#^p_m(P) = 1\\
&\#^p_m(Q) = 0 \text{ for } Q \in \mathcal{P} - \{P\}\\
&\#^p_m(A \bullet B) = \#^p_m(A) + \#^p_m(B)\\
&\#^p_m(A \backslash C) = \#^p_m(C) - \#^{\overline{p}}_{\overline{m}}(A)\\
&\#^p_m(C / B) = \#^p_m(C) - \#^{\overline{p}}_{\overline{m}}(B)\\
&\#^\circ_m(A \wedge B) = m(\#^\circ_m(A), \#^\circ_m(B))\\
&\#^\bullet_m(A \wedge B) = \overline{m}(\#^\bullet_m(A), \#^\bullet_m(B))\\
&\#^\circ_m(A \vee B) = \overline{m}(\#^\circ_m(A), \#^\circ_m(B))\\
&\#^\bullet_m(A \vee B) = m(\#^\bullet_m(A), \#^\bullet_m(B))
\end{aligned}$$

We extend the counts $\#^\bullet_{min/max}$ to configurations by the following:

$$\begin{aligned}
&\#^\bullet_m(A, \Gamma) = \#^\bullet_m(A) + \#^\bullet_m(\Gamma)\\
&\#^\bullet_m(\Lambda) = 0
\end{aligned}$$

And we define the counts $\#_{min/max}$ of a sequent by:

$$\#_m(\Delta \Rightarrow A) = \#^\bullet_{\overline{m}}(\Delta) - \#^\circ_m(A)$$

[3] $\overline{\bullet} = \circ; \overline{\circ} = \bullet; \overline{max} = min; \overline{min} = \max$.

(2) **Lemma**
The following equality holds:

$$\#^{\bullet}_{\underline{m}}(A) = \#^{\circ}_{m}(A)\ (\star)$$

Proof. By induction on the complexity of **LA** types (as usual, i.h. abbreviates induction hypothesis):

- Atomic case: obvious.
- Product case: obvious (using i.h.).
- Slashes. Consider / (\ is completely similar):

$$\#^{\bullet}_{\underline{m}}(C/A) = \#^{\bullet}_{\underline{m}}(C) - \#^{\circ}_{m}(A)$$
$$\#^{\circ}_{m}(C/A) = \#^{\circ}_{m}(C) - \#^{\bullet}_{\underline{m}}(A)$$

  By i.h. $\#^{\bullet}_{\underline{m}}(C) = \#^{\circ}_{m}(C)$ and $\#^{\circ}_{m}(A) = \#^{\bullet}_{\underline{m}}(A)$, whence $\#^{\bullet}_{\underline{m}}(C/A) = \#^{\circ}_{m}(C/A)$.
- Conjunction:

$$\#^{\bullet}_{\underline{m}}(A \wedge B) = m(\#^{\bullet}_{\underline{m}}(A), \#^{\bullet}_{\underline{m}}(B))$$
$$\#^{\circ}_{m}(A \wedge B) = m(\#^{\circ}_{m}(A), \#^{\circ}_{m}(B))$$

  By i.h. $\#^{\bullet}_{\underline{m}}(A) = \#^{\circ}_{m}(A)$ and $\#^{\circ}_{m}(B) = \#^{\bullet}_{\underline{m}}(B)$, whence $\#^{\bullet}_{\underline{m}}(A \wedge B) = \#^{\circ}_{m}(A \wedge B)$.
- Disjunction: similar to the case of conjunction.

□

This count invariant satisfies also the soundness theorem (1). By using the previous lemma (2) we can almost mimick the proof from (1). The definition of count invariant we present in this appendix has turned out to be interesting for an ongoing research on a count invariant extended to the exponential modality of linear logic ! (Girard 1987[2]).

References

1. Andreoli, J.M.: Logic programming with focusing in linear logic. Journal of Logic and Computation 2(3), 297–347 (1992)
2. Girard, J.-Y.: Linear logic. Theoretical Computer Science 50, 1–102 (1987)
3. Hendriks, H.: Studied flexibility. Categories and types in syntax and semantics. PhD thesis, Universiteit van Amsterdam. ILLC, Amsterdam (1993)
4. Hepple, M.: Normal form theorem proving for the Lambek calculus. In: Karlgren, H. (ed.) Proceedings of COLING, Stockholm (1990)
5. Kanazawa, M.: The Lambek calculus enriched with additional connectives. Journal of Logic, Language and Information 1, 141–171 (1992)
6. König, E.: Parsing as natural deduction. In: Proceedings of the Annual Meeting of the Association for Computational Linguistics, Vancouver (1989)
7. Lambek, J.: On the Calculus of Syntactic Types. In: Jakobson, R. (ed.) Structure of Language and its Mathematical Aspects, Proceedings of the Symposia in Applied Mathematics XII, pp. 166–178. American Mathematical Society, Providence (1961)
8. Moortgat, M.: Multimodal linguistic inference. Journal of Logic, Language and Information 5(3,4), 371–401 (1996); Also in Bulletin of the IGPL 3(2,3), 371–401 (1995)
9. Morrill, G.: Grammar and Logical Types. In: Stockhof, M., Torenvliet, L. (eds.) Proceedings of the Seventh Amsterdam Colloquium, pp. 429–450 (1990); Barry, G., Morrill, G.: Studies in Categorial Grammar, Edinburgh Working Papers in Cognitive Science, vol. 5, pp. 127–148 (1990); Revised version published as Grammar and Logic. Theoria, LXII 3, 260–293 (1996)

10. Morrill, G.: Categorial Formalisation of Relativisation: Pied Piping, Islands, and Extraction Sites. Technical Report LSI-92-23-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya (1992)
11. Morrill, G.: Logic Programming of the Displacement Calculus. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS, vol. 6736, pp. 175–189. Springer, Heidelberg (2011)
12. Morrill, G.: CatLog: A Categorial Parser/Theorem-Prover. In: LACL 2012 System Demonstrations, Logical Aspects of Computational Linguistics 2012, Nantes, pp. 13–16 (2012)
13. Morrill, G., Valentín, O., Fadda, M.: The Displacement Calculus. Journal of Logic, Language and Information 20(1), 1–48 (2011), doi:10.1007/s10849-010-9129-2.
14. Morrill, G.V.: Type Logical Grammar: Categorial Logic of Signs. Kluwer Academic Publishers, Dordrecht (1994)
15. Morrill, G.V.: Categorial Grammar: Logical Syntax, Semantics, and Processing. Oxford University Press, New York and Oxford (2011)
16. van Benthem, J.: Language in Action: Categories, Lambdas, and Dynamic Logic. Number 130 in Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam (1991); (revised student edition printed in 1995 by the MIT Press)

Some Higher Order Functions on Binary Relations

R. Zuber*

Rayé des cadres du CNRS, Paris, France
Richard.Zuber@linguist.univ-paris-diderot.fr

Abstract. Some higher order functions needed in linguistic semantics are studied. Such functions take binary relations or sets and binary relations as arguments and give sets of type $\langle 1 \rangle$ quantifiers as output. Formal constraints on such functions are generalisations of similar constraints holding for generalised quantifiers and "simple" functions on binary relations. Some theorems concerning such constraints, in particular (generalised) conservativity and cardinality, are proved and some relationships with (Frege) irreducible quantifiers are indicated.

1 Introduction

Functions taking binary relations as arguments are needed in (linguistic) semantics simply because transitive verb phrases (TVPs) are very naturally interpreted by binary relations. Thus, to mention the simplest case, functions denoted by (nominal) direct objects (which form verb phrases (VPs) with TVPs) take binary relations and give as result denotations of VPs. These functions have been investigated in connection with the study of type $\langle 1 \rangle$ quantifiers. Indeed, if type $\langle 1 \rangle$ quantifiers are typically denoted by noun phrases (NPs) and typical direct objects are NPs, then it is natural to allow type $\langle 1 \rangle$ quantifiers to apply also to binary relations in order to interpret VPs obtained by the application of the direct object NP to a TVP. Technically (see below) this is done by extending their domain of application. Extended in that way, type $\langle 1 \rangle$ quantifiers are arity reducers: they apply to n-ary relations and give n-1-ary relations as result and in particular they apply to binary relations and give sets as results.

There are obviously other means to obtain syntactically complex VPs. The analysis presented here is meant eventually to describe the semantics of sentences of the form given in (1):

(1) NP TVP GNP

In this schema, GNP is a generalised noun phrase. GNPs are linguistic objects that can play the role of syntactic arguments of TVPs. So "ordinary" NPs or DPs (determiner phrases) are GNPs. However there are GNPs which differ from "ordinary" NPs in that they cannot play the role of all verbal arguments; in

* Thanks to Makoto Kanazawa for some important remarks and to Ross Charnock for the usual help with English.

G. Morrill and M.-J. Nederhof (Eds.): FG 2012/2013, LNCS 8036, pp. 277–291, 2013.

particular they cannot occur in subject position. I will be interested in the formal properties of functions denoted by such GNPs.

Furthermore, GNPs can themselves be syntactically complex, and have the form GDet(CN), where CN is a common noun and GDet is a generalised determiner, that is an expression which takes CNs as arguments and gives a GNP as a result. So I will also be interested in functions which are denotations of GDets.

Since CNs are supposed to denote sets and TVPs are supposed to denote binary relations, I will also study some functions which take (binary) relations as arguments or sets and binary relations as arguments.

Depending on the type of output one can distinguish two classes of functions interpreting GNPs and GDets: (1) *simple functions*, which give sets (of individuals) as output and (2) *higher order* functions, which have sets of type $\langle 1\rangle$ quantifiers as output.

Simple functions are needed to interpret sentences like (2) or (3):

(2) Leo hates himself.

(3) Leo hates every philosopher including himself.

In (2) *himself* is a syntactic argument of *hates* with which it forms a VP. It is thus a GNP and it denotes the function $SELF$ defined as $SELF(R) = \{x : \langle x, x\rangle \in R\})$ which is an arity reducer. It can be shown (Keenan 2007) that $SELF$ is not an extension of a type $\langle 1\rangle$ quantifier and thus is not denoted by a "typical" NP.

In (3) we have an (anaphoric) GDet *every...including himself* which also denotes a simple function (cf. Zuber 2010b). Similarly simple functions are needed for the semantics of generalised comparative and superlative determiners (Zuber 2011a).

Simple functions have been already studied to various degrees (Keenan and Westerståhl 1997, Keenan 2007, Zuber 2010b, Zuber 2011a). This paper is devoted to a preliminary study of higher order functions needed in linguistic semantics. As will be shown in more detail below, such functions are needed to interpret reciprocal constructions, constructions with *the same* and the so-called cumulative readings of some quantifiers. Moreover, such functions give rise to Frege irreducible quantifiers.

2 Formal Preliminaries

We will consider binary relations and functions over universe E which is supposed to be finite. If a function takes only a binary relation as argument, its type is noted $\langle 2 : \tau\rangle$, where τ is the type of the output; if a function takes a set and a binary relation as arguments, its type is noted $\langle 1, 2 : \tau\rangle$. If $\tau = 1$ then the output of the function is a set of individuals and thus the type of the function is $\langle 2 : 1\rangle$. For instance the function $SELF$ is of this type. The case we will basically consider here is when τ corresponds to a set of type $\langle 1\rangle$ quantifiers and thus τ equals, in Montagovian notation, $\langle\langle\langle e, t\rangle t\rangle t\rangle$. In short, the type of such functions

will be noted either $\langle 2:\langle 1\rangle\rangle$ (functions from binary relations to sets of type $\langle 1\rangle$ quantifiers)) or $\langle 1,2:\langle 1\rangle\rangle$ (functions from sets and binary relations to sets of type $\langle 1\rangle$ quantifiers).

Let R be a binary relation. Then : $dom(R) = \{x : \exists_y \langle x, y\rangle \in R\}$ and $rg(R) = \{x : \exists_y \langle y, x\rangle \in R\}$. Furthermore, for any $a \in E$, $aR = \{x : \langle a, x\rangle \in R\}$ and $Ra = \{x : \langle x, a\rangle \in R\}$. The relation R^{-1} is the converse of R and the relation R^S is the maximal symmetric relation included in R, that is $R^S = R \cap R^{-1}$. A type $\langle 2:1\rangle$ or type $\langle 2:\langle 1\rangle\rangle$ function F is convertible iff $F(R) = F(R^{-1})$. Relation I is defined as $I = \{\langle x, x\rangle : x \in E\}$. The relation R^t is the transitive closure of the relation R, that is the smallest transitive relation in which R is included.

Basic type $\langle 1\rangle$ quantifiers are functions from sets (sub-sets of E) to truth-values. In this case they are denotations of subject NPs. However, NPs can also occur in oblique positions and in this case their denotations do not take sets (denotations of verb phrases) as arguments but rather denotations of intransitive verb phrases, that is relations, as arguments. To account for this eventuality it has been proposed to extend the domain of application of basic type $\langle 1\rangle$ quantifiers so that they apply to n-ary relations and act as arity reducers, that is have as output an (n–1)-ary relation (see Keenan 1987). Since we are basically interested in binary relations, the domain of application of basic type $\langle 1\rangle$ quantifiers will be extended by adding to their domain the set of binary relations. In this case the quantifier Q can act as a "subject" quantifier or a "direct object" quantifier giving rise to the *nominative case extension* Q_{nom} and *accusative case extension* Q_{acc} respectively. They are defined as follows (Keenan 1987, Keenan and Westerstahl 1997):

Definition 1. *For each type* $\langle 1\rangle$ *quantifier* Q, $Q_{nom}(R) = \{a : Q(Ra) = 1\}$.

Definition 2. *For each type* $\langle 1\rangle$ *quantifier* Q, $Q_{acc}(R) = \{a : Q(aR) = 1\}$.

From now on $Q_{nom}(R)$ will be noted $Q(R)$. Nominative and accusative extensions can thus be considered as functions from binary relations to sets. From now on, by type $\langle 1\rangle$ quantifiers I will mean basic type $\langle 1\rangle$ quantifiers as well as their nominative and accusative extensions.

Given that type $\langle 1\rangle$ quantifiers and their arguments form Boolean algebras, every quantifier Q has its Boolean complement, denoted by $\neg Q$, and its post-complement $Q\neg$, defined as follows: $Q\neg = \{P : P \subseteq E \wedge P' \in Q\}$ (where P' is the Boolean complement of P). The dual Q^d of the quantifier Q is, by definition, $Q^d = \neg(Q\neg) = (\neg Q)\neg$. A quantifier Q is self-dual iff $Q = Q^d$. These definitions work also for extended type $\langle 1\rangle$ quantifiers. It easy to see for instance that $\neg(Q_{acc}) = (\neg Q)_{acc}$ and $(Q^d)_{acc} = (Q_{acc})^d$. A type $\langle 1\rangle$ quantifier Q is *positive* iff $Q(\emptyset) = 0$ and Q is atomic iff it contains exactly one element that is if $Q = \{A\}$ for some $A \subseteq E$.

A special class of type $\langle 1\rangle$ quantifiers is formed by *individuals*, that is ultrafilters generated by an element of E. Thus I_a is an individual (generated by $a \in E$) iff $I_a = \{X : a \in X\}$. Ultrafilters are (special) filters. A (principal) filter generated by the set $A \subseteq E$ is the following quantifier: $Ft(A) = \{X : X \subseteq E \wedge A \subseteq X\}$.

One property that we will use is the property of *living on*. The basic type $\langle 1\rangle$ quantifier lives on the set A (where $A \subseteq E$) iff for all $X \subseteq E$, $Q(X) = Q(X \cap A)$. If E is finite then there is always the smallest set on which a quantifier Q lives: it is the meet of all sets on which Q lives. The fact that A is the smallest set on which the quantifier Q lives will be noted $Li(Q, A)$. If $A \in Q$ then A is called the witness set of Q: $A = Wt(Q)$. The quantifier Q is called *plural*, noted $Q \in PL$, iff $\exists_{a,b\in E}$ such that $Q \subseteq I_a \cap I_b$.i

Functions from pairs of sets to truth-values or binary relations between sets are type $\langle 1, 1\rangle$ quantifiers. In NLs they are denoted by (unary) determiners and in this case they obey various constraints. Recall first the constraint of conservativity for type $\langle 1, 1\rangle$ quantifiers:

Definition 3. *$F \in CONS$ iff $F(X, Y) = F(X, X \cap Y)$ for any $X, Y \subseteq E$*

Conservative quantifiers have two important sub-classes: intersective and co-intersective quantifiers (Keenan 1993): a type $\langle 1, 1\rangle$ quantifier F is intersective (resp. co-intersective) iff $F(X_1, Y_1) = F(X_2, Y_2)$ whenever $X_1 \cap Y_1 = X_2 \cap Y_2$ (resp. $X_1 \cap Y_1' = X_2 \cap Y_2'$). Furthermore, a type $\langle 1, 1\rangle$ quantifier F is cardinal (resp. co-cardinal) iff $F(X_1, Y_1) = F(X_2, Y_2)$ whenever $|X_1 \cap Y_1| = |X_2 \cap Y_2|$ (resp. $|X_1 \cap Y_1'| = |X_2 \cap Y_2'|$). For instance numerals denote cardinal quantifiers and "exceptive numerals" like *every... but n* denote co-cardional quantifiers. .

All the above properties of quantifiers can be generalised so that they apply to simple and higher order functions. The following definitions will be used in what follows (cf. Zuber 2010a):

Definition 4. *A function F of type $\langle 1, 2 : \tau\rangle$ is conservative iff $F(X, R) = F(X, (E \times X) \cap R)$*

Definition 5. *A type $\langle 1, 2 : \tau\rangle$ function is intersective iff $F(X_1, R_1) = F(X_2, R_2)$ whenever $(E \times X_1) \cap R_1 = (E \times X_2) \cap R_2$.*

Definition 6. *A type $\langle 1, 2 : \tau\rangle$ function is cardinal iff $F(X_1, R_1) = F(X_2, R_2)$ whenever $\forall y(|X_1 \cap yR_1| = |X_2 \cap yR_2|)$.*

One can notice that cardinal functions are intersective and intersective functions are conservative.

As in the case for type $\langle 1, 1\rangle$ quantifiers it is possible to give other, equivalent, definitions of conservativity, intersectivity and co-intersectivity of type $\langle 1, 2 : \tau\rangle$ functions. I will use the definition of intersectivity given by:

Proposition 1. *A type $\langle 1, 2 : \tau\rangle$ function F is intersective iff $F(X, R) = F(E, (E \times X) \cap R)$.*

Observe that most of the above definitions do not depend on the type τ of the result of the application of the function. So obviously they can be used with higher order functions. Type $\langle 2 : 1\rangle$ functions can also be (predicate or argument) invariant and invariance is a property depending on the type of the output of the function. Thus (see Keenan and Westerståhl 1997) a type $\langle 2 : 1\rangle$

function F is predicate invariant iff $a \in F(R) \equiv a \in F(S)$ whenever $aR = aS$. For instance the function $SELF$ is predicate invariant. Similarly, a type $\langle 2:1\rangle$ function F is argument invariant iff $a \in F(R) \equiv b \in F(R)$ whenever $aR = bR$. Functions interpreting some superlatives or comparatives are argument invariant (see Zuber 2011a). The following definitions are generalisations of predicate and argument invariance applying to type $\langle 2:\langle 1\rangle\rangle$ functions:

Definition 7. *A type $\langle 2:\langle 1\rangle\rangle$ function F satisfies **HPI** (higher order predicate invariance) iff for any positive type $\langle 1\rangle$ quantifier Q, any $A \subseteq E$, any binary relations R, S, if $A = Wt(Q)$ and $Ft(A)R = Ft(A)S$ then $Q \in F(R)$ iff $Q \in F(S)$.*

Definition 8. *A type $\langle 2:\langle 1\rangle\rangle$ function F satisfies **HAI** (higher order argument invariance) iff for any positive type $\langle 1\rangle$ quantifiers Q_1 and Q_2, any $A, B \subseteq E$, any binary relation R, if $A = Wt(Q_1)$ and $B = Wt(Q_2)$, and $Ft(A)R = Ft(B)R$ then $Q_1 \in F(R)$ iff $Q_2 \in F(R)$.*

To show that a function does not satisfy the **HAI** the following obvious proposition may be used:

Proposition 2. *Let F be a type $\langle 2:\langle 1\rangle\rangle$ function which satisfies **HAI** and let $R = E \times C$, for $C \subseteq E$ arbitrary. Then for any $X \subseteq E$, $Ft(X) \in F(R)$ or for any X, $Ft(X) \notin F(R)$*

I will also use a generalisation of argument invariance applying to type $\langle 1,2:\langle 1\rangle\rangle$ functions which interpret some GDets. It is defined as follows:

Definition 9. *A type $\langle 1,2:\langle 1\rangle\rangle$ function F satisfies **D1HAI** (higher order argument invariance for unary determiners) iff for any positive type $\langle 1\rangle$ quantifiers Q_1 and Q_2, any $A, B \subseteq E$, any binary relation R, if $A = Wt(Q_1)$ and $B = Wt(Q_2)$, and $Ft(A)R \cap X = Ft(B)R \cap X$ then $Q_1 \in F(X, R)$ iff $Q_2 \in F(X, R)$.*

3 Some Higher Order Functions

I what follows I will show that some GNPs in NLs denote higher order functions. Before doing this a comment about the categorial status of objects involved is in order. Recall that GNPs apply to TVPs and give VPs as result. So what is the category of such VPs. Ignoring directionality, the subject NPs in the constructions we are interested in are of the category $S/(S/NP))$. This means that, in order to avoid type mismatch, verb phrases must be raised and have the category $S/(S/(S/NP))$. Then their denotational type is $\langle\langle\langle e,t\rangle t\rangle t\rangle$. Consequently, sentences of the form (1) are true iff the quantifier denoted by the NP is an element of the set denoted by $TVP\ GNP$.

Of course, any type $\langle 2:1\rangle$ function whose output is denoted by a VP can be lifted to the type $\langle\langle\langle e,t\rangle t\rangle t\rangle$ function. This is in particular the case with the acusative and nominative extensions of a type $\langle 1\rangle$ quantifier. For instance the

accusative extension of a type $\langle 1\rangle$ quantifier can be lifted to type $\langle\langle\langle e,t\rangle t\rangle t\rangle$ function in the way indicated in (4). Such functions will be called *accusative lifts*. More generally iff F is a type $\langle 2:1\rangle$ function, its lift F^L, a type $\langle 2:\langle 1\rangle\rangle$ function, is defined in (5):

(4) $Q^L_{acc}(R) = \{Z : Z(Q_{acc}(R)) = 1\}$.

(5) $F^L(R) = \{Z : Z(F(R)) = 1\}$

The variable Z above runs over the set of type $\langle 1\rangle$ quantifiers.

There are of course genuine type $\langle\langle\langle e,t\rangle t\rangle t\rangle$ (or type $\langle 2:\langle 1\rangle\rangle$ in our notation) functions, that is such that they are not lifts of simple type $\langle 2:1\rangle$ functions. I will now briefly present three classes of type $\langle 2:\langle 1\rangle\rangle$ which are denoted by GNPs and show that they are not accusative lifts. These are, first, functions denoted by reciprocals, possibly Booleanly complex (Zuber 2012a), second, functions involved in the interpretation of constructions with *the same* and, third, functions needed to interpret cummulative readings.

Reciprocal sentences have a general form given in (6) and an example of such a reciprocal sentence is given in (7):

(6) $NP\ TVP\ EACH\text{-}OTHER$.

(7) Three philosophers hit each other.

The semantics of reciprocal sentences is a complex matter (as shown for instance in Dalrymple *et al.* 1998). In fact there does not seem that there is a general agreement concerning the data and the interpretation of reciprocal constructions (cf. Beck 2000). The situation is even more complex because in many languages reflexive markers on verb can give rise to reciprocal interpretation of the action associated with the verb marked in that way, without necessarily excluding the reflexive interpretation. For that reason we can start with the function given in (8):

(8) $RFL\text{-}RECIP(R) = \{Q : A = Wt(Q) \wedge Q(Dom((A \times A) \cap R^S) = 1\}$

Informally, this function can be considered as denotation of an anaphor like *each other or oneself or themselves*. In other words it does not make *a priori* a distinction between "purely" reflexive and "purely" reciprocal interpretation, as apparently it happens in many languages. Observe in particular that individuals can be in the output of this function. Furthermore, the meet of two individuals can be in the output of this function even if they are in the relation R with themselves only.

The following function excludes the "reflexive part" and interprets purely reciprocal anaphors (in their strong logical reading, with "full" reciprocity):

(9) $LEA(R) = \{Q : Li(Q.A) \wedge Q \in PL \wedge Q(Dom((A \times A) \cap (R \cap R^{-1}) \cap I')) = 1\}$, where I' is the complement of the identity relation I.

The function LEA can Booleanly combine with other, possibly lifted, higher order functions and give a "complex" higher order function. Such complex functions are needed to interpret sentences in which *each other* is modified by a categorially polyvalent particle (like *only, also at least, even*) or is a part of a Boolean compound. The following examples illustrate some of such cases :

(10) Leo and Lea hug each other and most teachers.

(11) Most philosophers hate each other and themselves.

(12) Two monks hug only/even each other.

Since the functions we are considering form a Boolean algebra, the higher order functions interpreting Boolean compounds with *each other* are easy to define. As an illustration, in (13) we have the function which interprets the GNP *each other and themselves* in (11) and in (14) - the function which can be used to interprets the GNP which is a conjunction of *each other* and an NP (in these examples the variable Z runs over the set of type $\langle 1\rangle$ quantifiers:

(13) $LEA\text{-}SELF(R) = LEA(R) \cap SELF^L(R)$

(14) $LEA_{Qconj}(R) = LEA(R) \cap Q^L_{acc}(R)$

The above functions are based on the relation R^S. Sentences in (15) have somewhat illogical interpretation. Functions corresponding to these interpretations are given in (16):

(15) a. Five students followed each other.
b. All pupils followed each other and two teachers.

(16) a. $ILEA(R) = \{Z : \exists_{A\subseteq E}(Li(Z, A) \wedge A \times A \cap I' \subseteq R^t\}$
b. $ILEA_{Qconj}(R) = ILEA(R) \cap Q^L_{acc}(R)$

Let us see now some constraints on the above functions. First we have:

Proposition 3. *Let* $F \in \{RFL\text{-}RECIP, LEA, ILEA\}$ *and* $R = S^{-1}$. *Then* $F(R) = F(S)$

Proof. We prove the above property only for $ILEA$. Suppose *a contrario* that $Q \in ILEA(R)$ and $Q \notin ILEA(S)$. This means that for A such that $Li(Q, A)$, $(A \times A) \cap I' \subseteq R^t$ and $(A \times A) \cap I' \not\subseteq S^t$. Thus for some $a \neq b$, $a, b \in A$ we have $\langle a, b\rangle \in R^t$ and $\langle b, a\rangle \in R^t$ and $\langle a, b\rangle \notin S^t$. Consequently $\langle b, a\rangle \notin R$. But then, since R^t is the smallest transitive relation containing R, for some $c \in E$ we have $\langle b, c\rangle \in R$ and $\langle c, a\rangle \in R$ and thus $\langle c, b\rangle \in S$ and $\langle a, c\rangle \in S$. Hence $\langle a, b\rangle \in S^t$. Contradiction. □

Proposition 3 has an interesting consequence: since $R = (R^{-1})^{-1}$, it follows from proposition 3 that functions $RFL\text{-}RECIP, LEA$ and $ILEA$ are convertible.

The above properties do not hold for complex higher order functions that is functions denoted by syntactically complex reciprocals. For higher order functions based on the relation R^S the following proposition holds:

Proposition 4. *Let* $F \in \{RFL\text{-}RECIP, LEA, LEA_Q\}$, $R = S^{-1}$ *and* $Dom(R) = Dom(S)$. *Then* $F(R) = F(S)$.

The above functions satisfy also predicate invariance:

Proposition 5. *If* $F \in \{RFL\text{-}RECIP, LEA, LEA_Q\}$ *then* F *satisfies* ***HPI***

Proof. We prove only that $RFL\text{-}RECIP$ satisfies **HPI**. Suppose that $A = Wt(Q)$ and that $Q \in REF\text{-}RECIP(R)$. We have to show that if for some binary relation S (i) holds (i): $\forall_{x \in A}(xR = xS)$ then $Q \in RFL\text{-}RECIP(S)$. Given the definition of $RFL\text{-}RECIP$ this happens when $Q(Dom((A \times A) \cap (S \cap S^{-1}) = 1$. But if (i) holds then $(A \times A) \cap (R \cap R^{-1}) = (A \times A) \cap (S \cap S^{-1})$. Hence $Q \in RFL\text{-}RECIP(S)$. □

Interestingly, the above functions do not satisfy argument invariance:

Proposition 6. *If* $F \in \{RFL\text{-}RECIP, LEA, LEA_Q\}$ *then* F *does not satisfy* ***HAI***

Proof. We prove only that the function $RFL\text{-}RECIP$ does not satisfy **HAI**. Given its definition in (8) we can see that for $C \subseteq E$ arbitrary , for any C_1 such that $C \subseteq C_1$ we have $Ft(C_1) \notin RFL\text{-}RECIP(E \times C)$ and for any $C_2 \subseteq C$ we have $Ft(C_2) \in RFL\text{-}RECIP(E \times C)$. Hence, given proposition 2, $RFL\text{-}RECIP$ does not satisfy **HPI**. □

There are clear cases of higher order functions having sets and relations as arguments. They are involved in the semantics of constructions with *the same* and *different*. Consider the following examples:

(17) a. Leo and Lea read the same books.
b. Most teachers/every teacher/no two students read the same books.

There are some restrictions on the subject NPs which can occur in sentences like the above. I will consider that they have to denote a plural type $\langle 1 \rangle$ quantifier. The generalised determiner *the same* takes the common noun *books* as argument and forms a GNP. As example (17b) shows this GNP applies to a TVP and gives a VP of the category $S/(S/(S/NP))$. Consequently *the same* denotes a type $\langle 1.2 : \tau \rangle$ function, where τ is, in Montagovian notation, of type $\langle\langle\langle e, t\rangle t\rangle t\rangle$.

To obtain the description of the function denoted by *the same* in the above examples the following observation is useful: this function is fixed by the complement of its relational argument: $F(X, R) = F(X, R')$. Thus if (18a) is true then the set of books that Leo and Lea did not read is also the same: (18a) and (18b) have the same truth value:

(18) a. The books that Leo and Lea read are the same.
b. The books that Leo and Lea did not read are the same.

Furthermore, the union of the set of books read by Leo and Lea with the set of books not read by Leo and Lea equals the set of all books. In other words,

there is a set of books such that every book from this set was read by both Leo and Lea and no book from the complement set (of books) was read by Leo or Lea. This observation leads directly to the description of the needed function: sentence of the form given in (19) is true iff the quantifier denoted by NP is an element of the set $SAME(X, R)$ defined in (20), where X is the denotation of the CN and R the denotation of TVP:

(19) $NP\ TVP\ THE\text{-}SAME(CN)$.

(20) $SAME(X, R) = \{Z : Z \in PL \wedge \exists_{A \subseteq X}(A \subseteq QR) \wedge ((X \cap A') \subseteq QR)\}$.

We need also functions expressing "numerical sameness" to analyse examples like (21a). One can suppose that in this example "exactly the same number" is involved and thus that (21a) cannot mean, with the subject *Leo and Lea*, something like (21b). The corresponding function needed to interpret (21a) is given in (22), where $n(X)$ is a type $\langle 1 \rangle$ quantifier such that $n(X)(Y) = 1$ iff $|X \cap Y| = n$ and $N^+ = N \cup \{0\}$:

(21) a. Leo and Lea/most students/two teachers read the same number of books.

b. Leo and Lea read the same number of books and, in addition, Leo read two more.

(22) $SAME\text{-}N(X, R) = \{Z : \exists_{n \in N^+} Z((n(X)_{acc}(R) = 1\}$

Both functions, in (20) and in (22), have their Boolean complements. They are denoted, respectively, by *different* and *different number number of*. For instance the function $DIFF\text{-}N$ given in (24b) is needed to interpret the sentence in (24a):

(23) a. Leo and Lea read a different number of books.

b. $DIFF\text{-}N(X, R) = \{Z : \forall_{n \in N^+} Z((n(X)_{acc}(R) = 0\}$.

The example in (21a) should be distinguished from the one in (24a). Observe that (24a) does not entail (17a) given that both (24a) and (24b) can be true at the same time. What is interesting, however, in connection with (24b), is the fact that the total number of books read by both Leo and Lea is the same: as (24b) shows the number of different books read by both cannot be different for each reader. In other words, the number of books that they did not read is also the same. Thus the function needed to analyse (24a) is given in (25), where (24a) is taken in the "exactly" reading:

(24) a. Leo and Lea read the same 5 books.

b. Leo and Lea read the same 5 books and 7 different ones.

(25) $SAME_n(X, R) = \{Z : |Z(R) \cap X| = n \wedge \exists_{m \geq n} Z((m(X)_{acc}(R) = 1\}$

Thus, even if (25a) does not entail (16a), it is true that (25a) entails (22a) (with *Leo and Lea* as grammatical subject).

Let us see some formal properties of the above functions related to the interpretation of *the same* in various linguistic contexts. It is easy to show that the functions in (19) (and thus in (20), in (21)), in (23), in (24b) and in (26), are all conservative. Moreover, using Proposition 1 it is easy to prove:

Proposition 7. *The function* $SAME(X, R)$ *is intersective.*

We also have the following proposition:

Proposition 8. *The function SAME-N is cardinal.*

Proof. We have to show that $SAME\text{-}N(X_1, R_1) = SAME\text{-}N(X_2, R_2)$ if (i): $\forall_{y \in E}(|X_1 \cap yR_1| = |X_2 \cap yR_2|)$ holds. But if (i) holds then $(n(X_1)_{acc}(R_1) = \{y : |X_1 \cap yR_1| = n\} = \{y : |X_2 \cap yR_2| = n\} = (n(X_2)_{acc}(R_2)$ and we get the needed equality by the replacement of equals. □

Consider finally example (26a) with the intended reading entailing (26b). To interpret this sentence we need the function given in (27):

(26) a. Leo and Lea read only the same 5 books.
b. Both Leo and Lea read exactly 5 books.

(27) $ONLY\text{-}SAME_n(X, R) = \{Z : Z \in SAME_n(X, R) \wedge Z^d MORE_n(X)_{acc} R = 0\}$, where $MORE_n(X) = \{Y : |X \cap Y| > n\}$

All the above functions related to the interpretation of the GNP formed with *the same* behave like simple functions interpreting various comparative and superlative constructions because they satisfy the higher order argument invariance as defined in definition 8.

The second class of higher order functions taking sets and relations as arguments are functions needed to interpret the so-called *cumulative readings* of some sentences. Consider (28):

(28) Leo and Lea/three philosophers wrote nine articles (for the journal).

The cumulative reading of this sentence is the reading under which it does not mean that Leo and Lea (or every of the three philosophers) wrote 9 articles each but that some articles were written by Leo or Lea (or some philosopher), maybe jointly, and that every article was written by (at least) one contributor indicated by the subject of the sentence. Sentences which give rise to cumulative reading have the general form given in (29):

(29) NP TVP CumDet(CN).

The CumDet is a *cumulative determiner* which is a GDet. Usually, on the "surface", it is not formally distinguished from an ordinary determiner (at least in English); in (28) it is the determiner *nine* "interpreted" cumulatively. There are of course constraints on which determiners can be interpreted cumulatively. They are ignored here.

The CumDet in (29) denotes a type $\langle 1,2:\langle 1\rangle\rangle$ function F_{cum} defined in (30):

(30) If Det denotes D, then $F_{cum}(X,R) = \{Z : \exists_A Li(Z,A) \wedge Z(SOME(X)_{acc}R) = 1 \wedge D(X)(SOME(A)_{acc}R^{-1}) = 1\}$, where Z runs over the set of plural type $\langle 1\rangle$ quantifiers and $SOME$ is a type $\langle 1,1\rangle$ quantifier defined as $SOME(X)(Y) = 1$ iff $X \cap Y \neq \emptyset$

The following proposition holds:

Proposition 9. *If D is conservative then the function F_{cum} is conservative.*

Thus all higher order functions taking sets and binary relations as arguments considered above are conservative.

To conclude this section I indicate two conditions, which functions which are accusative lifts, should satisfy:

(31) If a type $\langle 2:\langle 1\rangle\rangle$ function F is an accusative lift, then for any $a,b \in E$, any $A,B \subseteq E$ and any binary relation R and S the following conditions hold: (i) if $aR = bS$ then $I_a \in F(R)$ iff $I_b \in F(S)$ and (ii) $Ft(A) \in F(R)$ and $Ft(B) \in F(R)$ iff $Ft(A \cup B) \in F(R)$.

Using the above conditions it is easy to show that $SELF^L$, LEA and $SAME(C)$, for instance, are not accusative lifts.

4 Higher Order Functions and Fregean Quantifiers

One can observe that the functions we have discussed above are related to so-called *non-Fregean* (or *Frege irreducible*) quantifiers. A type $\langle 2\rangle$ quantifier F is Fregean, or Frege reducible, (cf. Keenan 1992) iff there exist two type $\langle 1\rangle$ quantifiers Q and Q_1 such that $F(R) = Q_1(Q_{acc}(R))$. A type $\langle 2\rangle$ quantifier is non-Fregean iff it is not Frege reducible.

Various tests showing that a type $\langle 2\rangle$ quantifier is Fregean have been established and various type $\langle 2\rangle$ quantifiers have been shown to be non-Fregean (Keenan 1992, van Eijck 2005) with their help. Keenan 1992 proved the following theorem which can be used to show that some functions are not Fregean (see also van Eijck 2005):

Proposition 10. *(Keenan) If F_1 and F_2 are Fregean (type $\langle 2\rangle$) quantifiers then $F_1 = F_2$ iff for all $A,B \subseteq E$ it holds that $F_1(A \times B) = F_2(A \times B)$*

Let me illustrate by a somewhat abstract example how proposition 9 can be used to show that some type $\langle 2\rangle$ quantifiers are not Fregean. For this the following proposition is useful (Zuber 2007):

Proposition 11. *Let Q be a type $\langle 1\rangle$ quantifier which is self-dual and positive. Then, for any $A,B \subseteq E$ and any type $\langle 1\rangle$ quantifier Q_1, the following holds: $Q(Q_1)_{acc}(A \times B) = Q_1 Q_{acc}(B \times A)$.*

We will construct a non-Fregean quantifier from two quantifiers Q and Q_1. The quantifier Q is defined as follows: $Q = AT\text{-}LEAST(n, C)$ and $Y \in Q$ iff $|C \cap Y| \geq n \wedge n = |C|/2$ and n is even. The quantifier Q is self-dual and positive. The quantifier $Q_1 = \neg(I_a \vee I_b)$ for some $a, b \in E$. Given proposition 6 the equivalence in (32) holds for any $A, B \subseteq E$:

(32) $Q(Q_1)_{acc}(A \times B) = Q_1(Q_{acc}(B \times A))$

We can now construct a non-Fregean type $\langle 2 \rangle$ quantifier F from quantifiers Q and Q_1: the type $\langle 2 \rangle$ quantifier F defined in (33) is non-Fregean:

(33) $F(R) = Q_1(Q_{acc}(R^{-1}))$ (or $F(R) = Q_1(Q(R))$

Indeed, it follows from proposition 11 that F takes the same value on product relations as the Fregean quantifier $Q(Q_1)_{acc}$. Given definition equivalence (32) and proposition 11 we have: $F(A \times B) = Q_1(Q_{acc}(B \times A) = Q(Q_1)_{acc}(A \times B)$. It is easy to show, however, that F and $Q(Q_1)_{acc}$ do not take the same value on the relation R defined as follows: $R = A \times \{a\} \cup B \times \{b\}$ (if $|A| + |B| \geq |C|/2$, $A \leq |C|/2$ and $|B| \leq |C|/2$). Thus, following proposition 11, F is not Fregean.

The following proposition is a direct consequence of proposition 11:

Proposition 12. *If F_1 and F_2 are type $\langle 2 : \langle 1 \rangle \rangle$ functions which are accusative lifts then $F_1(R) = F_2(R)$ iff for all $A, B \subseteq E$ it holds that $F_1(A \times B) = F_2(A \times B)$*

It follows from proposition 12 that to show that a type $\langle 1, 2 : \langle 1 \rangle \rangle$ function F is not an accusative lift it is enough to find an accusative lift G and a plural quantifier Q such that $Q \in F(A \times B)$ iff $Q \in G(A \times B)$ for any product-relation $A \times B$ but for some R non product $Q \in F(R) \not\equiv Q \in G(R)$.

To illustrate proposition 12, consider the the function F_1 defined with the help of the function $ONLY\text{-}SAME_n$ given in (28): $F_1(R) = ONLY\text{-}SAME_n(C, R)$, for a fixed $C \subseteq E$. Using proposition 10 one shows that function F_1 is not an accusative lift by comparing it with the accusative lift of the type $\langle 1 \rangle$ quantifier $EXACTLY(n)(C)$ defined as $EXACTLY(n)(C)(Y) = 1$ iff $|C \cap Y| = n$. Indeed the following equality holds for any $A, B \subseteq E$ and any plural quantifier $Q = I_a \wedge I_b$, eher $a, b \in E$):

(34) $Q \in ONLY\text{-}SAME_n(C, A \times B)$ iff $Q \in (EXACTLY(n)(C)_{acc})^L(A \times B)$

Since obviously $Q \in ONLY\text{-}SAME_n(C)$ and $EXACTLY(n)(C)$ are different on at least one non product relation, it follows from (34) and proposition 8 that F_1 is not an accusative lift.

One can show in the similar way, using some results from Keenan 1992, that the function LEA is not an accusative lift. Similarly type $\langle 2 : \langle 1 \rangle \rangle$ functions obtained from $SAME$ and F_{cum} by fixing their set argument, that is the functions $G_C(R) = SAME(C, R)$ and $H_C(R) = F_{cum}(A, R)$, are not accusative lifts.

Type $\langle 2 : \langle 1 \rangle \rangle$ functions and type $\langle 1 \rangle$ quantifiers give rise to type $\langle 2 \rangle$ quantifiers in the following way. Let F be a type $\langle 2 : \langle 1 \rangle \rangle$ function and Q a type $\langle 1 \rangle$ quantifier. We will say that F and Q give rise to the type $\langle 2 \rangle$ quantifier $G_{F,Q}$ iff the following holds: $G_{F,Q}(R) = 1$ iff $Q \in F(R)$.

Accusative lifts give rise to Fregean quantifiers. More precisely we have:

Proposition 13. *Let F be a type $\langle 2:\langle 1\rangle\rangle$ function and Q a type $\langle 1\rangle$ quantifier. Define a type $\langle 2\rangle$ quantifier $G_{F,Q}$ as follows: $G_{F,Q}(R) = 1$ iff $Q \in F(R)$. Then F is an accusative lift iff $G_{F,Q}$ is Fregean for any Q.*

Interestingly Fregean quantifiers can also be obtained in a systematic way from lifts of some type $\langle 2:1\rangle$ functions which are not accusative lifts. Consider the following definition (Keenan and Westerståhl 1997), which can be considered as a particular case of definition 7:

Definition 10. *A type $\langle 2:1\rangle$ function F is* predicate invariant *iff for any $a \in E$ and any binary relations R, S, if $aR = aS$ then $a \in F(R)$ iff $a \in F(S)$.*

Obviously the accusative extension of a type $\langle 1\rangle$ quantifier is predicate invariant. Moreover, the functions $SELF$ and $SELF \wedge Q_{acc}$ (where Q is a type $\langle 1\rangle$ quantifier), for instance, are also predicate invariant. For predicate invariant functions we have (Zuber 2012b):

Proposition 14. *Let F be a type $\langle 2:1\rangle$ predicate invariant function. Define a type $\langle 2\rangle$ quantifier G_{F,I_a} as follows: $G_{F,I_a}(R) = 1$ iff $I_a \in F^L(R)$. Then G_{F,I_a} is Fregean for any $a \in E$.*

Proof. Define the function h_F which maps every $a \in E$ to a type $\langle 1\rangle$ quantifier in the following way: $h_F(a)(Y) = 1$ iff $a \in F(\{a\} \times Y)$. Since F is predicate invariant we have $y \in F(R)$ iff $y \in F(\{y\} \times yR)$ (because $yR = y(\{y\} \times yR)$). From this it follows that $G_{F,I_a}(R) = I_a((h_F(a))_{acc}(R)))$ for any $a \in E$. Thus G_{F,I_a} is equivalent to $Q_1(Q_{acc})$ where $Q_1 = I_a$ and $Q = h_F(a)$. □

The function $SELF$ gives rise also to non-Fregean quantifiers. To obtain a sufficient condition showing that $SELF$ gives rise to non-Fregean quantifiers we will use the following criterion (cf. van Eijck 2005):

Proposition 15. *(van Eijck) : Let F be a type $\langle 2\rangle$ quantifier such that $F(\emptyset) = 0$. The reduct F^* of F is defined as follows: $F^* = Q_1(Q_2)_{acc}$ where Q_1 and Q_2 are positive type $\langle 1\rangle$ quantifiers such that $Q_1(X \neq \emptyset) = 1$ iff $\exists_{B\subseteq E}F(X \times B) = 1$ and $Q_2(Y \neq \emptyset) = 1$ iff $\exists_{A\subseteq E}F(A \times Y) = 1$. Then F is Fregean iff $F^* = F$.*

Consider now a type $\langle 2\rangle$ quantifier F defined as $F(R) = Q(SELF(R))$, where Q is a positive type $\langle 1\rangle$ quantifier. Observe first that F is convertible, that is $F(R) = F(R^{-1})$. Furthermore, the reduct F^* of F has the following form: $F^* = Q(Q_{acc})$. Westerståhl (1996) proved that for Q positive the equality $Q(Q_{acc}(R)) = Q(Q_{acc}(R^{-1}))$ holds iff Q is either a union or intersection of individuals or a finite symmetric difference of individuals. Thus if Q is neither of them, the type $\langle 2\rangle$ quantifier $F = Q(SELF)$ is not Fregean. This is the case for instance with atomic quantifiers, which are neither unions nor intersections nor finite differences of individuals. An illustration is given in (35):

(35) Only Leo hates himself.

The NP *only Leo* denotes the atomic quantifier whose only member contains just the singleton to which Leo refers. This means that the type $\langle 2\rangle$ quantifier involved in the intertpretation of (35), that is the type $\langle 2\rangle$ quantifier denoted by *only Leo...himself*, is not Fregean.

5 Conclusive Remarks

To interpret complex VPs formed from transitive verb phrases, one needs functions taking binary relations as arguments. In this paper, higher order functions, that is functions which have sets of type $\langle 1 \rangle$ quantifiers as output (and binary relations and, possibly sets, as arguments), have been studied. All these functions are "conceptually simple" in the sense that they can be defined by the usual means of elementary set theory.

The output of higher order functions is denoted by VPs whose category is raised to $S/(S/(S/NP))$ in order to avoid type mismatch. Many of these functions are not lifts of "simple" functions. Various constraints obeyed by both kinds of functions have been presented. These constraints are natural generalisations, often not trivial, of similar constraints known from generalised quantifier theory and the analysis of simple functions on binary relations. In particular, it is shown that such functions are conservative, in the generalised sense. Some of these functions have a stronger property such as intersectivity or evencardinality.

It has been shown that it is preferable to treat some expressions as denoting higher order functions, and not quantifiers, given the fact that they can very easily form Boolean compounds. In particular reciprocals are considered as denoting higher order functions on binary relations, and not polyadic quantifiers, because they form Boolean compounds with reflexives or NPs in object positions. Syntactic arguments supporting this choice are easy to find.

However, it has also been shown that higher order functions give rise (in association with a type $\langle 1 \rangle$ quantifier) to Frege irreducible type $\langle 2 \rangle$ quantifiers. Some relationships between (Frege) reducibility of quantifiers obtained in this way and properties of some higher order functions involved have been established.

One can hope that the results presented in this paper will help us to answer various questions concerning the expressive power of natural languages. It is well-known (Keenan 2007, Zuber 2010b) that the existence of anaphors and anaphoric determiners in NLs shows that the expressive power of natural languages would be less than it is if the only noun phrases we needed were those interpretable as subjects of main clause intransitive verbs. The reason is that anaphors like *himself, herself* must be interpreted by functions from relations to sets which lie outside the class of generalised quantifiers as classically defined since, in particular they give rise to non-Fregean quantifiers and to verbal arguments which are not lifts of "classical" NPs. Similarly, some comparative constructions used in object positions cannot be interpreted by extended generalised quantifiers and their existence also extends the expressive power of natural languages (Zuber 2011a). In this paper some preliminary results are presented to show that the existence of higher order anaphors and comparatives even further extends the expressive power of NLs. However, functions denoted by such higher order anaphors and comparatives obey constraints similar to those interpreting simple anaphors and comparatives since the former are natural generalisations of the latter.

References

Beck, S.: Exceptions in Relational Plurals. In: Jackson, B., Matthews, T. (eds.) SALT X, pp. 1–16. Cornell University (2000)

Dalrymple, M., et al.: Reciprocal expressions and the concept of reciprocity. Linguistics and Philosophy 21, 159–210 (1998)

Keenan, E.L.: Semantic Case Theory. In: Groenendijk, J., Stokhof, M. (eds.) Proc. of the Sixth Amsterdam Colloquium (1987)

Keenan, E.L.: Beyond the Frege boundary. Linguistics and Philosophy 15, 199–221 (1992)

Keenan, E.L.: On the denotations of anaphors. Research on Language and Computation 5(1), 5–17 (2007)

Keenan, E.L., Westerståhl, D.: Generalized Quantifiers in Linguistics and Logic. In: van Benthem, J., ter Meulen, A. (eds.) Handbook of Logic and Language, pp. 837–893. Elsevier, Amsterdam (1997)

van Eijck, J.: Normal Forms for Characteristic Functions. Journal of Logic and Computation 15(2), 85–98 (2005)

Westerståhl, D.: Self-commuting quantifiers. The Journal of Symbolic Logic 61(1), 212–224 (1996)

Zuber, R.: Indépendance faible des quantificateurs. Logique et Analyse 198, 173–178 (2007)

Zuber, R.: Generalising Conservativity. In: Dawar, A., de Queiroz, R. (eds.) WoLLIC 2010. LNCS (LNAI), vol. 6188, pp. 247–258. Springer, Heidelberg (2010a)

Zuber, R.: Semantic Constraints on Anaphoric Determiners. Research on Language and Computation 8, 255–271 (2010b)

Zuber, R.: Some generalised comparative determiners. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS, vol. 6736, pp. 267–281. Springer, Heidelberg (2011a)

Zuber, R.: Generalised Quantifiers and the Semantics of the same. In: Ashton, N., et al. (eds.) Proceedings of SALT 21, e-Language, pp. 515–531 (2011b)

Zuber, R.: Reciprocals as higher order functions. In: Proceedings of the Ninth International Workshop of Logic and Engineering of Natural Language Semantics 9 (LENLS 9), pp. 118–129 (2012a)

Zuber, R.: Reflexives and non-Fregean quantifiers. In: Graf, T., et al. (eds.) Theories of Everything: in Honor of Ed Keenan, UCLA Working Papers in Linguistics, vol. 17, pp. 439–445 (2012b)

Author Index