Proceedings

of

Mathematics of Language 8

held in conjunction with

The Second
North American Summer School in
Logic, Language, and Information

Bloomington, Indiana, June 19-22, 2003

R. T. Oehrle & J. Rogers (editors)

Contents

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

Chapter 1

Introduction

This collection represents the preprinted proceedings of the eighth Mathematics of Language Conference (**MoL8**), held in conjunction with the 2nd North American Summer School in Logic, Language, and Information (NASSLLI), in Bloomington, Indiana, June 19-22, 2003. The chapters of this volume represent the papers selected for the conference on the basis of abstract submissions.[1] In addition to the submitted papers, the MoL8 conference program contains a variety of special events: a symposium on Language and Game Theory; a invited lecture by Ed Keenan and Ed Stabler (coextensive with the final lecture of their NASSLLI course*A Mathematical Theory of Grammatical Categories*; an invited lecture by Aravind Joshi; and a symposium on Statistical and Symbolic Aspects of Natural Language Learnability. The full conference program appears on the following page.

We would like to record our thanks to all those who submitted abstracts to the conference, to the program committee (listed following the program) for their hard work and high standards, to the participants in the special sessions. We are especially indebted to Larry Moss, past president of **MoL** who, in his guise as NASSLLI organizer, also served as *de facto* Local Arrangements Chair for **MoL8**. Finally, our preparation of this proceedings was greatly facilitated by a LaTeX 2_ε class file adapted from work by Geert-Jan Kruijff and Gerhard Jäger.

R. T. Oehrle
Berkeley, CA

J. Rogers
Earlham College
Richmond, Indiana

[1] The paper by Luis Casillas Martínez appears by title only, in the absence of a submitted version of the full paper.

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

Mathematics of Language 8 Program

Friday, June 20, 2003

9.00–12.30	*Symposium: Language and Game Theory* confirmed speakers: Martin Nowak (Harvard University) Rohit Parikh (CUNY) Robert van Rooy (University of Amsterdam) Discussion: Larry Moss (Indiana University)
12.30–14.00	**break**
14.00–14.30	*m-Linear Context-Free Rewriting Systems in Abstract Categorial Grammar* Philippe de Groote and Sylvain Pogodalla (LORIA)
14.30–15.00	*Discovering a new class of languages* Sean Fulop (University of Chicago)
15.00–15.30	*Global Index Grammar and Descriptive Power* Jose M. Castano (Brandeis University)
15.30–16.00	**break**
16.00–16.30	*On Scope Dominance with Monotone Quantifiers* Gilad Ben-Avi & Yoad Winter (Technion)
16.30–17.00	*Boolean Operators for Vectors: Negation and Disjunction of Word-Meanings* Dominic Widdows & Stanley Peters (CSLI, Stanford)
17.00–17.30	*A set-theoretical investigation of Panini's Sivasutras* Wiebke Petersen (Heinrich-Heine-Universität, Düsseldorf)

Saturday, June 21, 2003

9.00–9.30	*Some Remarks on Arbitrary Multiple Pattern Interpretation* Carlos Martin-Vide (Rovira i Virgili University, Tarragona) & Victor Mitrana (Bucharest)
9.30–10.00	*The Semantic Complexity of some Fragments of English* Ian Pratt-Hartmann (University of Manchester)
10.00–10.30	*The complexity of reasoning on finite trees* Patrick Blackburn (LORIA), Bernard Gaiffe (LORIA), Maarten Marx (Amsterdam)
10.30–11.00	**break**
11.00–12.00	*MoL/NASSLLI Lecture A Mathemtical Theory of Grammatical Categories* Ed Keenan & Ed Stabler (UCLA)
12.00–14.00	**break**
14.00–14.30	*Bounded and Ordered Satisfiability: Connecting Recognition with Lambek-style Calculi to Classical Satisfiability Testing*

	Periklis A. Papakonstantinou, Michail Flouris, Lap Chi Lau, Tsuyoshi Morioka, and Gerald Penn (Toronto)
14.30–15.00	*Querying Linguistic Treebanks with Monadic Second-Order Logic in Linear Time* Stephan Kepser (Tübingen)
15.00–15.30	*Spatial models for language: linguistic structure from an external perspective* Luis D. Casillas Martinez (Stanford)
15.30-16.00	*Learning local transductions is hard* Martin Jansche (Ohio State University)
16.00–16.30	**break**
16.30–17.30	*Invited Lecture* Aravind Joshi (University of Pennsylvania)
17.30–18.00	*MoL business meeting*
18.00	*MoL8 banquet*

Sunday, June 22, 2003

9.00–12.30	*Symposium: Statistical and Symbolic Aspects of Natural Language Learnability* confirmed speakers: Jason Eisner (Johns Hopkins) Makoto Kanazawa (Tokyo) Dan Osherson (Rice) Ed Stabler (UCLA)

Mathematics of Language 8 Program Committee

Anne Abeillé (Paris 7)
Tilman Becker (DFKI)
Patrick Blackburn (LORIA)
Pierre Boullier (INRIA)
Gosse Bouma (Groningen)
Wojciech Buszkowski (Poznań)
Christophe Fouquere (Paris 13)
Nissim Francez (Haifa)
Thilo Goetz (IBM)
Gerhard Jaeger (Potsdam)
David Johnson (IBM)
Mark Johnson (Brown)
Aravind Joshi (UPenn)
Ruth Kempson (London)
Andras Kornai (Metacarta)
Marcus Kracht (UCLA)
Uli Krieger (DFKI)
Geert-Jan Kruijff (Saarbrücken)
Natasha Kurtonina (Fitchburg)
Alain Lecomte (Grenoble)
Carlos Martin-Vide (Tarragona)
Jens Michaelis (Potsdam)
Mehryar Mohri (AT&T)
Uwe Mönnich (Tübingen)
Michael Moortgat (Utrecht)
Drew Moshier (Chapman)
Larry Moss (Indiana)
Mark-Jan Nederhof (Groningen)
Gertjan van Noord (Groningen)
Richard Oehrle (Berkeley, CA)
Gerald Penn (Toronto)
Stanley Peters (Stanford)
Geoffrey Pullum (Santa Cruz)
Owen Rambow (Columbia)
James Rogers (Earlham)
Christian Retore (INRIA & LaBRI, Bordeaux)
Robert van Rooy (Amsterdam)
Giorgio Satta (Padua)
Vijay K. Shanker (Delaware)
Ed Stabler (UCLA)
Mark Steedman (Edinburgh)
Hans Joerg Tiede (Illinois Wesleyan)
Christian Wartena (Potsdam)

Chapter 2

On Scope Dominance With Monotone Quantifiers

GILAD BEN-AVI* AND YOAD WINTER*

ABSTRACT. We characterize pairs of monotone generalized quantifiers Q_1 and Q_2 that give rise to an entailment relation between their two relative scope construals. This result is used for identifying entailment relations between the two scopal interpretations of simple sentences of the form NP_1-V-NP_2. The general characterization that we give turns out to cover more examples of such entailments besides the familiar type where the NPs are headed by *some* and *every*.

2.1 Introduction

Scope ambiguity in simple transitive sentences of the form NP_1-V-NP_2 is one of the well-studied areas in natural language semantics. It has been often observed that whether this kind of ambiguity is manifested in natural language may depend on entailment relations between the readings of such sentences. For instance, Zimmerman (1993) characterizes the class of *scopeless* ("name like") noun phrases – the class of NP_2s for which the two readings of the sentence NP_1-V-NP_2 are equivalent for any noun phrase NP_1 and transitive verb V. A more general notion, first addressed by Westerståhl (1986), involves uni-directional entailment between the two readings, which is referred to here as *scope dominance*. A sentence NP_1-V-NP_2 exhibits scope dominance if one of its readings entails the other. A familiar case is when the subject (or object) denotes an existential quantifier (e.g., *some student*) and the object (or subject, respectively) denotes a universal quantifier (e.g., *every teacher*). Westerståhl shows that in the class of non-trivial upward monotone (simple) quantifiers over finite domains, scope dominance appears if and only if the subject or object are existential or universal.

Altman et al. (2002) generalize Westerståhl's result, and show a full characterization of scope dominance with *arbitrary* upward monotone quantifiers over *countable* domains. In this paper we generalize Westerståhl's result in another way, and characterize scope dominance between simple upward *or downward* monotone quantifiers over finite domains.

*Computer Science, Technion, Haifa 32000, Israel; {bagilad,winter}@cs.technion.ac.il.

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

This result is based on the numerical presentation of quantifiers over finite domains as recently proposed by Väänänen and Westerståhl (2001). It leads to a general characterization of entailments over finite domains between readings of sentences with (potential) scope ambiguity as in the following cases, where both subject and object are monotone.

(1) Less than five referees read at least one of the abstracts.

(2) Less than five referees read each of the abstracts.

In sentence (1), the object narrow scope reading entails the object wide scope reading. In (2) the entailment between the two readings is in the opposite direction. Note that the definite noun phrase *the abstracts* leads in both sentences to the presupposition that abstracts exist, which is crucial for the respective entailments to hold. Similarly to Westerståhl's result about upward monotone quantifiers, in both examples scope dominance is created by the presence of an existential or universal quantifier. However, as we shall see, our extension of Westerståhl's characterization also reveals cases of scope dominance with monotone quantifiers other than *every* or *some*.

2.2 Background

This section briefly reviews some notions from generalized quantifier theory, which will be used in our characterization of scope dominance. A (*generalized*) *quantifier* over a domain E is a set $Q \subseteq \wp(E)$. A quantifier Q over E is *upward (downward) monotone* iff whenever $A \in Q$ and $A \subseteq A'$ ($A' \subseteq A$), then $A' \in Q$. In the sequel, we sometimes use the abbreviations "MON↑" and "MON↓" for "upward/downward monotone". A quantifier Q is called *trivial* iff either $Q = \emptyset$ or $Q = \wp(E)$.

Given a binary relation $R \subseteq E^2$ and $x \in E$ we write $R_x \stackrel{def}{=} \{y \in E : R(x,y)\}$ and $R^y \stackrel{def}{=} \{x \in E : R(x,y)\}$. The *Object Narrow Scope* (ONS) reading of a simple transitive sentence is naturally interpreted in a domain E as the proposition Q_1Q_2R as defined below, where Q_1 and Q_2 are the subject and object quantifiers (respectively) over E, and the relation $R \subseteq E^2$ is the denotation of the verb.

(3) $Q_1Q_2R \stackrel{def}{\Leftrightarrow} \{x \in E : R_x \in Q_2\} \in Q_1$.

Similarly, the *Object Wide Scope* (OWS) reading is interpreted as $Q_2Q_1R^{-1}$, which by (3) is equivalent to the requirement $\{y \in E : R^y \in Q_1\} \in Q_2$.

Given two quantifiers Q_1 and Q_2 we say that Q_1 is *scopally dominant* over Q_2 iff for every $R \subseteq E^2$: $Q_1Q_2R \Rightarrow Q_2Q_1R^{-1}$.

The *dual* of a quantifier Q over E is the quantifier $Q^d = \{X \subseteq E : E \setminus X \notin Q\}$. The following fact summarizes some simple properties of quantifier duality.

Fact 1. *For any quantifiers Q, Q_1, Q_2 over E:*

1. $(Q^d)^d = Q$
2. *Q_1 is scopally dominant over Q_2 iff Q_2^d is scopally dominant over Q_1^d*
3. $Q = \emptyset \Leftrightarrow Q^d = \wp(E)$

$\textbf{every}'(A)$	=	$\{X \subseteq E : \lvert A \cap X \rvert \geq \lvert A \rvert\}$
$\textbf{not\_every}'(A)$	=	$\{X \subseteq E : \lvert A \cap X \rvert < \lvert A \rvert\}$
$\textbf{some}'(A)$	=	$\{X \subseteq E : \lvert A \cap X \rvert \geq 1\}$
$\textbf{no}'(A)$	=	$\{X \subseteq E : \lvert A \cap X \rvert < 1\}$
$\textbf{more\_than\_half}'(A)$	=	$\{X \subseteq E : \lvert A \cap X \rvert \geq \lfloor \frac{\lvert A \rvert}{2} \rfloor + 1\}$
$\textbf{at\_least\_half}'(A)$	=	$\{X \subseteq E : \lvert A \cap X \rvert \geq \lceil \frac{\lvert A \rvert}{2} \rceil\}$
$\textbf{less\_than\_half}'(A)$	=	$\{X \subseteq E : \lvert A \cap X \rvert < \lceil \frac{\lvert A \rvert}{2} \rceil\}$

Table 2.1: CPI-based Quantifiers

4. Q is MON↑ *(*MON↓*) iff Q^d is* MON↑ *(*MON↓*).*

A *determiner* over a domain E is a function D that assigns to every $A \subseteq E$ a quantifier $D(A)$. In this paper we concentrate on *simple* quantifiers Q: quantifiers that satisfy $Q = D(A)$, for some $A \subseteq E$ and a *conservative* and *permutation invariant* determiner D. Standardly, by saying that a determiner D over E is *conservative* we mean that for all $A, B \subseteq E$: $B \in D(A) \Leftrightarrow B \cap A \in D(A)$. Also standardly, a determiner D over E is called *permutation invariant* iff for every permutation π on E, and for all $A, B \subseteq E$: $B \in D(A) \Leftrightarrow \pi B \in D(\pi A)$, where for a set $X \subseteq E$, $\pi X = \{\pi(x) : x \in X\}$. In the sequel, whenever a quantifier Q can be interpreted as $D(A)$ for such A and D, we say that Q is *CPI-based*.

As pointed out by Väänänen and Westerståhl (2001), every monotone CPI-based quantifier Q over a finite domain E can be represented as follows, for some $A \subseteq E$ and $n \geq 0$.

(4) a. $Q = \{X : \lvert A \cap X \rvert \geq n\}$, if Q is MON↑

b. $Q = \{X : \lvert A \cap X \rvert < n\}$, if Q is MON↓

The duals of such CPI-based quantifiers can be represented as follows, respectively.[1]

(5) a. $Q^d = \{X : \lvert A \cap X \rvert \geq \lvert A \rvert - n + 1\}$

b. $Q^d = \{X : \lvert A \cap X \rvert < \lvert A \rvert - n + 1\}$

In table 2.1 we give some examples of monotone CPI-based quantifiers $D(A)$ over a finite domain E for various determiners D and arbitrary sets $A \subseteq E$, together with their presentation according to the scheme in (4). In these examples, for any real number r, the notations $\lfloor r \rfloor$ and $\lceil r \rceil$ standardly stand for the integer value closest to r from below and from above, respectively.

[1] Provably, a dual of a CPI-based quantifier is also CPI-based.

2.3 Scope dominance with monotone CPI-based quantifiers over finite domains

This section characterizes the pairs of CPI-based quantifiers Q_1 and Q_2 over finite domains, where Q_1 is scopally dominant over Q_2. Proposition 3 below first addresses the case where Q_1 is MON↑ and Q_2 is MON↓. Its proof uses the following simple combinatorial lemma, whose proof is given here for sake of completeness.

Lemma 2. *Let $\ell, m, k, n \in \mathbb{N}$ s.t. $\ell, k > 0$, $m \geq 0$ and $0 < n \leq k$. Let X be a set with $|X| = k$. Then 1 and 2 below are equivalent:*

1. *There are ℓ subsets of X: $X_1, \ldots, X_\ell$, s.t. $|X_i| = n$, $1 \leq i \leq \ell$, and every $x \in X$ is in at most m of the X_is.*
2. *$\ell n \leq mk$.*

Proof. Let $X = \{x_1, \ldots x_k\}$. For every $X_1, \ldots, X_\ell \subseteq X$ let $m_i = |\{X_j : 1 \leq j \leq \ell \wedge x_i \in X_j\}|$.
(1) ⇒ (2):
Let $X_1, \ldots, X_\ell \subseteq X$ such that for every j s.t. $1 \leq j \leq \ell$: $|X_j| = n$, and for every i s.t. $1 \leq i \leq k$: $m_i \leq m$. Thus,

$$\ell n = \sum_{i=1}^{k} m_i \leq mk$$

(2) ⇒ (1):
Assume that $\ell n \leq mk$. Construct $X_1, \ldots, X_\ell \subseteq X$ as follows:

$$\begin{aligned}
X_1 &= \{x_1, \ldots, x_n\} \\
&\vdots \\
X_j &= \{x_{((j-1)n+1) \bmod k}, \cdots, x_{(jn) \bmod k}\} \\
&\vdots \\
X_\ell &= \{x_{((\ell-1)n+1) \bmod k}, \cdots, x_{(\ell n) \bmod k}\}
\end{aligned}$$

It is not hard to verify that for all i, j s.t. $1 \leq i, j \leq k$: $m_j - 1 \leq m_i \leq m_j + 1$. Assume for contradiction that for some i s.t. $1 \leq i \leq k$: $m_i = m' > m$. Thus,

$$\ell n = \sum_{i=1}^{k} m_i \geq m' + (m'-1)(k-1) = (m'-1)k + 1 > mk$$

in contradiction to the assumption that $\ell n \leq mk$. Hence, for all i s.t. $1 \leq i \leq k$: $m_i \leq m$. □

Proposition 3. *Let Q_1 and Q_2 be two CPI-based quantifiers over a finite domain E s.t. Q_1 is* MON↑ *and Q_2 is* MON↓*. According to the presentation in (4), assume that for some $A, B \subseteq E$ and $n, m \geq 0$: $Q_1 = \{X : |A \cap X| \geq n\}$ and $Q_2 = \{Y : |B \cap Y| < m\}$. Then Q_1 is scopally dominant over Q_2 iff one of the following holds:*

(i) $|A| < n + \frac{n}{m}$ *and both* $0 < n \leq |A|$ *and* $0 < m \leq |B|$ *(both quantifiers are not trivial.)*

(ii) $n > |A|$ *(*$Q_1 = \emptyset$*).*

(iii) $m > |B|$ *(*$Q_2 = \wp(E)$*).*

(iv) $n > 0$ *and* $m = 0$ *(*$Q_2 = \emptyset$ *and* $Q_1 \neq \wp(E)$*).*

Proof. It is easy to verify that if at least one of Q_1 and Q_2 is trivial, then Q_1 is scopally dominant over Q_2 iff one of the clauses (ii)-(iv) holds. Thus, we assume that both quantifiers are not trivial, i.e., $0 < n \leq |A|$ and $0 < m \leq |B|$. Now Q_1 is *not* scopally dominant over Q_2 iff the following condition holds:

C1. There exists $R \subseteq E^2$ such that $|\{x \in A : |R_x \cap B| < m\}| \geq n$ and $|\{y \in B : |R^y \cap A| \geq n\}| \geq m$.

We claim that C1 is equivalent to the following condition.

C2. There exist $T \subseteq E^2$ and $B' \subseteq B$ with $|B'| = m$ ($B' = \{b_1, \ldots, b_m\}$) such that $|A \setminus \bigcap_{i=1}^{m} T^{b_i}| \geq n$ and $\forall b \in B'\ |T^b \cap A| = n$.

To see that, assume first that C1 holds, and consider $B' = \{b_1, \ldots, b_m\} \subseteq \{y \in B : |R^y \cap A| \geq n\}$. For each b_i, let $A_i \subseteq R^{b_i} \cap A$, $|A_i| = n$. Define $T = \bigcup_{i=1}^{m}(A_i \times \{b_i\})$, and observe that from the assumptions about the A_is it follows that $\{x \in A : |R_x \cap B| < m\} \subseteq A \setminus \bigcap_{i=1}^{m} A_i$.
As for the other direction, if C2 holds, define $R = T \cap (A \times B')$.

Now, C2 is equivalent to the requirement that there exist $m+1$ subsets of A: $A_1, \ldots, A_m, A_{m+1}$ such that $|A_i| = n$, $1 \leq i \leq m+1$, and $\bigcap_{i=1}^{m+1} A_i = \emptyset$. To see that, let A_i corresponds to $T^{b_i} \cap A$ for any i s.t. $1 \leq i \leq m$, and let A_{m+1} corresponds to $A \setminus \bigcap_{i=1}^{m} A_i$. By Lemma 2, this requirement holds iff $|A| \geq n + \frac{n}{m}$. □

The dual of the kind of scope dominance that is characterized in Proposition 3 is the case in which Q_1 is MON↓ and Q_2 is MON↑. Using Fact 1 and the observation in (5), we get the following corollary of Proposition 3.

Corollary 4. *Let* Q_1 *and* Q_2 *be two CPI-based quantifiers over a finite domain* E *s.t.* Q_1 *is* MON↓ *and* Q_2 *is* MON↑*. According to the presentation in (4), assume that for some* $A, B \subseteq E$ *and* $n, m \geq 0$*:* $Q_1 = \{X : |A \cap X| < n\}$ *and* $Q_2 = \{Y : |B \cap Y| \geq m\}$*. Then* Q_1 *is scopally dominant over* Q_2 *iff one of the following holds:*

(i) $|B| > (m-1)(|A| - n + 2)$ *and both* $0 < n \leq |A|$ *and* $0 < m \leq |B|$ *(both quantifiers are not trivial.)*

(ii) $n = 0$ *(*$Q_1 = \emptyset$*).*

(iii) $m = 0$ *(*$Q_2 = \wp(E)$*).*

(iv) $n > |A|$ *and* $m \leq |B|$ *(*$Q_1 = \wp(E)$ *and* $Q_2 \neq \emptyset$*).*

Proposition 5 below covers the case in which both quantifiers are MON↓. The proof is similar to the proof of Proposition 3, and is omitted here.

Proposition 5. *Let Q_1 and Q_2 be two* MON↓ *CPI-based quantifiers over a finite domain E. According to the presentation in (4), assume that for some $A,B \subseteq E$ and $n,m \geq 0$: $Q_1 = \{X : |A \cap X| < n\}$ and $Q_2 = \{Y : |B \cap Y| < m\}$. Then Q_1 is scopally dominant over Q_2 iff one of the following holds:*

(i) $2 - \frac{|B|}{m} > \frac{n-1}{|A|-n+1}$ *and both* $0 < n \leq |A|$ *and* $0 < m \leq |B|$ *(both quantifiers are not trivial.)*

(ii) $n = 0$ *($Q_1 = \emptyset$).*

(iii) $m > |B|$ *($Q_2 = \wp(E)$).*

The same method that we use in the proof of Proposition 3, can also be used for the case in which the two quantifiers are MON↑, which is the case dealt with in Westerståhl (1986). This result is also mentioned here without proof.

Proposition 6. *Let Q_1 and Q_2 be two* MON↑ *CPI-based quantifiers over a finite domain E. According to the presentation in (4), assume that for some $A,B \subseteq E$ and $n,m \geq 0$: $Q_1 = \{X : |A \cap X| \geq n\}$ and $Q_2 = \{Y : |B \cap Y| \geq m\}$. Then Q_1 is scopally dominant over Q_2 iff one of the following holds:*

(i) $n = 1$ *or* $n > |A|$ *($Q_1 =$* **some**$'(A)$ *or* $Q_1 = \emptyset$*).*

(ii) $m = |B|$ *or* $m = 0$ *($Q_2 =$* **every**$'(A)$ *or* $Q_2 = \wp(E)$*).*

(iii) $n = 0$ *and* $m \leq |B|$ *($Q_1 = \wp(E)$ and $Q_2 \neq \emptyset$).*

(iv) $n > 0$ *and* $m > |B|$ *($Q_2 = \emptyset$ and $Q_1 \neq \wp(E)$).*

Examples: Let us consider some examples for scope dominance between CPI-based quantifiers over a finite domain E. For the representation of each quantifier, refer back to Table 2.1.

First, note that by Corollary 4, for every non-empty $A \subseteq E$, every MON↓ CPI-based quantifier is scopally dominant over **some**$'(A)$ (=(**every**$'(A))^d$). This accounts for the fact that the ONS reading of sentence (1), paraphrased in (6a) below, entails its OWS reading, paraphrased in (6b). Both readings are paraphrased with a presupposition about the existence of abstracts.[2]

(6) a. $|\{x : \textbf{referee}'(x) \wedge \exists y[\textbf{abstract}'(y) \wedge \textbf{read}'(x,y)]\}| < 5 \ \wedge \exists y[\textbf{abstract}'(y)]$

b. $\exists y[\textbf{abstract}'(y) \wedge |\{x : \textbf{referee}'(x) \wedge \textbf{read}'(x,y)\}| < 5] \ \wedge \exists y[\textbf{abstract}'(y)]$

Analogously to this scope dominance with existential quantification, Proposition 3 entails that for every non-empty $A \subseteq E$, **every**$'(A)$ is scopally dominant over every MON↓ CPI-based quantifier. This accounts for the entailment from the OWS reading of (2), with the *every–less-than-5* order of quantifiers, to its ONS reading, with the *less-than-5–every* order of quantifiers.

[2] Plausibly, plurality in sentence (1) leads to the presupposition that there are at least *two* abstracts. However, we do not use this presupposition here, since the relevant entailment also appears with the weaker presupposition that is assumed above.

Such examples with existential and universal quantifiers do not exhaust the cases of scope dominance with monotone quantifiers. By Proposition 3,
more_than_half$'(A)$ is scopally dominant over **no**$'(B)$ for all $A,B \subseteq E$. By Corollary 4, **not_every**$'(A)$ (=(**no**$'(A)$)^d) is scopally dominant over
at_least_half$'(B)$ (=(**more_than_half**$'(B)$)^d), for all $A,B \subseteq E$. Consider for instance the following sentences.

(7) a. More than half of the referees read no abstract.

b. No abstract was read by more than half of the referees.

Our characterization accounts for the entailment from the ONS interpretation of (7a) to its OWS interpretation, and for the opposite relation in (7b). However, for many speakers both sentences are unambiguous, and have only an ONS reading. Under this unambiguous interpretation, our characterization accounts for the entailment from (the unambiguous) sentence (7a) to (the unambiguous) sentence (7b). Note that the *more than*/*at least half of* quantifiers that are involved in these examples are not first order definable, so these entailments cannot be derived by any axiom system of the first order Predicate Calculus.

As an example in which both quantifiers are MON↓, note that Proposition 5 entails that **less_than_half**$'(A)$ is scopally dominant over **not_every**$'(B)$, for any $A \subseteq E$ and non-empty $B \subseteq E$.

2.4 Concluding remarks

In this paper we characterized scope dominance between upward/downward monotone CPI-based quantifiers over finite domains. This work is part of a wider project that aims to study ambiguity in natural language by way of characterizing entailments between readings of ambiguous sentences. This kind of entailments is a promising area for studying inference in natural language, where high expressibility requires strong restrictions on inferential structures. Moreover, with Van Deemteer (1998) we believe that a characterization of "semantically spurious" ambiguity may lead to improved underspecification methods, and to better techniques for reasoning with underspecified representations. This is of course a major task, and even the characterization of scope dominance that was presented in this paper still leaves some obvious questions open. Most notably, the behavior of non-CPI-based and non-monotone quantifiers, and of quantifiers over infinite domains needs to be further explored. These problems are currently under research.

Bibliography

A. Altman and Y. Peterzil and Y. Winter (2002). Scope dominance with upward monotone quantifiers. Unpublished ms., Technion and Haifa University. Downloadable at http://www.cs.technion.ac.il/~winter/.

K. van Deemteer (1998). Ambiguity and idiosyncratic interpretation. *Journal of Semantics*, **15**:5–36.

J. Väänänen and D. Westerståhl (2001). On the expressive power of monotone natural language quantifiers over finite sets. To appear in *Journal of Philosophical Logic*.

D. Westerståhl (1986). On the order between quantifiers. In Furberg M. et al., editors, *Acta Universitatis Gothoburgensis*, 273–285.

T. E. Zimmermann (1993). Scopeless quantifiers and operators. *Journal of Philosophical Logic*, **22**:545–561.

Chapter 3

Variable-free reasoning on finite trees

PATRICK BLACKBURN*, BERTRAND GAIFFE†, MAARTEN MARX‡

ABSTRACT.

In this paper we examine three modal languages that have been proposed in the model theoretic syntax literature for describing finite ordered trees. We compare their expressive power, and then examine a key complexity-theoretic issue: how expensive it is to decide — given a theory specifying a certain class of trees — whether a formula describes a model? Our main result is that for the languages proposed by Blackburn *et al.* and Palm this problem is EXPTIME-complete.

3.1 Introduction

Model theoretic syntax is an uncompromisingly declarative approach to natural language syntax: grammatical theories are logical theories, and grammatical structures are their models. Perhaps the best known work in this tradition is that of James Rogers (for example Rogers (1998)) in which grammatical theories are stated in monadic second-order logic. However other authors (in particular Kracht (1995, 1997), Blackburn and Meyer-Viol (1994) and Palm (1999)) use various kinds of *modal logic* (in essence, variable free formalisms for describing relational strcutures) to specify grammatical constraints. Palm (1999) contains some interesting linguistic examples and is a good introduction to (and motivation for) this approach.

In this paper we examine the modal languages proposed by Kracht, Palm, and Blackburn *et al.* for describing models based on finite trees. We compare their expressive power, and then examine a key complexity-theoretic issue: how expensive it is to decide — given a theory specifying a certain class of trees — whether a formula describes a model? Our main result is that for the languages of Blackburn *et al.* and Palm this problem is complete for the class of problems solvable in exponential time.

*Langue et Dialogue, LORIA, Nancy, France; patrick@aplog.org.
†Langue et Dialogue, LORIA, Nancy, France; gaiffe@loria.fr
‡ILLC, Universiteit van Amsterdam, The Netherlands; marx@science.uva.nl

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

3.2 The Languages $\mathscr{L}_B, \mathscr{L}_P$ and $\mathscr{L}_K$

We first recall the definitions of three modal languages proposed in the model-theoretic syntax literature for specifying declarative constraints on ordered trees. We start with the strongest, proposed by Marcus Kracht in Kracht (1995, 1997). The language will be called $\mathscr{L}_K$ (K for Kracht).

$\mathscr{L}_K$ is a propositional modal language identical to Propositional Dynamic Logic (PDL) Harel et al. (2000) over four basic programs $\leftarrow$, $\rightarrow$, $\uparrow$ and $\downarrow$, which explore the left-sister, right-sister, mother-of and daughter-of relations. Recall that PDL has two sorts of expressions: programs and propositions. We suppose we have fixed a non-empty, finite or countably infinite, set of atomic symbols A whose elements are typically denoted by p. $\mathscr{L}_K$'s syntax is as follows, writing π for programs and ϕ for propositions:

$$\begin{array}{lll} \pi & ::= & \leftarrow \mid \rightarrow \mid \uparrow \mid \downarrow \mid \pi;\pi \mid \pi \cup \pi \mid \pi^* \mid ?\phi \\ \phi & ::= & p \mid \top \mid \neg\phi \mid \phi \wedge \phi \mid \langle\pi\rangle\phi. \end{array}$$

We sometimes write $\mathscr{L}_K(\mathrm{A})$ to emphasize the dependence on A. We employ the usual boolean abbreviations and use $[\pi]\phi$ for $\neg\langle\pi\rangle\neg\phi$.

We interpret $\mathscr{L}_K(\mathrm{A})$ on *finite ordered trees* whose nodes are *labeled* with symbols drawn from A. We assume that the reader is familiar with finite trees and such concepts as 'daughter-of', 'mother-of', 'sister-of', 'root-node', 'terminal-node', and so on. If a node has no sister to the immediate right we call it a last node, and if it has no sister to the immediate left we call it a first node. Note that the root node is both first and last. The root node will always be called *root*. A labeling of a finite tree associates a subset of A with each tree node.

Formally, we present finite ordered trees as tuples $\mathbf{T} = (T, R_\rightarrow, R_\downarrow)$. Here T is the set of tree nodes and $R_\rightarrow$ and $R_\downarrow$ are the right-sister and daughter-of relations respectively. A pair $\mathfrak{M} = (\mathbf{T}, V)$, where $\mathbf{T}$ is a finite tree and $V : A \longrightarrow \mathit{Pow}(T)$, is called a *model*, and we say that V is a *labeling function* or a *valuation*. Given a model $\mathfrak{M}$, we simultaneously define a set of relations on $T \times T$ and the interpretation of the language $\mathscr{L}_K(A)$ on $\mathfrak{M}$:

$$\begin{array}{llllll} R_\uparrow & = & R_\downarrow^{-1} & R_{\pi\cup\pi'} & = & R_\pi \cup R_{\pi'} \\ R_\leftarrow & = & R_\rightarrow^{-1} & R_{\pi;\pi'} & = & R_\pi \circ R_{\pi'} \\ R_{\pi^*} & = & R_\pi^* & R_{?\phi} & = & \{(t,t) \mid \mathfrak{M}, t \models \phi\}. \end{array}$$

$$\begin{array}{rll} \mathfrak{M}, t \models p & \text{iff} & t \in V(p), \text{ for all } p \in \mathrm{A} \\ \mathfrak{M}, t \models \top & \text{iff} & t \in T \\ \mathfrak{M}, t \models \neg\phi & \text{iff} & \mathfrak{M}, t \not\models \phi \\ \mathfrak{M}, t \models \phi \wedge \psi & \text{iff} & \mathfrak{M}, t \models \phi \text{ and } \mathfrak{M}, t \models \psi \\ \mathfrak{M}, t \models \langle\pi\rangle\phi & \text{iff} & \exists t' \, (t R_\pi t' \text{ and } \mathfrak{M}, t' \models \phi). \end{array}$$

If $\mathfrak{M}, t \models \phi$, then we say ϕ is *satisfied* in $\mathfrak{M}$ at t. For any formula ϕ, if there is a model $\mathfrak{M}$ such that $\mathfrak{M}, \mathit{root} \models \phi$, then we say that ϕ is *satisfiable*. For Γ a set of formulas, and ϕ

a formula, we say that ϕ is a consequence of Γ (denoted by $\Gamma \models \phi$) if for every model in which Γ is satisfied at every node, ϕ is also satisfied at every node.

Below are two examples of such formulas: (3.2.1) says that every *a* node has a *b* and a *c* daughter, in that order, and no other daughters; and (3.2.2) says that every *a* node has a *b* first daughter followed by some number of *c* daughters, and no other daughters.

$$(3.2.1) a \quad \rightarrow \quad \langle\downarrow\rangle(\neg\langle\leftarrow\rangle\top \wedge b \wedge \langle\rightarrow\rangle(c \wedge \neg\langle\rightarrow\rangle\top))$$

$$(3.2.2) a \quad \rightarrow \quad \langle\downarrow\rangle(\neg\langle\leftarrow\rangle\top \wedge b \wedge \langle(\rightarrow;?c)^*\rangle\neg\langle\rightarrow\rangle\top).$$

A final remark. Note that we could have generated the same language by taking $\downarrow$ and $\rightarrow$ as primitive programs and closing the set of programs under converses.

Two more languages The two other languages proposed in the literature only differ from $\mathscr{L}_K$ in the programs they allow.

The language proposed by Blackburn, Meyer–Viol and de Rijke (1996), here called $\mathscr{L}_B$, is the weakest. It contains only the four basic programs plus their *transitive* closures, denoted by a superscript $(\cdot)^+$. This language is precisely as expressive as the language generated by the following programs:

$$\pi ::= \leftarrow \mid \rightarrow \mid \uparrow \mid \downarrow \mid \pi^*.$$

To see this, note that for $\pi \in \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$, the transitive closure operator is expressible by, $\langle\pi^+\rangle\phi \equiv \langle\pi\rangle\langle\pi^*\rangle\phi$. For the other direction, note that $\langle(\pi^*)^*\rangle\phi \equiv \langle\pi^*\rangle\phi$ and $\langle\pi^*\rangle\phi \equiv \phi \vee \langle\pi^+\rangle\phi$

The language proposed by Palm (1999), here called $\mathscr{L}_P$, lies between $\mathscr{L}_B$ and $\mathscr{L}_K$ with respect to expressive power. It is generated by the following programs[1]:

$$\pi ::= \leftarrow \mid \rightarrow \mid \uparrow \mid \downarrow \mid ?\phi;\pi \mid \pi^*.$$

Palm tried to designed his language to have exactly the expressive power required to reason about syntactical structures. At first glance, $\mathscr{L}_P$ seems rather weak compared with Kracht's language, for it lacks the composition, union and test operator constructors. However note that when these are applied outside of the scope of the Kleene star they are definable as follows: $\langle\pi;\pi'\rangle\phi \equiv \langle\pi\rangle\langle\pi'\rangle\phi$, $\langle\pi \cup \pi'\rangle\phi \equiv \langle\pi\rangle\phi \vee \langle\pi'\rangle\phi$, and $\langle?\psi\rangle\phi \equiv \psi \wedge \phi$. Palm claims that "The resulting 'tense' fragment of PDL holds sufficient expressivity to handle the linguistic demands on tree constraints".

Palm calls his language *Propositional Tense Logic for Finite Trees*, making an analogy with branching time logic. In branching time logic, $\langle\downarrow^*\rangle\phi$ and $\langle\uparrow^*\rangle\phi$ are called *sometimes in the future* ϕ and *sometimes in the past* ϕ, respectively. But besides these unary operators, branching time logic standardly makes use of the binary *until* and *since* connectives. *Until* is defined as: $\mathfrak{M},t \models U(\phi,\psi)$ iff there exists a time t' in the future of t with $\mathfrak{M},t' \models \phi$ and for all time points t'' in between t and t' it holds that $\mathfrak{M},t'' \models \psi$. *Since* has an analogous definition, but toward the past.

In fact, Palm's choice of the name *Tense Logic* is apt, for as we shall now see $\mathscr{L}_P$ is nothing but the simplest language $\mathscr{L}_B$ with four additional *until* operators defined as

[1] Palm's conditional paths π_ϕ are denoted here as $?\phi;\pi$.

follows. For $\pi \in \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$, $\mathfrak{M}, t \models U_\pi(\phi, \psi)$ iff there exists a t' such that $tR_{\pi^*}t'$ and $\mathfrak{M}, t' \models \phi$ and for all t'' such that $tR_{\pi^*}t''R_\pi t'$ it holds that $\mathfrak{M}, t'' \models \psi$. $U_\pi(\phi, \psi)$ is a very natural operation. For instance, the program `while` ϕ `do` π is expressed by $U_\pi(\neg\phi, \phi)$.

Theorem 3.2.1. *The language $\mathscr{L}_P$ is precisely as expressive as the language $\mathscr{L}_B$ with the additional four until programs.*

Proof: For one direction, note that for $\pi \in \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$, $U_\pi(\phi, \psi) \equiv \langle(?\psi; \pi)^*\rangle\phi$, and the right hand side is a Palm formula. For the other direction, we use induction on the complexity of the programs in $\mathscr{L}_P$ formulas. Inside this proof $\mathscr{L}_{until}$ denotes the language $\mathscr{L}_B$ with the additional four U_π programs.

Consider the formula $\langle\pi\rangle\phi$. There are three cases. If π is a basic program, then $\langle\pi\rangle\phi \in \mathscr{L}_{until}$. In the second case, π is of the form $?\psi; P$, for P a program. If P is a basic program, then $\langle\pi\rangle\phi = \langle?\psi; P\rangle\phi$ is equivalent to $\psi \wedge \langle P\rangle\phi$ which is in $\mathscr{L}_{until}$. If P itself is of the form $?\theta; P'$, then $\langle\pi\rangle\phi = \langle?\psi; (?\theta; P')\rangle\phi$ which is equivalent to $\langle?(\psi \wedge \theta); P'\rangle\phi$. Now P' is of smaller complexity than P, whence by inductive hypothesis, the last formula is equivalent to a formula in $\mathscr{L}_{until}$. Finally if P is of the form Q^*, then $\langle\pi\rangle\phi = \langle?\psi; Q^*\rangle\phi$ which is equivalent to $\psi \wedge \langle Q^*\rangle\phi$, which by IH then is equivalent to a formula in $\mathscr{L}_{until}$. In the third and last case, π is of the form P^*. If P is a basic program, $\langle\pi\rangle\phi \in \mathscr{L}_{until}$. If P itself is of the form Q^*, then $\langle\pi\rangle\phi = \langle(Q^*)^*\rangle\phi \equiv \langle Q^*\rangle\phi$ which then by IH is equivalent to a formula in $\mathscr{L}_{until}$. If P is of the form $?\psi; Q$, then if Q is atomic $\langle\pi\rangle\phi = \langle(?\psi; Q)^*\rangle\phi \equiv U_Q(\phi, \psi)$, whence in $\mathscr{L}_{until}$. If Q is of the form $?\theta; Q'$ it reduces as before. If $Q = (Q')^*$, then $\langle\pi\rangle\phi = \langle(?\psi; (Q')^*)^*\rangle\phi$ which is equivalent to $\phi \vee (\psi \wedge \langle(Q')^*\rangle\phi)$, which by IH is equivalent to a formula in $\mathscr{L}_{until}$. □

Actually, one can be even more economic in defining this extension of $\mathscr{L}_B$. Let us redefine $\mathscr{L}_{until}$ to be the modal language with the following four binary modal operators: for $\pi \in \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$, $\mathfrak{M}, t \models \mathit{Until}_\pi(\phi, \psi)$ iff there exists a t' such that $tR_{\pi^+}t'$ and $\mathfrak{M}, t' \models \phi$ and for all t'' such that $tR_{\pi^+}t''R_{\pi^+}t'$ it holds that $\mathfrak{M}, t'' \models \psi$. Then $\langle\pi\rangle\phi$ and $\langle\pi^*\rangle\phi$ can be defined to be $\mathit{Until}_\pi(\phi, \bot)$ and $\phi \vee \mathit{Until}_\pi(\phi, \top)$, respectively. The previous (non strict) until construct $U_\pi(\phi, \psi)$ is equivalent to $\phi \vee (\psi \wedge \mathit{Until}_\pi(\phi, \psi))$.

Let us briefly discuss the relationship between the modal languages discussed in this paper and the first order logic of ordered trees. Let $\mathscr{L}_{FO}$ denote the first order language over the signature with binary predicates $\{R_\downarrow, R_\leftarrow, R_{\downarrow^*}, R_{\leftarrow^*}\}$ and countably many unary predicates. $\mathscr{L}_{FO}$ is interpreted on ordered trees in the obvious way, with $R_\downarrow$ being the daughter relation, and so on. Kracht's language $\mathscr{L}_K$ can express properties beyond the power of $\mathscr{L}_{FO}$. For examples, it can express the property of having an odd number of daughters:

(3.2.3) $\langle\downarrow\rangle(\neg\langle\leftarrow\rangle\top \wedge \langle(\rightarrow;\rightarrow)^*\rangle\neg\langle\rightarrow\rangle\top)$.

On the other hand, Theorem 3.2.1 entails that every Palm formula is equivalent to a formula $\phi(x)$ in $\mathscr{L}_{FO}$; we simply use the *standard translation* of until into $\mathscr{L}_{FO}$ (see Blackburn *et al* (2001). We conjecture that the converse is also true: $\mathscr{L}_P$ is functionally complete with respect to $\mathscr{L}_{FO}$. For unordered trees such a result (generalizing Kamp's famous theorem to trees) can be found in Schlingloff (1992).

Theorem 3.2.1 together with (3.2.3) entail that $\mathscr{L}_P$ is strictly contained in $\mathscr{L}_K$. That $\mathscr{L}_B$ is strictly contained in $\mathscr{L}_P$ follows from the well known fact that until is not expressible on linear orders from the future and past modalities. For a concrete example of difference in expressive power, note that the property of having exactly 2 p daughters is expressible in $\mathscr{L}_P$:

(3.2.4) $\langle\downarrow\rangle(p \wedge \neg\langle\leftarrow^+\rangle p \wedge \langle\rightarrow\rangle\langle(?\neg p;\rightarrow)^*\rangle(p \wedge \neg\langle\rightarrow^+\rangle p))$

However an easy *bisimulation* argument (see Blackburn *et al* (2001) for the definition of bisimulation) can be used to show that $\mathscr{L}_B$ cannot express this property.

We conclude this section with a summary of the relative expressive power of the languages we have discussed:

- $\mathscr{L}_B \subsetneq \mathscr{L}_P = \mathscr{L}_{until} \subsetneq \mathscr{L}_K$.
- $\mathscr{L}_P \subseteq \mathscr{L}_{FO}$ and $\mathscr{L}_K \not\subset \mathscr{L}_{FO}$.
- **Conjecture:** $\mathscr{L}_{until} = \mathscr{L}_{FO}$.

3.3 Complexity

In model theoretic syntax we specify a certain class of trees by stating a theory Θ in a tree language (Θ is our grammatical theory). Thus a key question is: given a formula ϕ, does ϕ describe a structure that is grammatical with respect to this theory? More formally: does there exist a model $\mathfrak{M}$ such that $\mathfrak{M}$ is a model of Θ (i.e., every formula in Θ is true at every node in $\mathfrak{M}$) *and* $\mathfrak{M}$ satisfies ϕ (i.e., ϕ is true at the root of $\mathfrak{M}$)? This holds iff $\Theta \not\models \mathit{root} \rightarrow \neg\phi$. (Here and below we also use *root* to denote the formula $\neg\langle\uparrow\rangle\top$, which indeed is satisfied at the root of a tree only.) This is the type of problem we will study. For L a language, the L consequence problem consists of all pairs (Γ,χ) with $\Gamma\cup\{\chi\}$ a finite set of L formulas such that $\Gamma \models \chi$. We now study the complexity of this problem for $\mathscr{L}_B, \mathscr{L}_P$ and $\mathscr{L}_K$.

Decidability of the $\mathscr{L}_K$ consequence problem is shown in Kracht (1997), Theorem 5, via a reduction to the $\mathscr{L}_B$ consequence problem. Unfortunately the reduction is not correct (a counterexample is given in the Appendix to this paper). However $\mathscr{L}_K$ decidability can be proved by interpreting it in $L^2_{K,P}$, the monadic second order logic of variably branching trees of Rogers (1998). (The decidability of the satisfiability problem for $L^2_{K,P}$ follows, in turn, via an interpretation into $S\omega S$.) The translation of $\mathscr{L}_K$ formulas into $L^2_{K,P}$ is straightforward. Note, in particular, that we can use second order quantification to define the transitive closure of a relation: for R any binary relation, xR^*y holds iff

$$x = y \vee \forall X(X(x) \wedge \forall z,z'(X(z) \wedge zRz' \rightarrow X(z')) \rightarrow X(y)).$$

Note that although this reduction yields $\mathscr{L}_K$ decidability, it only gives us a non elementary decision procedure.

What of the complexity of these consequence problems? In Blackburn et al. (1996) the problem for $\mathscr{L}_B$ was claimed to be in EXPTIME[2], but the proof contains a mistake. Here we show that the claim is indeed correct, and that the same result holds for the language $\mathscr{L}_P$. Before we go into the proof details we consider the problem in a bit more detail. We first look at the lower bound:

Theorem 3.3.1. *The consequence problem for the language with only* $\downarrow$ *is EXPTIME–hard.*

Proof: This is an immediate corollary of Spaan (1993) analysis of the lower bound result for PDL. She notes that the following fragment of PDL is EXPTIME-hard: formulas of the form $\psi \wedge [a^*]\theta$, (where ψ and θ contain only the atomic program a and no embedded modalities) that are satisfiable at the root of a finite binary tree. Identifying the program a with $\downarrow$, the result follows (because $[\downarrow^*]\theta \wedge \psi$ is satisfiable at the root of a finite tree iff $\theta \not\models \mathit{root} \rightarrow \neg\psi$). □ For full PDL this bound is optimal. There is even a stronger result: every satisfiable PDL formula ϕ can be satisfied on a model with size exponential in the length of ϕ. Unfortunately with tree-based models there is no hope for such a result:

> For every natural number n, there exists a satisfiable formula of size $\mathscr{O}(n^2)$ in the language with only $\downarrow$ and $\downarrow^*$ which can only be satisfied on at least binary branching trees of depth at least 2^n.

A formula which forces the deep branch is given in Blackburn et al. (2001): Proposition 6.51; one only has to add the conjunct $[\downarrow^*](\langle\downarrow\rangle p \wedge \langle\downarrow\rangle\neg p)$ for some new variable p to enforce binary branching. Note that the size of the model is double exponential in the size of the formula. This means that a decision algorithm which tries to construct a tree model must run at least in exponential space, as it will need to keep a whole branch in memory.

Fortunately we can do better, taking a cue from the completeness proof for a related language in Blackburn and Meyer-Viol (1994). Instead of constructing a model we design an algorithm which searches for a "good" set of labelings of the nodes of a model. Label sets consist of subformulas of the formula ϕ whose satisfiability is to be decided. From a good set of labels we can construct a labeled tree model which satisfies ϕ. The gain in complexity comes from the fact that the number of labels is bound by an exponential in the number of subformulas of ϕ. As we shall show, the search for a good set of labels among the possible ones can be implemented in time polynomial in the number of possible labels using the technique of elimination of Hintikka sets developed by Pratt (1979). Thus we will be able to prove:

Theorem 3.3.2. *The* $\mathscr{L}_P$ *consequence problem is in EXPTIME.*

The proof of Theorem 3.3.2 consists of a reduction and a decision algorithm. The reduction combines ideas from Kracht (1997), Theorem 5 and Rabin's reduction of $S\omega S$ to $S2S$.

[2]EXPTIME is the class of all problems solvable in exponential time. A problem is solvable in exponential time if there is a deterministic exponentially time bounded Turing machine that solves it. A deterministic Turing machine is exponentially time bounded if there is a polynomial $p(n)$ such that the machine always halts after at most $2^{p(n)}$ steps, where n is the length of the input.

Let $\mathscr{L}_2$ be the modal language with only the two programs $\{\downarrow_1, \downarrow_2\}$ and the modal constant *root*. $\mathscr{L}_2$ is interpreted on finite ordered *binary* trees, with $\downarrow_1$ and $\downarrow_2$ interpreted by the first and second daughter relation, respectively, and *root* holds exactly at the root. We present such trees by triples $(T, \succ_1, \succ_2)$.

Lemma 3.3.3. There is an effective reduction from the $\mathscr{L}_P$ consequence problem to the $\mathscr{L}_2$ consequence problem.

The proof is provided in the appendix. The theorem now follows from the previous lemma together with the following one, which we shall prove in the next section:

Lemma 3.3.4. The $\mathscr{L}_2$ consequence problem is in EXPTIME.

3.4 Deciding $\mathscr{L}_2$

We will give an EXPTIME algorithm that on input $\mathscr{L}_2$ formulas γ, χ decides whether there exists a model $\mathfrak{M}$ in which γ is true everywhere and χ is true at the root. To this the consequence problem reduces because $\gamma \not\models \phi$ iff there exists a model in which $\gamma \wedge (p \leftrightarrow \neg\phi \vee \langle\downarrow_1\rangle p \vee \langle\downarrow_2\rangle p)$ is true everywhere and p is true at the root. Here p is a new propositional variable whose intended meaning is $\langle(\downarrow_1 \cup \downarrow_2)^*\rangle\neg\phi$.

Preliminaries. Recall that a set of formulas Σ is said to be closed under subformulas iff for all $\phi \in \Sigma$, if ψ is a subformula of ϕ then $\psi \in \Sigma$. It is closed under single negations if whenever ϕ is in the set and ϕ is not of the form $\neg\psi$ then also $\neg\phi$ is in the set. For Σ a set of formulas, $Cl(\Sigma)$ (called the *closure* of Σ) is defined to be the smallest set of formulas containing Σ that is closed under subformulas and single negations and which contains the constant *root* and the formulas $\langle\downarrow_1\rangle\top$ and $\langle\downarrow_2\rangle\top$. From now on we fix two arbitrary formulas γ and χ.

Definition 3.4.1 (Hintikka Set). *Let $A \subseteq Cl(\{\gamma, \chi\})$. We call A a Hintikka Set if A satisfies the following conditions:*

1. *$\gamma \in A$ and $\top \in A$.*
2. *If $\phi \in Cl(\{\gamma, \chi\})$ then $\phi \in A$ iff $\neg\phi \notin A$.*
3. *If $\phi \wedge \psi \in Cl(\{\gamma, \chi\})$ then $\phi \wedge \psi \in A$ iff $\phi \in A$ and $\psi \in A$.*
4. *$\langle\downarrow_1\rangle\top \in A$ iff $\langle\downarrow_2\rangle\top \in A$.*

Let $HS(\gamma, \chi)$ denote the set of all Hintikka Sets which are a subset of $Cl(\gamma, \chi)$. Note that $|HS(\gamma, \chi)| \leq 2^{|Cl(\gamma,\chi)|}$.

For H a set of Hintikka sets, let $l : H \longrightarrow \{0, 1, \ldots, |H|\}$ be a function assigning to each $A \in H$ a level. We call a structure (H, l) an ordered set of Hintikka sets.

Definition 3.4.2 (Saturation). *Let (H, l) be an ordered set of Hintikka sets and let k be either 1 or 2. We call (H, l)* saturated *if for all $A \in H$, $\langle\downarrow_k\rangle\phi \in A$ only if there exists a $B \in H$ such that $l(A) > l(B)$ and for all $\langle\downarrow_k\rangle\psi \in Cl(\gamma, \chi)$, $\langle\downarrow_k\rangle\psi \in A$ iff $\psi \in B$.*

The connection. We are ready to formulate our most important lemma.

Lemma 3.4.1. The following are equivalent:

1. There exists a model over a finite binary branching tree in which γ is true everywhere and χ is true at the root;
2. There exists a saturated ordered set of Hintikka Sets (H,l), with $H \subseteq HS(\gamma,\chi)$ and there is an $A \in H$ with $\{root,\chi\} \subseteq A$.

Proof: First assume $\mathfrak{M}$ is a model over a finite binary branching tree in which γ is true everywhere and χ is true at the root. For each node n define $A_n = \{\psi \in Cl(\gamma,\chi) \mid \mathfrak{M},n \models \psi\}$. Obviously each A_n is a Hintikka set and there is an A with $\{root,\chi\} \subseteq A$. Let H be the set of all such A_n. Let $\hat{A}_n$ abbreviate the conjunction of all formulas in A_n. Inductively define the level function on H. First define which Hintikka Sets are of level 0:

$$l(A) = 0 \text{ if } t \in A.$$

Next, suppose the i-th level is defined. First define: $S_i = \{A \in H \mid l(A) \leq i\}$. Next, if $H \backslash S_i$ is non-empty then the $i+1$-th level is defined as follows: $l(A) = i+1$ if $A \notin S_i$ and

$$\mathfrak{M}, root \models \langle(\downarrow_1 \cup \downarrow_2)^*\rangle(\hat{A} \wedge [\downarrow_1 \cup \downarrow_2][(\downarrow_1 \cup \downarrow_2)^*] \bigvee_{B \in S_i} \hat{B}).$$

On the other hand, if $H \backslash S_i$ is empty then there is no $i+1$-th level. It is not hard to show that (H,l) is saturated.

Now assume (H,l) is saturated and there is an $A_0 \in H$ with $\{root,\chi\} \subseteq A_0$.

We inductively construct a finite binary tree and a function h from the nodes to H in such a way that we can turn the tree into a model $\mathfrak{M}$ for which we can prove the truth lemma,

$$\text{for all } \psi \in Cl(\gamma,\chi),\ \mathfrak{M},n \models \psi \text{ if and only if } \psi \in h(n).$$

By the first condition on Hintikka sets and the existence of $A_0 \in H$, this yields a model in which γ is true everywhere and χ at the root. Let $\mathscr{T}$ be some denumerably infinite set; we shall use (finitely many) of its elements as the tree nodes.

Stage 0. Define T_0 to be $\{t_0\}$; $\succ_1^0$ to be $\emptyset$; $\succ_2^0$ to be $\emptyset$; and h_0 to be $\{\langle w_0,A_0\rangle\}$.

Stage $n+1$. Suppose n stages of the inductive construction have been performed. We call a pair $\langle t,k\rangle$ (where $t \in T_n$ and $k \in \{1,2\}$) an *unsatisfied demand* iff $\langle\downarrow_k\rangle\phi \in h(t)$ but there is no $t' \in T_n$ such that $t \succ_k t'$. If there are no unsatisfied demands the construction is complete. Otherwise, choose an unsatisfied demand $\langle t,k\rangle$. As (H,l) is saturated there exists a $B \in H$ such that $l(h_n(t)) > l(B)$ and for all $\langle\downarrow_k\rangle\psi \in Cl(\Sigma)$, $\langle\downarrow_k\rangle\psi \in h_n(t)$ iff $\psi \in B$. Let $t' \in \mathscr{T} \backslash T_n$. Define:

$$\begin{aligned} T_{n+1} &= T_n \cup \{t'\} \\ \succ_k^{n+1} &= \succ_k^n \cup \{\langle t, t'\rangle\} \\ \succ_j^{n+1} &= \succ_j^n \\ h_{n+1} &= h_n \cup \{\langle t', B\rangle\}. \end{aligned}$$

While adjoining a new node t' to t as described in the inductive step may result in new unsatisfied demands $\langle t', k\rangle$, where $k \in \{1,2\}$, we were careful to choose $h(t')$ from a strictly lower level than $h(t)$. This means that in the course of the construction we will be forced to map the newly adjoined node t' to a Hintikka set of level zero; but doing so cannot give rise to an unsatisfied demand. Thus the construction process terminates.

Let $\langle T, \succ_1, \succ_2\rangle$ be the result of the final stage. Note that by the last condition on Hintikka Sets and the fact that there are no unsatisfied demands every non leaf node has two daughters. Turn $\langle W, \succ_1, \succ_2\rangle$ into a model $\mathfrak{M} = \langle W, \succ_1, \succ_2, V\rangle$ by setting $n \in V(p)$ iff $p \in h(n)$.

Prove the truth lemma by an induction on the complexity of the formulas. The base case is by definition of V. The boolean cases are by conditions 2 and 3 on Hintikka sets. The cases for $\langle\downarrow_k\rangle$ follow from the fact that all demands are satisfied.

□

The algorithm. The decision algorithm for $\mathscr{L}_2$ satisfiability is presented in Figure 3.1. Its most important properties are presented in the next lemma.

Lemma 3.4.2. 1. *Elimination of* $HS(\gamma,\chi)$ terminates after at most $|HS(\gamma,\chi)|$ rounds of the do loop.
2. The statement "$\langle$`S, l`$\rangle$ is a saturated ordered set of Hintikka sets" holds after the do loop of *Elimination of* $HS(\gamma,\chi)$.

Proof: (1) The bound function of the do loop is the size of `Pool` which is being reduced in every round, or the loop terminates because `L`$= \emptyset$. The initial size of `Pool` is bounded by $|HS(\gamma,\chi)| = 2^{|Cl(\gamma,\chi)|}$.
(2) Because the statement "$HS(\gamma,\chi)$ = `Pool` $\uplus$ `S` and $\langle$`S, l`$\rangle$ is a saturated ordered set of Hintikka sets" holds before the do loop and is an invariant of the do loop. □ This lemma

immediately yields our desired result:

PROOF OF LEMMA 3.3.4. In order to decide whether there exists a model is which γ is true everywhere and χ is true at the root we run *Elimination of* $HS(\gamma,\chi)$. The algorithm is correct by Lemma 3.4.1 and part 2 of Lemma 3.4.2. By part 1 the algorithm terminates after at most $|HS(\gamma,\chi)| \leq 2^{|Cl(\gamma,\chi)|}$ rounds of the do loop. As in Pratt (1979), the tests inside the do loop take time bounded by $p(|HS(\gamma,\chi)|)$ for some polynomial p. Since $|Cl(\gamma,\chi)|$ is linear in the number of subformulas of γ,χ, the algorithm is in EXPTIME. QED

```
begin
   L    := {A ∈ HS(γ,χ) | ¬⟨↓1⟩⊤ ∈ A};
   Pool:= HS(γ,χ)\ L;
   S    := L;
   i    := 0;
   l    := {(A,i) | A ∈ L};
   do L ≠ ∅ →
      L    := {A ∈ Pool | (S ∪ {A}, l ∪ (A, i + 1))
              is a saturated ordered set of
              Hintikka Sets };
      Pool:= Pool \ L;
      S    := S ∪ L;
      i    := i+1;
      l    := l ∪ {(A,i) | A ∈ L}
   od;
   if ∃A ∈ S: {χ, root} ⊆ A
      then true
      else fail
   fi
end
```

Figure 3.1: The algorithm *elimination of* $HS(\gamma,\chi)$.

3.5 Conclusions

We discussed the relative expressivity of three modal languages proposed for specifying grammatical constraints on finite ordered trees. We added a fourth language, $\mathscr{L}_{until}$, and conjectured it to be precisely as expressive as the first order language of ordered trees. We showed that the consequence problems for $\mathscr{L}_B$, $\mathscr{L}_P$ and $\mathscr{L}_{until}$ are EXPTIME-complete. We conjecture that the same bound holds for $\mathscr{L}_K$ as well (note that if Kracht's polynomial reduction of $\mathscr{L}_K$ satisfiability to $\mathscr{L}_B$ satisfiability can be repaired, this follows immediately from the results in this paper).

Palm argued that writing grammatical constraints in the language $\mathscr{L}_P$ is straightforward and yields formulas which are simpler and easier to understand than first order formulas. We think this is due to the lack of variables and the direct use of the "tree-axis" in $\mathscr{L}_P$ formulas. It is interesting to note that the language XPath contains exactly these two features. XPath was designed to extract elements from XML documents, and the natural models of XML documents are finite ordered trees.

Acknowledgements This work was carried out as part of the INRIA funded research partnership between LIT (Language and Inference Technology Group, University of Amsterdam) and LED (Langue et Dialogue, LORIA, Nancy). Marx is supported by NWO grant 612.000.106.

Bibliography

Blackburn, P., M. de Rijke, and Y. Venema (2001). *Modal Logic*. Cambridge University Press.

Blackburn, P. and W. Meyer-Viol (1994). Linguistics, logic, and finite trees. *Logic Journal of the IGPL*, **2**:3–29.

Blackburn, P., W. Meyer-Viol, and M. de Rijke (1996). A proof system for finite trees. In H. K. Büning, ed., *Computer Science Logic*, volume 1092 of *LNCS*, pp. 86–105. Springer.

Harel, D., D. Kozen, and J. Tiuryn (2000). *Dynamic Logic*. MIT Press.

Kracht, M. (1995). Syntactic codes and grammar refinement. *Journal of Logic, Language and Information*, **4**:41–60.

Kracht, M. (1997). Inessential features. In C. Retore, ed., *Logical Aspects of Computational Linguistics*, number 1328 in LNAI, pp. 43–62. Springer.

Palm, A. (1999). Propositional tense logic for trees. In *Sixth Meeting on Mathematics of Language*. University of Central Florida, Orlando, Florida.

Pratt, V. (1979). Models of program logics. In *Proceedings of the 20th IEEE symposium on Foundations of Computer Science*, pp. 115–122.

Rogers, J. (1998). *A descriptive approach to language theoretic complexity*. CSLI Press.

Schlingloff, B.-H. (1992). Expressive completeness of temporal logic of trees. *Journal of Applied Non–Classical Logics*, **2**(2):157–180.

Spaan, E. (1993). *Complexity of modal logics*. Ph.D. thesis, University of Amsterdam, Institute for Logic, Language and Computation.

Weyer, M. (2002). Decidability of S1S and S2S. In E. G. et al., ed., *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*, pp. 207–230. Springer.

Appendix

Counterexample to Kracht's reduction. We present a counterexample to the reduction from $\mathscr{L}_K$ to $\mathscr{L}_B$ given in the proof of Theorem 5 in Kracht (1997). Take the following non satisfiable formula $\langle(\downarrow^*)^*\rangle\bot$. Then $\nabla(\psi)$ is

$$\begin{array}{lcl} q_\bot & \leftrightarrow & \bot \\ q_{\langle(\downarrow^*)^*\rangle\bot} & \leftrightarrow & q_\bot \vee q_{\langle\downarrow^*\rangle\langle(\downarrow^*)^*\rangle\bot} \\ q_{\langle\downarrow^*\rangle\langle(\downarrow^*)^*\rangle\bot} & \leftrightarrow & q_{\langle(\downarrow^*)^*\rangle\bot} \vee q_{\langle\downarrow\rangle\langle\downarrow^*\rangle\langle(\downarrow^*)^*\rangle\bot} \\ q_{\langle\downarrow\rangle\langle\downarrow^*\rangle\langle(\downarrow^*)^*\rangle\bot} & \leftrightarrow & \langle\downarrow\rangle q_{\langle\downarrow^*\rangle\langle(\downarrow^*)^*\rangle\bot}. \end{array}$$

$q_{\langle(\downarrow^*)^*\rangle\bot}$ can be made true in the tree with domain $\{0,00\}$, with 0 the root and 00 her daughter and the following valuation:

$$\begin{array}{lcl} V(0) & = & \{q_{\langle\downarrow\rangle\langle\downarrow^*\rangle\langle(\downarrow^*)^*\rangle\bot}, q_{\langle\downarrow^*\rangle\langle(\downarrow^*)^*\rangle\bot}, q_{\langle(\downarrow^*)^*\rangle\bot}\} \\ V(00) & = & \{q_{\langle\downarrow^*\rangle\langle(\downarrow^*)^*\rangle\bot}, q_{\langle(\downarrow^*)^*\rangle\bot}\} \end{array}$$

This model makes $\nabla(\psi)$ true. But clearly we do not have $0 \models \langle(\uparrow;\downarrow)^*\rangle\bot$, contrary to Kracht's claim that for every node n, and for every formula χ in the Fisher Ladner closure it holds that $n \models \chi \leftrightarrow q_\chi$.

Proof of Lemma 3.3.3 Although Kracht's reduction of $\mathscr{L}_K$ to $\mathscr{L}_B$ is flawed, his approach can used to give a reduction of $\mathscr{L}_P$ to $\mathscr{L}_2$, and we shall do so here. Note that $\gamma_1,\ldots,\gamma_n \models \chi$ iff $\models [\downarrow^*](\gamma_1 \wedge \ldots \wedge \gamma_n) \rightarrow \chi$. Thus we need only reduce the consequence problem for empty Γ. The proof of Theorem 3.2.1 gives an effective reduction from $\mathscr{L}_P$ to $\mathscr{L}_{until}$ formulas.

Let $\chi \in \mathscr{L}_{until}$. Let $Cl(\chi)$ be the smallest set of formulas containing all subformulas of χ and which is closed under taking single negations and under the rule: $Until_\pi(\phi,\psi) \in Cl(\chi) \Rightarrow \psi \wedge Until_\pi(\phi,\psi) \in Cl(\chi)$.

We associate a formula $\nabla(\chi)$ with χ as follows. We create for each $\phi \in Cl(\chi)$, a new propositional variable q_ϕ. Now $\nabla(\chi)$ "axiomatizes" these new variables as follows:

$$\begin{array}{lcl} q_p & \leftrightarrow & p \\ q_{\neg\phi} & \leftrightarrow & \neg q_\phi \\ q_{\phi\wedge\psi} & \leftrightarrow & q_\phi \wedge q_\psi \\ q_{Until_\pi(\phi,\psi)} & \leftrightarrow & \langle\pi\rangle q_\phi \vee \langle\pi\rangle q_{(\psi\wedge Until_\pi(\phi,\psi))}. \end{array}$$

We claim that for every model $\mathfrak{M}$ which validates $\nabla(\chi)$, for every node n and for every subformula $\phi \in Cl(\chi)$, $\mathfrak{M},n \models q_\phi$ iff $\mathfrak{M},n \models \phi$.

The proof is by induction on the structure of the formula, and for the left to right direction of the until case by induction on the depth of direction of π. We do that case for $Until_\downarrow$. Let n be a leaf. By the axiom in $\nabla(\chi)$, $\mathfrak{M},n \not\models q_{Until_\downarrow(\phi,\psi)}$. But also $\mathfrak{M},n \not\models Until_\downarrow(\phi,\psi)$. Now let n be a node with $k+1$ descendants, and let the claim hold for nodes with k descendants. Let $\mathfrak{M},n \models q_{Until_\downarrow(\phi,\psi)}$. Then by the axiom $\mathfrak{M},n \models \langle\downarrow\rangle q_\phi$ or $\mathfrak{M},n \models \langle\downarrow\rangle q_{(\psi\wedge Until_\downarrow(\phi,\psi))}$. In the first case, there exists a daughter m of n and $\mathfrak{M},m \models q_\phi$. By inductive hypothesis, $\mathfrak{M},m \models \phi$, whence $\mathfrak{M},n \models Until_\downarrow(\phi,\psi)$. In the second case, there exists a daughter m of n and $\mathfrak{M},m \models q_\psi$ and $\mathfrak{M},m \models q_{Until_\downarrow(\phi,\psi)}$. Whence, by first inductive hypothesis, $\mathfrak{M},m \models \psi$ and the second inductive hypothesis $\mathfrak{M},m \models Until_\downarrow(\phi,\psi)$. But then also $\mathfrak{M},n \models Until_\downarrow(\phi,\psi)$. Hence the following holds for each $\chi \in \mathscr{L}_{until}$,

$$\models \chi \Leftrightarrow \nabla(\chi) \models q_\chi.$$

Note that the only modalities occurring in $\nabla(\chi)$ are $\langle\pi\rangle$ for π one of the four compass arrows. We can further reduce the number of arrows to only $\downarrow,\rightarrow$ when we add two modal constants *root* and *first* for the root and first elements, respectively.

Let χ be a formula in this fragment. As before create a new variable q_ϕ for each (single negation of a) subformula ϕ of χ. Create $\nabla(\chi)$ as follows:

$$\begin{array}{lcl} q_p & \leftrightarrow & p \\ q_{\neg\phi} & \leftrightarrow & \neg q_\phi \\ q_{\phi\wedge\psi} & \leftrightarrow & q_\phi \wedge q_\psi \\ q_{\langle\pi\rangle\phi} & \leftrightarrow & \langle\pi\rangle q_\phi \text{ for } \pi \in \{\downarrow,\rightarrow\}. \end{array}$$

And for each subformula $\langle\uparrow\rangle\phi$ and $\langle\leftarrow\rangle\phi$ we add to $\nabla\chi$ the axioms

$$\begin{array}{llllllll} q_\phi & \rightarrow & [\downarrow]q_{\langle\uparrow\rangle\phi}, & \langle\downarrow\rangle q_{\langle\uparrow\rangle\phi} & \rightarrow & q_\phi, & q_{\langle\uparrow\rangle\phi} & \rightarrow \neg root, \\ q_\phi & \rightarrow & [\rightarrow]q_{\langle\leftarrow\rangle\phi}, & \langle\rightarrow\rangle q_{\langle\leftarrow\rangle\phi} & \rightarrow & q_\phi, & q_{\langle\leftarrow\rangle\phi} & \rightarrow \neg first. \end{array}$$

We claim that for every model $\mathfrak{M}$ which validates $\nabla(\chi)$, for every node n and for every subformula $\phi \in Cl(\chi)$, $\mathfrak{M},n \models q_\phi$ iff $\mathfrak{M},n \models \phi$. An easy induction shows this. We do the case for $\langle\uparrow\rangle\phi$. If $n \models \langle\uparrow\rangle\phi$, then the parent of n models ϕ, whence by inductive hypothesis, it models q_ϕ, so by the axiom $q_\phi \rightarrow [\downarrow]q_{\langle\uparrow\rangle\phi}$, $n \Vdash q_{\langle\uparrow\rangle\phi}$. Conversely, if $n \models q_{\langle\uparrow\rangle\phi}$, then by axiom $q_{\langle\uparrow\rangle\phi} \rightarrow \neg root$, n is not the root. So the parent of n exists and it models $\langle\downarrow\rangle q_{\langle\uparrow\rangle\phi}$.

Then it models q_ϕ by axiom $\langle\downarrow\rangle q_{\langle\uparrow\rangle\phi} \to q_\phi$, and by inductive hypothesis it models ϕ. Thus $n \models \langle\uparrow\rangle\phi$. Hence, the following holds

$$\gamma \models \chi \Leftrightarrow \nabla(\gamma \wedge \chi), q_\gamma \models q_\chi.$$

Note that the formulas on the right hand side only contain the modalities $\langle\downarrow\rangle$ and $\langle\rightarrow\rangle$. Finally we reduce the consequence problem to that of binary branching trees. Let χ be a formula, let d and q_{first} be new variables. Let $(\cdot)'$ be the following translation:

$$\begin{array}{lcl}
p' & = & p \\
(\neg\phi)' & = & \neg\phi' \\
(\phi \wedge \psi)' & = & \phi' \wedge \psi' \\
(\langle\downarrow\rangle\phi)' & = & \langle\downarrow_1\rangle\langle(?d;\downarrow_2)^*\rangle(d \wedge \phi') \\
(\langle\rightarrow\rangle\phi)' & = & \langle\downarrow_2\rangle(d \wedge \phi') \\
\mathit{root}' & = & \mathit{root} \\
\mathit{first}' & = & q_{\mathit{first}}.
\end{array}$$

Note that this translation goes to the Palm language generated from the programs $\downarrow_1$ and $\downarrow_2$. Then χ is satisfiable on a tree in which γ is true in every node iff $d \wedge \chi'$ is satisfiable on a binary branching tree in which $d \to \gamma'$ and $[\downarrow_1]q_{\mathit{first}} \wedge [\downarrow_2]\neg q_{\mathit{first}} \wedge (\mathit{root} \to q_{\mathit{first}})$ is true everywhere. This is shown using the main idea from the reduction from $S\omega S$ to $S2S$ explained in Weyer (2002). Whence we have that

$$\gamma \models \chi \Leftrightarrow d \to \gamma' \wedge [\downarrow_1]q_{\mathit{first}} \wedge [\downarrow_2]\neg q_{\mathit{first}} \wedge (\mathit{root} \to q_{\mathit{first}}) \models d \to \chi'.$$

Now we can use the first reduction again to reduce this problem to the consequence problem of the language with just the modalities $\langle\downarrow_1\rangle$ and $\langle\downarrow_2\rangle$, interpreted on binary trees.

Chaining these reductions together, we obtain the reduction stated in the lemma.

Chapter 4

Spatial models for language: linguistic structure from an external perspective

LUIS D. CASILLAS MARTÍNEZ*

* Linguistics, Stanford University; casillas@stanford.edu

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

Chapter 5

Global Index Grammars and Descriptive Power

JOSÉ M. CASTAÑO*

ABSTRACT. We review the properties of Global Index Grammars (GIGs), a grammar formalism that uses a stack of indices associated with productions and has restricted context-sensitive power. We show how the *control* of the derivation is performed and how this impacts in the descriptive power of this formalism both in the string languages and the structural descriptions that GIGs can generate.

5.1 Introduction

The notion of *mild context-sensitivity* was introduced in Joshi (1985) as a possible model to express the required properties of formalisms that might describe Natural Language (NL) phenomena. It requires four properties:[1] a) *constant growth* property (or the stronger *semilinearity* property); b) polynomial parsability; c) *limited cross-serial* dependencies, i.e. some limited context-sensitivity d) proper inclusion of context free languages. The canonical NL problems which exceed context free power are: *multiple agreements*, *reduplication*, *crossing dependencies*.[2]

Many formalisms have been proposed to extend the power of context-free grammars using *control* devices, where the control device is a context-free grammar (see Dassow et al. (1997) regarding control languages). The appeal of this approach is that many of the attractive properties of context-free languages may be inherited (e.g. polynomial parsability, semilinearity, closure properties). Those models can be generalized such that additional *control* levels[3] can be added. They form hierarchies of levels of languages, where a language of level k properly includes a language of level $k-1$. For example in Weir (1992),

*Computer Science, Brandeis University, Waltham, MA, U.S.A.; jcastano@cs.brandeis.edu.

[1] See for example, Joshi et al. (1991), Weir (1988).

[2] However other phenomena (e.g. *scrambling*, Georgian Case and Chinese numbers) might be considered to be beyond certain *mildly context-sensitive* formalisms.

[3] The corresponding automaton models use embedded or an additional constrained stack. In such case, the generalization and hierarchy of levels is obtained using additional levels of embeddedness, or additional stacks (cf. Weir (1988), Cherubini et al. (1996)).

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

Mildly Context-sensitive Languages (MCSLs) are characterized by such a geometric hierarchy of control grammar levels (see also, Khabbaz (1974), Seki et al. (1991), Cherubini et al. (1996)). Those generalizations provide more expressive power but at a computational cost: the complexity of the recognition problem is dependent on the language level: for a MCSL level-k it is in $O(n^{3\cdot 2^{k-1}})$.

In Castaño (2003), we introduced Global Index Grammars (GIGs) and the corresponding languages, GILs. We presented a Chomsky-Schützenberger representation theorem for GILs. We showed that GIGs have enough descriptive power to capture the three phenomena mentioned above (*reduplication*, *multiple agreements*, and *crossed agreements*) in their generalized forms. GILs include such languages as $\{ww^+ \mid w \in \Sigma^*\}$, $\{a^n b^m (c^n d^m)^+ \mid n\, m \geq 1\}$ and $\{\, a^n (b^n c^n)^+ \mid n \geq 1\}$ which are beyond the power of Tree Adjoining Languages and beyond the power of any level-k control language. Recognition of the language generated by a GIG is in **bounded** polynomial time: $O(n^6)$, however for bounded state grammars with unambiguous indexing it is $O(n)$.

The equivalent model of automata was presented in Castaño (2003b). Also an algorithm to construct an LR parsing table for GILs was presented there. The automaton model and the grammar can be used to prove that the family of GILs is an Abstract Family of Languages using the same techniques to prove it for CFLs (cf. Castaño (2003b)). GILs have also the semilinear property, a proof can be easily built following the proof presented in Harju et al. (2001) for counter automata. Therefore GILs have at least three of the four properties required for Mildly context sensitivity: a) semi-linearity b) polynomial parsability c) proper inclusion of context free languages. The fourth property, *limited cross-serial dependencies* does not hold of GILs given they contain the MIX (or Bach) language.

The goal of this paper is to show how the properties of GILs are related to the peculiarities of the *control* device that regulates the derivation. Though this mechanism looks similar to the control device in Linear Indexed Grammars (LIGs, cf. Gazdar (1988)), its behavior differs relative to the trees generated by both formalisms.

GIGs offer additional descriptive power as compared to LIGs (and weakly equivalent formalisms) regarding the canonical NL problems mentioned above, and the same computational cost in terms of asymptotic complexity. They also offer additional descriptive power in terms of the structural descriptions they can generate for the same set of string languages, being able to produce *dependent paths*.[4] However those dependent paths are not obtained by encoding the dependency in the path itself.

This paper is organized as follows: Section 2 reviews Global Index Grammars and their properties, we give examples of its weak descriptive power and we discuss how *control* of the derivation is performed in GIGs. Section 3 discusses the strong descriptive power of GIGs.

[4] For the notion of dependent paths see for instance Vijay-Shanker et al. (1987) or Joshi (2000).

5.2 Global Index Grammars

5.2.1 Linear Indexed Grammars

Indexed grammars (IGs, Aho (1968)), and Linear Index Grammars, (LIGs; LILs) Gazdar (1988), have the capability to associate stacks of indices with symbols in the grammar rules. IGs are not semilinear. LIGs are Indexed Grammars with an additional constraint in the form of the productions: the stack of indices can be "transmitted" only to one non-terminal. As a consequence they are semilinear and belong to the class of MCSGs.

A **Linear Indexed Grammar** is a 5-tuple (V,T,I,P,S), where V is the set of variables, T the set of terminals, I the set of *indices*, S in V is the start symbol, and P is a finite set of productions of the form, where $A,B \in V$, $\alpha,\gamma \in (V \cup T)^*$, $i \in I$:

a. $A[..] \rightarrow \alpha\, B[..]\, \gamma$ b. $A[i..] \rightarrow \alpha\, B[..]\, \gamma$ c. $A[..] \rightarrow \alpha B[i..]\, \gamma$

Example 1. $L(G_{wcw}) = \{wcw \mid w \in \{a,b\}^*\}$,
$G_{ww} = (\{S,R\},\{a,b\},\{i,j\},S,P)$ *and P is:*

$1.S[..] \rightarrow aS[i..]$ $2.S[..] \rightarrow bS[j..]$ $3.S[..] \rightarrow cR[..]$ $4.R[i..] \rightarrow R[..]a$
$5.R[j..] \rightarrow R[..]b$ $5.\ R[] \rightarrow \varepsilon$

5.2.2 Global Indexed Grammars

GIGs use the stack of indices as a global control structure. This formalism provides a global but restricted context that can be updated at any local point in the derivation. GIGs are a kind of *regulated rewriting* mechanism (cf. Dassow and Păun (1989)) with global context and history of the derivation (or ordered derivation) as the main characteristics of its regulating device. The introduction of indices in the derivation is restricted to productions that are in Greibach normal form (i.e. in which the right hand side starts with a terminal). An additional constraint that is imposed on GIGs is strict leftmost derivation whenever indices are introduced or removed in the derivation.

Definition 1. *A GIG is a 6-tuple $G = (N,T,I,S,\#,P)$ where N,T,I are finite pairwise disjoint sets and 1) N are nonterminals 2) T are terminals 3) I a set of stack indices 4) $S \in N$ is the start symbol 5) # is the start stack symbol (not in I,N,T) and 6) P is a finite set of productions, having the following form:*

a.1 $A \underset{\varepsilon}{\rightarrow} \alpha$	*(epsilon rules)*	*or the equivalent*	$A \rightarrow \alpha$
a.2 $A \underset{[y]}{\rightarrow} \alpha$	*(epsilon with constraints)*	*or in LIG format:*	$[y..]A \rightarrow [y..]\alpha$
b. $A \underset{x}{\rightarrow} a\,\beta$	*(push)*		$[..]A \rightarrow [x..]a\,\beta$
c. $A \underset{\bar{x}}{\rightarrow} \alpha$	*(pop)*		$[x..]A \rightarrow [..]\alpha$

Note the difference between *push* (type b) and *pop* rules (type c): *push* rules require the right-hand side of the rule to contain a terminal in the first position.

Pop rules do not require a terminal at all. That constraint on *push* rules is a crucial property of GIGs. Derivations in a GIG are similar to those in a CFG except that it is possible to modify a string of indices. We define the *derives* relation $\Rightarrow$ on *sentential forms*, which are strings in $I^*\#(N \cup T)^*$ as follows. Let β and γ be in $(N \cup T)^*$, δ be in I^*, x in I, w be in T^* and X_i in $(N \cup T)$.

1. If $A \underset{\mu}{\rightarrow} X_1 \ldots X_n$ is a production of type (a.) (i.e. $\mu = \varepsilon$ or $\mu = [x], x \in I$) then:
   $\delta\#\beta A\gamma \underset{\mu}{\Rightarrow} \delta\#\beta X_1 \ldots X_n\gamma$ or $x\delta\#\beta A\gamma \underset{\mu}{\Rightarrow} x\delta\#\beta X_1 \ldots X_n\gamma$
2. If $A \underset{\mu}{\rightarrow} aX_1 \ldots X_n$ is a production of type (b.) or *push*: $\mu = x, x \in I$, then:
   $\delta\#wA\gamma \underset{\mu}{\Rightarrow} x\delta\#waX_1 \ldots X_n\gamma$
3. If $A \underset{\mu}{\rightarrow} X_1 \ldots X_n$ is a production of type (c.) or *pop* : $\mu = \overline{x}, x \in I$, then:
   $x\delta\#wA\gamma \underset{\mu}{\Rightarrow} \delta\#wX_1 \ldots\ldots X_n\gamma$

The reflexive and transitive closure of $\Rightarrow$ is denoted, as usual by $\overset{*}{\Rightarrow}$. We define the language of a GIG, G, $L(G)$ to be: $\{w | \#S \overset{*}{\Rightarrow} \#w$ and w is in $T^*\}$

The main difference between IGs, LIGs and GIGs, corresponds to the interpretation of the *derives* relation relative to the behavior of the stack of indices. In IGs the stacks of indices are distributed over the non-terminals of the right-hand side of the rule. In this way the same *control* words can be associated with multiple paths. This allows dependent paths. In LIGs, indices are associated with only one non-terminal at right-hand side of the rule. Thus there is only one stack affected at each derivation step, with the consequence that LILs are semi-linear. GIGs share this *uniqueness* of the stack with LIGs: there is only one stack to be considered per derivation. Unlike LIGs and IGs, the stack of indices is independent of non-terminals in the GIG case. *Push* rules (type b) are constrained to start the right-hand side with a terminal as specified in (6.b) in the GIG definition. The *derives* definition requires a *leftmost* derivation for those productions (*push* and *pop* rules) that affect the stack of indices.

The following example shows that GILs contain a language not contained in LILs, nor in the family of MCSLs. This language is relevant for modeling coordination in Natural Language as observed, for example, in Gazdar (1988).

Example 2 (Multiple Copies). $L(G_{wwn}) = \{ww^+ \mid w \in \{a,b\}^*\}$
$G_{wwn} = (\{S,R,A,B,C,L\},\{a,b\},\{i,j\},S,\#,P)$ and where P is:
$S \rightarrow AS \mid BS \mid C \quad C \rightarrow RC \mid L \quad R \underset{\overline{i}}{\rightarrow} RA \quad R \underset{\overline{j}}{\rightarrow} RB \quad R \underset{[\#]}{\rightarrow} \varepsilon$
$A \underset{i}{\rightarrow} a \quad B \underset{j}{\rightarrow} b \quad L \underset{\overline{i}}{\rightarrow} La \mid a \quad L \underset{\overline{j}}{\rightarrow} Lb \mid b$
The derivation of *ababab*:

$\#S \Rightarrow \#AS \Rightarrow i\#aS \Rightarrow i\#aBS \Rightarrow ji\#abS \Rightarrow ji\#abC \Rightarrow ji\#abRC \Rightarrow i\#abRBC \Rightarrow \#abRABC \Rightarrow$
$\#abABC \Rightarrow i\#abaBC \Rightarrow ji\#ababC \Rightarrow ji\#ababL \Rightarrow i\#ababLb \Rightarrow \#ababab$

The next example shows the MIX (or Bach) language. Gazdar (1988) conjectured the MIX language is not an IL. GILs are semilinear, therefore ILs and GILs are incomparable under set inclusion.

Example 3 (MIX language). .

$L(G_{mix}) = \{w | w \in \{a,b,c\}^* \text{ and } |a|_w = |b|_w = |c|_w \geq 1\}$
$G_{mix} = (\{S,D,F,L\},\{a,b,c\},\{i,j,k,l,m,n\},S,\#,P)$ where P is:
$S \to FS \mid DS \mid LS \mid \varepsilon \quad F \underset{i}{\to} c \quad F \underset{j}{\to} b \quad F \underset{k}{\to} a$
$D \underset{i}{\to} aSb \mid bSa \quad D \underset{j}{\to} aSc \mid cSa \quad D \underset{k}{\to} bSc \mid cSb \quad D \underset{l}{\to} aSb \mid bSa$
$D \underset{m}{\to} aSc \mid cSa \quad D \underset{n}{\to} bSc \mid cSb \quad L \underset{l}{\to} c \quad L \underset{m}{\to} b \quad L \underset{n}{\to} a$

The following language cannot be generated by LIGs. It is mentioned in Vijay-Shanker et al. (1987) in relation to the definition of composition in Steedman (1985) Categorial Grammars, which permits composition of functions with unbounded number of arguments and generates tree sets with dependent paths.

Example 4 (Dependent branches). .

$L(G_{sum}) = \{\, a^n b^m c^m d^l e^l f^n \mid n = m + l \geq 1\}$,
$G_{sum} = (\{S,R,F,L\},\{a,b,c,d,e,f\},\{i\},S,\#,P)$ where P is:

$$S \underset{i}{\to} aSf \mid R \quad R \to F\,L \mid F \mid L \quad F \underset{i}{\to} b\,F\,c \mid b\,c \quad L \underset{i}{\to} d\,L\,e \mid d\,e$$

The derivation of $aabcdeff$:
$\#S \Rightarrow i\#aSf \Rightarrow ii\#aaSff \Rightarrow ii\#aaRff \Rightarrow ii\#aaFLff \Rightarrow i\#aabcLff \Rightarrow \#aabcdeff$

5.2.3 Control of the derivation in LIGs and GIGs

Every LIL can be characterized by a language $L(G,C)$, where G is a labelled grammar (cf. Weir (1992)), $G = (N,T,L,S,P)$, and C is a control set defined by a CFG (a Dyck language):
$\{a_1 ... a_n | < S,\varepsilon > \overset{*}{\Rightarrow} < a_1,w_1 > ... < a_n,w_n >, a_i \in T \cup \{\varepsilon\}, w_1,...,w_n \in C\}$.

In other words, *control* strings w_i are not necessarily *connected* to each other. Those control strings are encoded in the derivation of each *spine* as depicted in figure 5.1 at the left, but every substring encoded in a *spine* has to belong to the control set language. Those control words describe the properties of a path (a *spine*) in the tree generated by the grammar G, and every possible *spine* is independent.

We defined the language of a GIG G, $L(G)$ to be: $\{w | \#S \overset{*}{\Rightarrow} \#w$ and w is in $T^*\}$. We can obtain an explicit control language modifying the derivation as follows.

First modify the derives relations such that *pop* productions rewrite the complement of the index: $x\delta\#wA\gamma \underset{\mu}{\Rightarrow} \bar{x}x\delta\#wX_1.....X_n\gamma$

Then, define the language of a GIG G, to be the control language $L(G,C)$: $\{w|\#S \overset{*}{\Rightarrow} \delta\#w$ and w is in T^*, δ is in $C\}$, and C is defined to be the Dyck language over the alphabet $I \cup \bar{I}$ (the set of stack indices and their complements).

It is easy to see that no control substring obtained in a derivation subtree is necessarily in the control language, as is depicted in the figure 5.1 at the right. In other words, the control of the derivation can be distributed over different paths, however those paths are connected transversally by the leftmost derivation order.

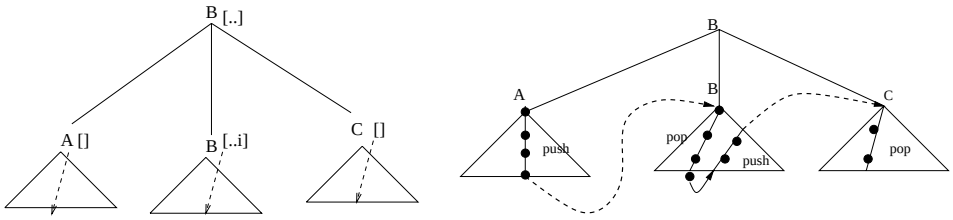


Figure 5.1: LIGs: multiple spines (left) and GIGs: leftmost derivation

5.3 GIGs and structural descriptions

Gazdar (1988) introduces Linear Indexed Grammars and discusses their applicability to Natural Language problems. This discussion is addressed not in terms of weak generative capacity but in terms of strong-generative capacity. Similar approaches are also presented in Vijay-Shanker et al. (1987) and Joshi (2000) (see Miller (1999) concerning weak and strong generative capacity). In this section we review some of the abstract configurations that are argued for in Gazdar (1988).

5.3.1 The palindrome language

CFGs can recognize the language $\{ww^R|w \in \Sigma^*\}$ but they cannot generate the structural description depicted in figure 5.2 (we follow Gazdar's notation: the leftmost element within the brackets corresponds to the top of the stack):

Gazdar suggests that the configuration at the left would be necessary to represent Scandinavian unbounded dependencies. Such an structure can be obtained using a GIG (and of course a LIG). But the exact mirror image of that structure, (i.e. the structure at the right) cannot be generated by a GIG because it would require *push* productions with a non terminal in the first position of the right-hand side.

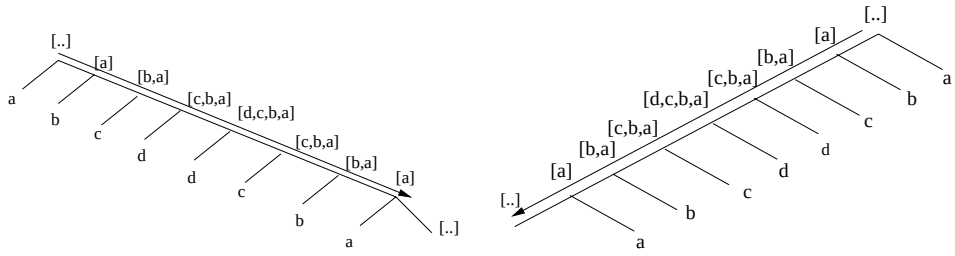


Figure 5.2: Non context-free structural descriptions for the language $\{ww^R | w \in \Sigma\}$ Gazdar (1988)

However GIGs generate a similar structural description as depicted in figure 5.3 at the left. In such structure the dependencies are introduced in the leftmost derivation order. The English adjective constructions that Gazdar argues can motivate the LIG derivation, are generated by the following GIG grammar. The corresponding structural description is shown in figure 5.3.

Example 5 (Comparative Construction). .

$G_{adj} = (\{AP, NP, \bar{A}, A\}, \{a, b, c\}, \{i, j\}, AP, \#, P)$ *where P is:*

$$AP \to AP\ NP \qquad AP \to \bar{A} \qquad \bar{A} \to \bar{A}\ A$$
$$A \underset{i}{\to} a \qquad A \underset{j}{\to} b \qquad A \underset{k}{\to} c \qquad NP \underset{\bar{i}}{\to} a\ NP$$
$$NP \underset{\bar{j}}{\to} b\ NP \qquad NP \underset{\bar{k}}{\to} c\ NP$$

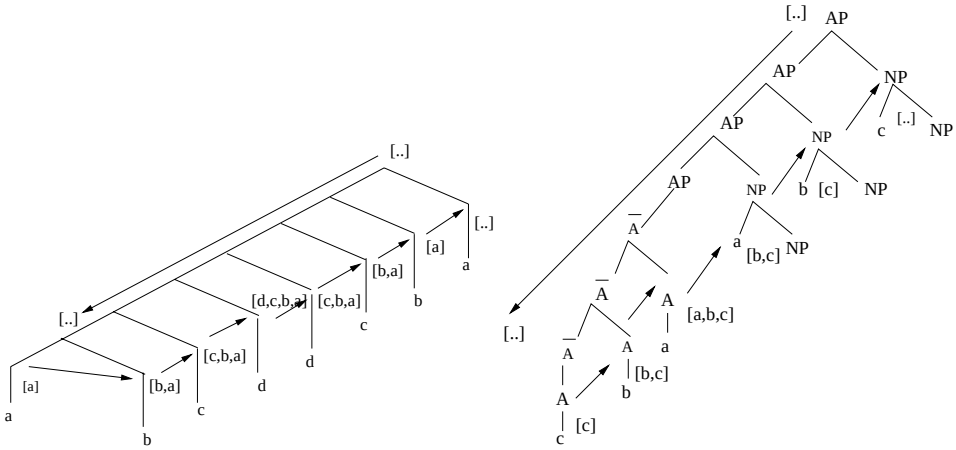


Figure 5.3: GIG structural descriptions for the language ww^R

It should be noted that the operations on indices are reversed as compared to the LIG case shown in right figure of 5.2. On the other hand, it can be noticed also that the introduction of indices is dependent on the presence of lexical information and its *transmission* is not carried through a top-down *spine*, as in the LIG case. The arrows show the leftmost derivation order that is required by the operations on the stack.

5.3.2 The Copy Language

Gazdar presents the two possible LIG structural descriptions for the copy language depicted in figure 5.4.

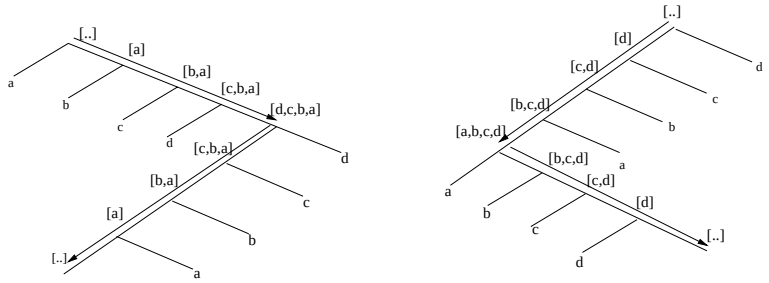


Figure 5.4: LIG structural descriptions of the copy language Gazdar (1988)

The structural description at the left in figure 5.4 can be obtained using GIGs, but not the one at the right. However Gazdar argues that the tree structure shown in figure 5.5 at the left, could be more appropriate for some Natural Language phenomenon that might be modeled with a copy language. Such structure cannot be generated by a LIG, but can be generated by an IG.

GIGs cannot produce this structural description either, but they can generate the one presented in the figure 5.5 at the right, where the arrows depict the left-most derivation order. GIGs can also produce similar structural descriptions for the language of multiple copies (the language $\{ww^{+} \mid w \in \Sigma^{*}\}$ as shown in figure 5.6, corresponding to a grammar like the one shown in example 2.

5.3.3 Multiple dependencies

There is no discussion of the applicability of multiple dependency structures in Gazdar (1988). The relevant structures that can be produced by a LIG are depicted in figures 5.7 and 5.8 (left). GIGs can generate the same structures as in 5.7 and the somewhat equivalent in 5.8 at the right.

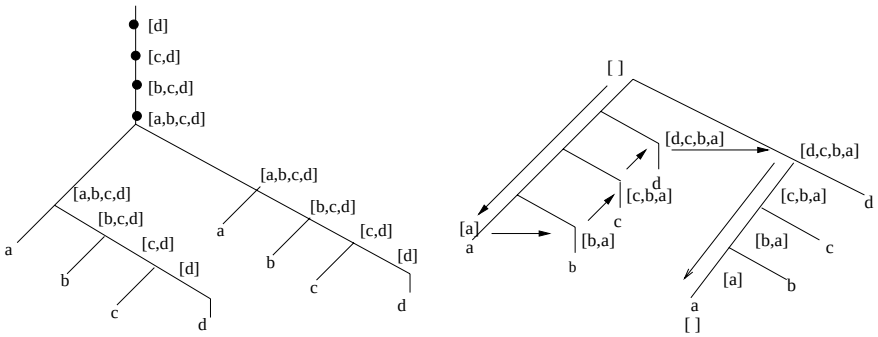


Figure 5.5: An IG structural description of the copy language Gazdar (1988) (left) and a GIG structural description (right)

Also, GIGs can produce other structures that cannot be produced by a LIG, as we show in figure 5.9, including those corresponding to G_{sum} discussed above.

5.3.4 Conclusions

We have reviewed GIGs and GILs and their most important properties. We showed that the descriptive power of GIGs is beyond CFGs. CFLs are properly included in GILs by definition. We showed also that GIGs include some languages that are not in the LIL/TAL family nor in the MCSLs as characterized in Weir (1992). The similarity between GIGs and LIGs, strongly suggests that LILs might be included in GILs. We presented a comparison of the structural descriptions that LIGs and GIGs can be generate. We have shown that GIGs generate structural descriptions for the copy and multiple dependency languages which can not be generated by LIGs. Finally, we have shown also that the extra power that characterizes GIGs, corresponds to the ability of GIGs to generate dependent paths without *copying* the stack but *distributing* the control in different paths.

Acknowledgments: Thanks to J. Pustejovsky for his continuous support and encouragement on this project. Many thanks also to the anonymous reviewers who provided many helpful comments. This work was partially supported by NLM Grant R01 LM06649-02.

Bibliography

Aho, A. V. (1968). Indexed grammars - an extension of context-free grammars. *Journal of the Association for Computing Machinery*, **15**(4):647–671.

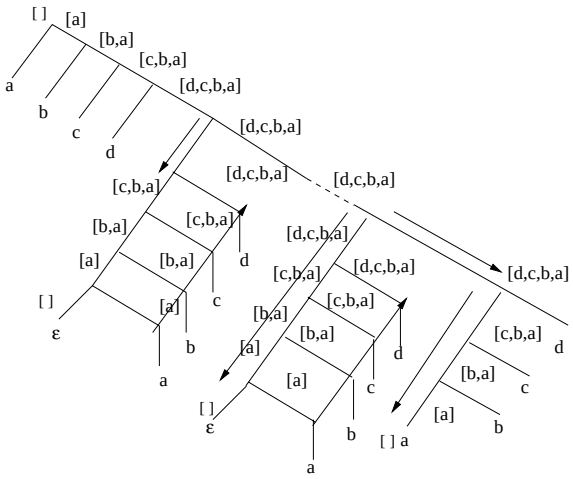


Figure 5.6: A GIG structural description for the multiple copy language

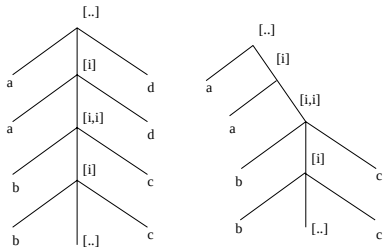


Figure 5.7: LIG and GIG structural descriptions of 3 and 4 dependencies

Castaño, J. (2003). GIGs: Restricted context-sensitive descriptive power in bounded polynomial-time. In *Proc. of Cicling 2003, Mexico City, February 16-22*.

Castaño, J. (2003b). LR Parsing for Global Index Languages (GILs). In *In Proceeding of CIAA 2003, Santa Barbara,CA.*

Cherubini, A., L. Breveglieri, C. Citrini, and S. Reghizzi (1996). Multipushdown languages and grammars. *International Journal of Foundations of Computer Science*, **7(3)**:253–292.

Dassow, J. and G. Păun (1989). *Regulated Rewriting in Formal Language Theory*. Springer, Berlin, Heidelberg, New York.

Dassow, J., G. Păun, and A. Salomaa (1997). Grammars with controlled deriva-

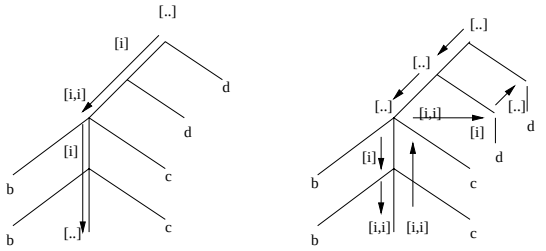


Figure 5.8: A LIG (left) and a GIG (right) structural descriptions of 3 dependencies

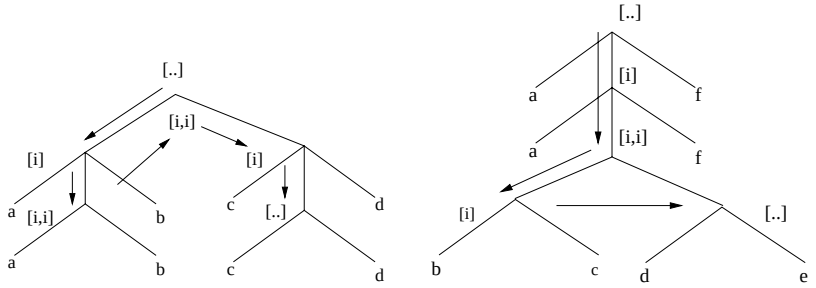


Figure 5.9: A GIG structural descriptions of 4 dependencies (left) and the example of grammar G_{sum} (right)

tions. In G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages, Vol. 2*. Springer, Berlin,.

Gazdar, G. (1988). Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, eds., *Natural Language Parsing and Linguistic Theories*, pp. 69–94. D. Reidel, Dordrecht.

Harju, T., O. Ibarra, J. Karhumäki, and A. Salomaa (2001). Decision questions concerning semilinearity, morphisms and commutation of languages. In *LNCS 2076*, p. 579ff. Springer.

Joshi, A. (1985). Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural description? In D. Dowty, L. Karttunen, and A. Zwicky, eds., *Natural language processing: psycholinguistic, computational and theoretical perspectives*, pp. 206–250. Chicago University Press, New York.

Joshi, A. (2000). Relationship between strong and weak generative power of formal systems. In *Proceedings of the Fifth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pp. 107–114. Paris, France.

Joshi, A., K. Vijay-Shanker, and D. Weir (1991). The convergence of mildly context-sensitive grammatical formalisms. In P. Sells, S. Shieber, and T. Wasow, eds., *Foundational issues in natural language processing*, pp. 31–81. MIT Press, Cambridge, MA.

Khabbaz, N. A. (1974). A geometric hierarchy of languages. *Journal of Computer and System Sciences*, **8**(2):142–157.

Miller, P. (1999). *Strong Generative Capacity*. CSLI Publications, Stanford University, Stanford CA, USA.

Seki, H., T. Matsumura, M. Fujii, and T. Kasami (1991). On multiple context-free grammars. *Theoretical Computer. Science*, pp. 191–229.

Steedman, M. (1985). Dependency and coordination in the grammar of dutch and english. *Language*, pp. 523–568.

Vijay-Shanker, K., D. J. Weir, and A. K. Joshi (1987). Characterizing structural descriptions produced by various grammatical formalisms. In *Proc. of the 25th ACL*, pp. 104–111. Stanford, CA.

Weir, D. (1988). *Characterizing mildly context-sensitive grammar formalisms*. Ph.D. thesis, University of Pennsylvania.

Weir, D. J. (1992). A geometric hierarchy beyond context-free languages. *Theoretical Computer Science*, **104**(2):235–261.

Chapter 6

Bounded and Ordered Satisfiability: Connecting Recognition with Lambek-style Calculi to Classical Satisfiability Testing

MICHAIL FLOURIS*, LAP CHI LAU*, TSUYOSHI MORIOKA*, PERIKLIS A. PAPAKONSTANTINOU*, GERALD PENN*

6.1 Introduction

It is well known that the Lambek Grammars are weakly equivalent to the Context-Free Grammars (CFGs, Pentus 1993, 1997), and that testing string membership with a CFG is in *P* (Earley 1970). Nevertheless, Pentus (2003) has recently proven that sequent derivability in the Lambek Calculus with product is NP-complete. The complexity of the corresponding problem for the product-free fragment remains unknown. This fragment is significant, given the at best limited apparent motivation for products in linguistic applications of the calculus. In this paper, when we mention the Lambek Calculus (LC) or Lambek Grammars (LG), we are referring to the product-free fragment.

Pentus (1997) has presented an algorithm that transforms a product-free Lambek Grammar to a weakly equivalent CFG, but the transformed grammar is exponentially larger in the worst case. Since the grammar is considered a part of the instance for the decision problem of string membership, this does not resolve the open complexity problem for the product-free calculus.

This paper studies the connection between the LG string membership problem and the SAT problem, which was the first problem shown to be NP-complete (Cook 1971). Much of the previous work on parsing with Lambek grammars has derived

*Dept. of Computer Science, University of Toronto; email correspondence to gpenn@cs.toronto.edu.

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

its inspiration from recognition algorithms for rewriting systems (Hepple 1992), string algebras (Morrill 1996) or graph theory (Moot and Puite 1999; Penn 2002), but fundamentally, LC is a logical framework, like the classical propositional logic upon which SAT is based. The crucial difference is the sensitivity to resources and order that LC incorporates. What we argue here is: **(1)** that a sense of order can be imposed on classical SAT using the polarity that propositional variables already possess (unlike LC), **(2)** that the corresponding *ordered* SAT problem is still NP-complete, **(3)** that this new version of SAT leads to a new and simpler proof of the NP-completeness of the product-free Lambek Calculus with permutation (LP) by an implementation of "locks" and "keys" somewhat reminiscent of the proposed modal extensions of categorial logics (Kurtonina and Moortgat 1996), **(4)** that the problem can be further restricted *bounded-distance* SAT in order to fit into LC, but **(5)** that bounded-distance SAT can be solved in polynomial time. In addition, **(6)** we also prove the first non-trivial hardness result for LC that we are aware of, namely that it is LOGCFL-hard. LOGCFL consists of all languages log-space reducible to a context-free language. We shall use LC and LP to refer both to the respective calculi and to the respective decision problems that determine membership in the set of encodings of pairs (G,x), where G is a Lambek Grammar over an alphabet Σ, and $x \in \Sigma^*$ is a string.

6.2 An NP-complete variation of SAT

Our ordered variation of 3-SAT can be stated as follows:
NFPO-SAT
INSTANCE: Let U be a set of variables and $S = \langle C_1, C_2, \ldots C_n \rangle$ a sequence of clauses, where $|C_i| \leq 3$, such that **(1)** each variable $x \in U$ occurs at most once as a negative literal, **(2)** if $\neg x \in C_i$ there exist no $j < i$ where $x \in C_j$, and **(3)** every clause which contains a negative literal cannot contain any positive literal. Let a formula $\Phi = \bigcup_{i=1\ldots n} C_i$.
QUESTION: Is Φ satisfiable ?

NFPO stands for "Negative First Positive the Others", that is: (a) the clauses are ordered such that the first occurrence of a variable is either negative or positive and all the subsequent occurrences are positive, and (b) every clause contains either all positive or all negative variables. Hence, in this variation we can refer to *positive* and *negative* clauses, with the obvious meaning.

Theorem 6.2.1. NFPO-SAT *is NP-complete.*

Proof: The problem is in *NP* for the same reason that SAT is in NP. We reduce 3-SAT to NFPO-SAT:
Let $\Phi = C_i, \ldots C_n$ be a 3-SAT instance. For each variable x:

1. Let x occur negatively in one or more clauses.
2. Introduce y such that:

$$\Phi' = \Phi \wedge (y \leftrightarrow \neg x), \text{ i.e.,}$$
$$\Phi' = \Phi \wedge (\neg y \vee \neg x) \wedge (y \vee x)$$

3. Rename the negative occurrences of x by y.
4. Iteratively change each variable, using Φ' instead of Φ, apart from the newly introduced variable.

Finally, order the clauses of Φ' by placing all clauses with negative literals before all clauses with only positive literals.

Given a fixed truth assignment, τ, for which $\tau(\Phi) = T$, let τ' be an extended truth assignment for Φ' such that $\tau'(y) = \neg\tau(x)$. Then $\tau'(\Phi') = T$. Hence, Φ' is satisfiable iff Φ is satisfiable. It is also obvious that the reduction works in quadratic time w.r.t. the input length. Note that the reduction works also in logarithmic space. □

It will also be useful for us to consider a version of NFPO-SAT with an additional condition:
BD-NFPO-SAT (Bounded-Distance NFPO-SAT)
INSTANCE: Let U be a set of n variables and $S = \langle C_1, C_2, \ldots C_m \rangle$ a sequence of clauses where $|C_i| \leq 3$ such that **(1)** each variable $x \in U$ occurs at most once as a negative literal, **(2)** if $\neg x \in C_i$ there exist no $j < i$ where $x \in C_j$, **(3)** every clause which contains a negative literal cannot contain any positive literal, and **(4)** there exists $k = \lceil \log n \rceil$ such that, for each variable $x \in U$, if i is minimum with respect to the occurrence of x in C_i, and j is the maximum j such that $x \in C_j$, then $j - i \leq k$. Let a formula $\Phi = \bigcup_{i=1\ldots n} C_i$.
QUESTION: Is Φ satisfiable ?

This version is not NP-complete. In fact, we can prove that the BD-SAT $\in NL \subseteq P$, where NL stands for the class of languages decided by nondeterministic log-space Turing Machines.
Non-deterministic log space algorithm for BD-NFPO-SAT
(i) Guess values for the variables of the k clauses.
(ii) If the subformula is not satisfied reject
(iii) Keep the k last assignments and make a guess for the next clause
(iv) If the clause cannot be satisfied, or if consistency is lost with the previous assignment reject, otherwise add the new guessed values and repeat (iii) (intuitively: slide the k window by one clause to the right).

Bounded distance satisfiability problems are also of independent interest. We state the following theorem without any proof. In Section 6.6, we prove a stronger result for the hardness of LC.

Theorem 6.2.2. BD-NFPO-SAT *is complete for NL.*

The reduction of BD-NFPO-SAT to LC trivially implies:

Corollary 6.2.3. BD-NFPO-SAT *is hard for NL.*

6.3 Reducing SAT to LP

We will follow the Natural Deduction presentation of the Lambek Calculus. We will use the following deduction quite often, which is valid with or without the rule of permutation:

Lemma 6.3.1. *Let A, B and C categories of the Lambek Calculus. The following deductions can be derived using only elimination and introduction rules:*

$$\frac{A/B \quad B/C}{A/C}, \quad \frac{A/A \quad A/A}{A/A}, \quad \frac{A/A \quad A/A \ldots A/A}{A/A} \tag{6.3.1}$$

That LP is NP-complete is already known, both as a corollary of a more general result for multiplicative Linear Logic (Kanovitch 1991, 1992; Lincoln et al. 1990), or directly (Doerre 1996; Florencio 2002). In addition, not all *LP* grammars are weakly equivalent to CFGs. We sketch two different proofs for the NP-completeness of LP, both by reducing NFPO-SAT to LP. The order of presentation of the proofs is such that we successively make more use of the ordering constraints imposed by NFPO-SAT.

Proof 1. It is well known (Lincoln et al. 1990) that $LP \in NP$. Assume that we have a formula Φ in NFPO-CNF form. For each variable x_i occurring positively in Φ we introduce a basic category X_i, and for each variable occurring negatively, we introduce a new category $\overline{X_i}$. We also have a special basic category A. Assume that we have m clauses $C_i, i = 1, \ldots, m$ and n variables $x_i, i = 1, \ldots, n$. We construct the following string $w = c_1 c_2 \ldots c_m x_1 x_1' x_2 x_2' \ldots x_n x_n'$, where c_i, x_i and x_i' are distinct symbols of an alphabet Σ. We construct the mapping f (Lexicon) for each symbol of the alphabet as follows: for each variable x_i occurring positively in the clause C_j we add to the set $f(c_j)$ the elements $((A/A)/X_i)/X_i$ and X_i/X_i, and for each x_i occurring negatively in C_j we add the elements $((A/A)/\overline{X_i})/\overline{X_i}$ and $\overline{X_i}/\overline{X_i}$. Also, for all $1 \leq i \leq n$, $f(x_i) = f(x_i') = \{X_i, \overline{X_i}\}$. This algorithm is trivially polynomial time.

The substring $w_1 = c_1 \ldots c_m$ of w, corresponds to the selection of a single literal from each clause that witnesses that clause's truth. In the first clause C_j for

which a variable x_i is selected, this corresponds to choosing for c_j the category $((A/A)/X_i)/X_i$, and every other selection for the same variable in a later clause corresponds to choosing X_i/X_i. For any k, if a variable is selected in k clauses then, by lemma 6.3.1, we can still derive two categories for this substring: $((A/A)/X_i)/X_i$ and (after some deductions) X_i/X_i which altogether result in $((A/A)/X_i)/X_i$.

The substring $w_2 = x_1x_1' \dots x_nx_n'$ enforces the consistency of the selected variables. If a variable x_i has been selected together with its negation $\neg x_i$ then after doing several deductions we have both $((A/A)/X_i)/X_i$ and $((A/A)/\overline{X_i})/\overline{X_i}$, but the x_ix_i' component of the substring w_2 can only deduce either $((A/A)/X_i)/X_i$ or (exclusively) $((A/A)/\overline{X_i})/\overline{X_i}$ to A/A. So to derive A/A, we can select either x_i or its negation but not both. Thus, the LP grammar constructed has A/A as its distinguished category. Hence, if there is a deduction to A/A the formula is satisfiable.

It is easy to see that if the formula is satisfied, then we can select the categories to derive A/A in a fairly simple way: choose the $((A/A)/X_i)/X_i$ or X_i/X_i if x_i is true and $((A/A)/\overline{X_i})/\overline{X_i}$ or $\overline{X_i}/\overline{X_i}$ if $\neg x_i$ is true from the corresponding clauses.

Remark 6.3.1. In this reduction, we did not take into account the constraints imposed by the NFPO-SAT and hence the same reduction holds for the unconstrained version of SAT.

Remark 6.3.2. Notice that we could have also ordered the literals in each clause, and replace each c_i by a substring $c_{i,1}c_{i,2}\dots c_{i,k}$ where k is the number of variables in C_i. Then we need extra basic categories $A_{i,l}$ because now each $c_{i,l}$ corresponds to a literal in clause C_i. If x is the first literal, we would have: $f(c_{i,1}) = \{A_{i,1}, X/X/A_{i,k}/A_{i,k-1}/\dots/A_{i,2},\ A/A/X/X/A_{i,k}/A_{i,k-1}/\dots/A_{i,2}\}$, and $f(c_{i,2}) = \{A_{i,2}, A_{i,1}\backslash X/X/A_{i,k}/A_{i,k-1}/\dots/A_{i,3}, A_{i,1}\backslash A/A/X/X/A_{i,k}/A_{i,k-1}/\dots/A_{i,3}\}$, and so on. From among the variables of each clause, one emerges as functor in this part of any successful derivation. This variable is the witness selected to attest to the clause's truth. The same procedure takes place to select one literal from every clause, and the rest of the proof can proceed as previously described.

In the next section, we present a proof that takes into account the sense of ordering inherent to NFPO-SAT. Note that there is no need to do this when reducing to LP, since LP does not have any ordering constraints. It will be useful, however, when it comes to considering LC.

6.4 Enforcing restrictions with locks

We now employ the familiar notion of *locks* and *keys*. The notion of *locks* will be especially useful in LC deductions, where permutation is missing. We introduce the idea only by an example here: $(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.

The notion of a lock prevents us from selecting both a variable and its negation. Thus, if $\neg x_1$ is selected from the first clause then it imposes a *lock*, L_1, which is a special basic category. In the next clause, the only literals that can *unlock* this lock should be every other variable apart from x_1. For this example and the corresponding string $c_{1,x_1}c_{1,x_2}c_{1,x_3}c_{2,\neg x_1}c_{2,x_2}c_{2,x_3}$ we have that

$$\begin{aligned} f(c_{1,x_1}) &= \{A_1,(A/A)/L_1/A_2/A_3\},\\ f(c_{1,x_2}) &= \{A_2,A_1\backslash(A/A)/A_3\},\\ f(c_{1,x_3}) &= \{A_3,A_2\backslash A_1\backslash(A/A)\},\\ f(c_{2,\neg x_1}) &= \{B_1,(A/A)/B_2/B_3\},\\ f(c_{2,x_2}) &= \{B_2,B_1\backslash L_1/B_3,B_1\backslash(A/A)/B_3\},\\ \text{and } f(c_{3,x_3}) &= \{B_3,B_2\backslash B_1\backslash L_1,B_2\backslash B_1\backslash(A/A)\}. \end{aligned}$$

Notice, that if we had another negative variable in the first clause then we could have easily added keys to the other two variables in the second clause. This task is performed by just adding new elements to the corresponding sets and thus it does not change the time needed to compute these sets to something more than polynomial. Furthermore, in NFPO-SAT, all negative literals occur first, so a c_i with locks comes before any c_j with matching keys. The difficult part is to see what happens if the two above clauses are far enough apart. This means that we have to propagate the locks correspondingly. Notice that we could have only one symbol in the string corresponding to each clause (instead of one symbol for each variable), as in our earlier proof. But now, we will combine the idea of locks and keys with our observation in Remark 6.3.2.

Proof 2. Assume that we have an instance of NFPO-SAT with n variables and m clauses. Construct

$$w = w_{11}w_{12}w_{13}w'_{11}w'_{12}w'_{13}\,w_{21}w_{22}w_{23}w'_{21}w'_{22}w'_{23}\,\ldots\,w_{m1}w_{m2}w_{m3}w'_{m1}w'_{m2}w'_{m3}.$$

Intuitively, w_{ik} and w'_{ik} correspond to the k-th literal of the i-th clause. We know that in NFPO-SAT every clause contains either only negative or only positive literals. In addition, we know that all the negative clauses precede the positive clauses in the sequence. Assume that C is a negative clause. Then each literal $\neg x \in C$ sets a lock $A/A/L_x/L_x/\ldots/L_x/L_x$ (L_x appears as many times as the positive occurrences of x). The keys for this variable occur in categories assigned to the other variables in the (positive) clauses where x occurs positively. For some positive $C = (x \vee y \vee z)$, then $f(w_x) = \{B_1, L_y/B_2/B_3, A/A/B_2/B_3\}$, $f(w'_x) = \{B'_1, L_z/B'_2/B'_3, A/A/B'_2/B'_3\}$, where B_i, B'_i are used as described in Remark 6.3.2. That is, literal x holds the keys for the other two literals y and z. $f(w_y)$, $f(w_y)'$, $f(w_z)$ and $f(w_z)'$ are constructed analogously.

6.5 Restricting to LC

This second proof is conceptually less dependent on permutation in the sense that it is only used to combine locks and keys. Also, observe that our constructions involve only first-order categories, where the recognition problem is known to be in P. The main task of adapting the previous reductions to LC (with no permutation) is that we need a way of propagating the locks with no permutation at all. If we have a tuple of clauses we can easily compute the sequence in which the literals appear. The problem is that when we put a lock in some clause we may need to change the order of the previously placed locks. So, we need a sufficient number of rewritings. But we do not know the exact number of locks previously placed, which in the worst case could be $\sum_i^n \binom{n}{i} = 2^n - 1$, where n is the number of variables with locks placed in a negative clause to the left.

BD-NFPO-SAT places exactly the bound we need to avoid an exponential explosion in this case. The resulting reduction, of course, says nothing about whether recognition in LC is NP-hard. Intuitively, NP-complete problems involve a significant amount of communication among their parts. For example, if we flip the value of a variable x then this has an effect to the whole formula. When we bound this communication in this way, then we fall in the complexity hierarchy from *NP*-complete to membership in *NL*. What is needed is a reduction that uses higher-order categories in order to avoid this.

6.5.1 Example of the LC-embedding of BD-NFPO-SAT

For simplicity, assume that the distance bound suffices to cover all of the following portion of a formula:

$$(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_5 \vee x_1 \vee x_2) \wedge (\neg x_6 \vee \neg x_7 \vee \neg x_8) \wedge (x_6 \vee x_1 \vee x_3) \wedge (x_1 \vee x_7 \vee x_8) \wedge \ldots$$

In the clause $(x \vee y \vee z)$ the literals y and z are called *neighbours* of x. Only variables in the first and the third clause can place locks. If some $\neg x$ has placed a lock, then with respect to every other occurrence of x except the last one, the neighbours of x should propagate the lock placed by x. In the clause with the last occurrence of x, the neighbours of x unlock the lock previously imposed by $\neg x$. Below, by "propagate" we refer to the fact that if a positive literal is selected to be true then it should rewrite all locks concerning all variables occurring negatively before itself with the following two constraints: (a) it does not propagate locks placed by its own negation, and (b) it does not propagate locks from variables with instances only before, i.e., to the left of, its clause.

A simple form of this reduction follows: recall from Remark 6.3.2 that we can have a single symbol corresponding to a clause. For the above formula we have the string $w_1w_2w_3w_4w_5$, where w_1, w_2, w_3, w_4, w_5 are all different symbols of an alphabet. The target category we want to deduce is A/A. We construct the lexicon as follows: $f(w_1) = \{A/A/L_1, A/A/L_2, A/A/L_3\}$, where $A/A/L_1, A/A/L_2, A/A/L_3$ correspond to the locks placed by $\neg x_1, \neg x_2, \neg x_3$ respectively. $f(w_2) = \{L_1/L_1, L_2, L_3/L_3\}$, because x_5 and x_1 can unlock x_2 (L_2) which occurs for the last time in the second clause, x_5 and x_2 can propagate the lock for x_1 (L_1/L_1), and x_5 can propagate the lock for x_3 (L_3/L_3). In the same fashion, we construct the rest of the lexicon, where $L_{i_1 i_2 \ldots i_k}$ denotes a composite lock, corresponding to the k locks placed by $x_{i_1}, \ldots, x_{i_k}$:

$$f(w_3) = \{L_1/L_{16}, L_2/L_{26}, L_3/L_{36}, A/A/L_6, L_1/L_{17}, L_2/L_{27}, L_3/L_{37}, A/A/L_7, L_1/L_{18}, L_2/L_{28}, L_3/L_{38}, A/A/L_8\},$$

$$f(w_4) = \{L_3, L_{36}/L_6, L_{37}/L_7, L_{38}/L_8, L_6, L_{16}/L_1, L_{26}/L_2, L_{36}/L_3, L_3, L_{36}/L_6, L_{37}/L_7, L_{38}/L_8\},$$

$$f(w_5) = \{L_7, L_8\}.$$

Notice that the above formula is satisfiable, e.g., by $x_1 = 1, x_5 = 1, x_6 = 0, x_3 = 1, x_7 = 1$, corresponding to $A/A/L_1\ L_1/L_1\ L_1/L_{16}\ L_{16}/L_1\ L_1$, from which A/A can be derived.

6.6 LOGCFL-Hardness of LC

In the previous sections we developed some machinery, based on ordered satisfiability, in order to show hardness results for LC and LP. A restriction of this machinery, in which the parameter of the longest distance between two appearances of a variable is bounded, exhausts its limits for a logarithm in the number of variables. The reason is that if the distance is more than a logarithm then the reduction is no longer polytime (or log-space). We also saw in Section 6.2 that LC is NL-hard.

In this section we use another approach, which proves a stronger hardness result for LC, namely that it is LOGCFL-hard. All of our reductions belong to L, which is the class of languages characterized by their decidability with deterministic logarithmic space Turing Machines. One characterization of LOGCFL is as the class of all languages log-space reducible to a Context-Free Language (CFL). It contains NL (Sudborough 1978) and is contained in P. Cook (1985) contains more information on LOGCFL. All told, we have the following containments relative to our problem of interest:

$$L \subseteq NL \subseteq LOGCFL \subseteq P \subseteq NP$$

None of these containments is known to be proper, although it is widely conjectured that every one of these containments is proper.

Theorem 6.6.1. *LC is hard for LOGCFL.*

Note that this is not a proof of LOGCFL-completeness. If we knew that LC belonged to LOGCFL, we would know that LC is in polytime. *Proof:* Fix an arbitrary language $A \in LOGCFL$. By the definition of LOGCFL, there exists a context-free language L and a log-space computable function f such that, for all x, $x \in A$ iff $f(x) \in L$. We prove that there exists a log-space computable function $g(x)$ such that $x \in A$ if and only if $g(x) \in LC$. We require the following well-known lemma:

Lemma 6.6.2. *If L is a CFL then $L\backslash\{\varepsilon\}$ is also context-free, where ε is the empty string.*

Since $L\backslash\varepsilon$ is context-free, there exists a CFG G in Greibach normal form such that $L\backslash\varepsilon = L(G)$. Two cases arise:

1. If $\varepsilon \in L$, then $g(x)$ is computed as follows. If $f(x) = \varepsilon$ then let $g(x)$ be some fixed accepting instance for LC. Otherwise, from G and $f(x)$, we construct as $g(x)$ an instance of LC with string $f(x)$ and a Lambek Grammar defined as follows: for every Greibach-normal rule, $N_0 \longrightarrow t\ N_1\ \ldots\ N_n$, where t is terminal and the N_i are non-terminals, add to the categories assigned to t in the Lambek Grammar the category $N_0/N_n/\ldots/N_1$.

2. $\varepsilon \notin L$. This case is handled similarly to the above except that, if $f(x) = \varepsilon$, then let $g(x)$ be some fixed non-accepting instance of LC.

□

Bibliography

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of 3rd Annual IEEE Symposium on the Foundations of Computer Science*, pp. 151–158.

Cook, S. A. (1985). A taxonomy of problems with fast parallel algorithms. *Information and Control*, **64**(1-3):2–22.

Doerre, J. (1996). Parsing for semidirectional lambek grammar is NP-complete. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, **13**(2):90–102.

Florencio, C. C. (2002). A note on the complexity of the associative-commutative lambek calculus. In *Proceedings of the 6th Int'l Workshop on Tree Adjoining Grammar and Related Framework*, pp. 101–106.

Hepple, M. (1992). Chart parsing lambek grammars: Model extensions and incrementatlity. In *Proceedings of the 14th Int'l Conference on Computational Linguistics*.

Kanovitch, M. (1991). The multiplicative fragment of linear logic is NP-complete. Technical Report X-91-13, University of Amsterdam, 1TL1 Prepublication Series.

Kanovitch, M. (1992). Horn-programming in linear logic is NP-complete. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, pp. 200–210.

Kurtonina, N. and M. Moortgat (1996). Structural control. In P. Blackburn and M. de Rijke, eds., *Specifying Syntactic Structures*. CSLI Publications.

Lincoln, P., J. Mitchell, A. Scedrov, and N. Shankar (1990). Decision problems on propositional linear logic. In *Proceedings of 31st Annual IEEE Symposium on the Foundations of Computer Science*.

Moot, R. and Q. Puite (1999). Proof nets for multimodal categorial grammars. In G.-J. M. Kruijff and R. T. Oehrle, eds., *Proceedings of the Conference on Formal Grammar*.

Morrill, G. (1996). Memoisation of categorial proof nets: parallelism in categorial processing. Technical Report LSI-96-24-R, Dept. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya.

Penn, G. (2002). A graph-theoretic approach to sequent derivability in the lambek calculus. In *Electronic Notes in Theoretical Computer Science: Proceedings of the Conference on Formal Grammar / 7th Meeting on Mathematics of Language*, volume 53.

Pentus, M. (1993). Lambek grammars are context free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pp. 429–433.

Pentus, M. (1997). Product-free lambek calculus and context-free grammars. *Journal of Symbolic Logic*, **62**(2):648–660.

Pentus, M. (2003). Lambek calculus is np-complete. Technical Report TR-2003005, CUNY Graduate Centre Ph.D. Program in Computer Science.

Sudborough, I. H. (1978). On the tape complexity of deterministic context-free languages. *Journal of the ACM,* **25**(3):405–414.

Chapter 7

Discovering a new class of languages

SEAN A. FULOP*

ABSTRACT. This paper outlines a new approach to specifying formal constraints on grammars and their languages, in pursuit of more far-reaching statements about possible human languages than, e.g., "they are context-free." We propose that possible human languages, conceived as term-labeled tree languages consisting of syntactic word trees annotated by semantic lambda terms, can be said to be *syntactically homogeneous* and *finitely illustratable.* These are new properties of term-labeled tree languages which, when used to specify a subset of all languages generated by a large class of multimodal type-logical grammars, yield a new class of languages which cross-cuts the traditional Chomsky hierarchy. A discovery procedure for type-logical lexicons is then outlined, whose range is proven to generate precisely these languages. Our new approach is actually quite old, having roots in ideas of the American Structuralist school.

7.1 Introduction

We set out to motivate and define a new class of formal languages that is not directly related to the Chomsky hierarchy, and which in fact completely cross-cuts it. This effort answers to a complaint that resonates through modern theoretical syntax, to the effect that the linguistic interest of the Chomsky hierarchy has long ago expired because the formal constraints on languages that it provides are not very useful for considering what possible human languages might be like. So when we consider, e.g., whether "human languages are context-free," we really mean the upper bound on the language complexity. We never ask "are all context-free languages possibly human," since we all know the answer: "of course not."

*Linguistics & Computer Science, The University of Chicago, Chicago, IL, U.S.A; sfulop@uchicago.edu

[0]Thanks to Jerry Sadock, Gerhard Jäger, and Jason Merchant for discussions that stimulated the development of this work. Thanks to the March 2002 colloquium audience at The University of Chicago for receiving the initial ideas. Thanks as always to Ed Stabler, for beginnings. This work was supported in part by a Research Incentive grant from the University of Chicago Dept. of Computer Science.

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

Our new class consists not of string languages, but of *term-labeled tree* languages, where a term-labeled tree is an unlabeled syntactic structure paired with a compositional semantic term (a lambda term). The class is defined with reference to a new formalization of the ideas of the American Structuralist school which permits the rigorous statement of conditions on the syntactic (and semantic) distribution of linguistic elements in a term-labeled tree (TLT) language. We call our new class of such languages *syntactically homogeneous,* and give a strict definition in terms of the structural conditions that must be met. Some of these languages have the further property of being *finitely illustratable,* forming an important subclass.

We next discuss a new algorithm which can discover lexicons for type-logical grammar (i.e. assignments of sets of syntactic types/categories to words) given a type logic (drawn from a broad but strictly defined class of "permitted" logics) and a sample of TLTs (or strings, increasing the computational complexity immensely) *which do not show their types.* The method is based on earlier work in this vein (Fulop, 2003) which discovered so-called *optimally unified* lexicons under the same learning conditions. Upon realizing that the class of optimally unified grammars that resulted from that work automatically adhered to certain Structuralist-style conditions on the distribution of linguistic elements, efforts were undertaken to improve the distributional analysis performed by the procedure beyond the naive one produced by optimally unifying the lexicon. The end product is a new, but similar, discovery procedure that outputs type-logical lexicons which we dub *structurally unified.* The TLT languages of such grammars, it turns out, are always syntactically homogeneous, and finitely illustratable. In fact, we demonstrate that the total range of the procedure, over all permitted type logics, is precisely a class of type-logical grammars which generate all and only the finitely illustratable syntactically homogeneous term-labeled tree languages generated by the permitted logics.

The mathematical interest of the preceding is, we hope, self-evident, but what about linguistics? It is conjectured that the condition of a TLT language's being at once syntactically homogeneous and finitely illustratable is sufficiently strong to characterize possible human languages fairly tightly. That is to say, given a reasonable vocabulary of human language words and/or morphemes, any such language meeting a couple of other simple conditions "could reasonably be" human. Syntactic homogeneity of a language is thus offered as a precise property capturing the primary essence of a language's "being human," which is surely more than could ever be said of context-freeness and the like.

7.2 Term-labeled tree languages

A type-logical grammar $G = \langle V_G, I_G, R_G \rangle$ consists of a vocabulary V_G, a lexical function I_G assigning sets of categories/types to words, and a type logic R_G which treats the categories as formulae. Owing to space constraints, we forego the details of type logic (v. Fulop 2002b; Moortgat 1997). The type logics and grammars that concern us are all based upon the nonassociative system of Lambek (1961) called 'NL,' whose formulae involve the two logical operators $/, \backslash$ which we call "slashes." The logics may be enriched with the addition of multiple indexed families of slashes, as well as unary modalities (Morrill, 1994) that license structural rules for rearranging the syntactic elements. We work with the logics in Gentzen's sequent formulation (Gentzen, 1934).

Given such a general landscape, we do have to limit ourselves to certain regions. It is not possible to prove anything worthwhile about a class of "all modally enriched type logics," and it is not even clear what this would mean because it imposes no restrictions at all on the nature of structural rules. It is clear that unrestricted logics of this kind can be created which imbue grammars based upon them with Turing-complete generating capacity (Carpenter, 1999), and this is not only undesirable, it makes the enterprise of proving a single sentence undecidable! On the other hand, many type logics have pleasant properties for sentence deduction; NL, for instance, enjoys Cut-elimination and has the subformula property, and its decidability is guaranteed as well.

We accordingly limit our further considerations to those classes $\mathfrak{L}_k$ of type logics R_G adhering to the following restrictions, which have been introduced and motivated elsewhere (Fulop, 2002b): 1. R_G possesses just a finite number of families of slashes, a finite number of families of unary modalities, and a finite number of structural rules which together preserve Cut-elimination and the subformula property of the base logic NL. 2. All structural rules are applicable only in the presence of certain unary modalities. 3. Each class $\mathfrak{L}_k$ of type logics consists of all logics satisfying the above, and whose well-formed formulae are restricted to those in which each subformula has at most k unary modal operators on its left edge.

Suffice it to say that a class $\mathfrak{L}_k$ with k "sufficiently large" (e.g. $k = 2$) includes logics with sufficient generating capacity to handle the kind of context-sensitivity occasionally demanded by natural languages, but any logic in the class remains decidable. Our results in this paper will generally be about the union of all such classes, denoted $\bigcup_k \mathfrak{L}_k$; this large class comprises all of the candidates for the Universal Type Logic in this setting, and we presume that some such logic could generate every possible human language, with only the lexicons varying. The precise weak generating capacity of grammars based on these logics in terms of the Chom-

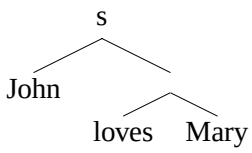
sky hierarchy is in general not particularly well-understood. It has been established that the base logic NL yields grammars of context-free languages exactly, while the addition of modals together with certain sets of modally licensed structural rules (frequently called "interaction postulates") yields some context-sensitivity. For current results along these lines, see Jäger (2003, 2002).

A key aspect of our learning algorithm is the connection between the syntactic structure of sentences, expressed as a tree of Lambek formulae, and the semantic structure, expressed as a lambda term. Any type logic $R_G \in \bigcup_k \mathfrak{L}_k$ can be shown to induce a fragment Λ_{NL} of the simply typed lambda calculus such that for every sequent $\Gamma \Rightarrow s$ in R_G provable by some proof Π there will be a term $N^s \in \Lambda_{\mathrm{NL}}$ such that N is a *homomorphic construction* of the sequent proof Π. Let Λ_{NL} denote the largest sublanguage of the simply typed lambda language each of whose terms N conforms to the following constraints: 1. Term N may not contain vacuous abstraction; 2. No subterms of N are without free variables or constants; 3. No subterm of N contains more than one free occurrence of the same variable; 4. Every prefix λx in N binds exactly one free variable occurrence, and that occurrence is either leftmost or rightmost in the subterm abstracted over by the prefix.

The facts mentioned above are proven elsewhere (Fulop, 2002a,b), and the idea of a homomorphic construction has been formalized in those references. Roughly, a lambda term is a homomorphic construction of a sequent proof just when the term can be used as a construction of the proof in the sense of Howard (1980), by invoking a generalized (homomorphic) version of the Curry-Howard formulae-as-types correspondence between lambda terms and proofs (see also Wansing 1992). The usual correspondence is generalized by interpreting any slash of the type logic as the functional type arrow '$\leftarrow$', and any modal operator is not interpreted at all.

A grammar G generates a language of word trees (from the antecedents of provable sentence sequents) labeled by these terms, called *term-labeled trees.* The lambda calculus is used in a standard fashion to model the compositional meaning structures of natural language; an example of a term-labeled tree that is *unsubtyped* (i.e. not showing subterm types) is:

(7.2.1) $((\mathbf{loves}(\mathbf{Mary}))(\mathbf{John}))^{\mathrm{s}}$:



We will in the sequel use a standard compact representation of syntax trees with the aid of square brackets. The bold faced items are constants of the lambda language, and the bolding of e.g. **Mary** is usually taken to be the meaning of 'Mary.'

Let us say that a grammar *G generates* a subtyped term-labeled tree

N^s: Struc

if and only if: 1. The tree Struc is generated by G with some proof Π in R_G; this means Π must be a proof of some particular sequent

Γ: Struc $\Rightarrow s$.

2. N^s is a homomorphic construction of Π. Now let us say that a grammar G generates an unsubtyped term-labeled sentence tree N^s_{ust}: Struc iff there is some subtyped term-labeled tree N^s: Struc generated by G such that N^s_{ust} is just N^s stripped of the type labels on its subterms. The set of (subtyped) term-labeled sentence trees generated by G is called the *(subtyped) term-labeled tree language* of G, and is denoted $\Lambda TL(G)$.

We will be most interested in the languages of unsubtyped term-labeled trees generated by various classes of type-logical grammars. We maintain that such TLT languages are the most relevant for representing natural languages, which we can all recognize as consisting of syntactically structured sentences with intended compositional meaning recipes. We assume here that meaning composition, at least in its skeletal form, is representable as a lambda term of the fragment defined above. It is important to note that this fragment is not in general sufficient for representing realistic semantic readings in many cases because of the strict limits placed on the locations of bound variables, but the composition of meaning from subterms can nonetheless be shown.

7.3 Distribution and syntactic homogeneity

We formalize the notions of syntactic distribution that were proposed by linguists of the American Structuralist tradition, in particular Bloomfield (1933) and Wells (1947). By the term *usage* it is meant a term-labeled tree $M[\mathbf{x}]$: Struc$[x]$ having a placeholder variable $\mathbf{x}$ in place of a subterm in the lambda term M and a corresponding placeholder x in place of the subtree in the tree Struc whose meaning occurs at the position occupied by $\mathbf{x}$ in the lambda term. The two placeholder positions as a pair will be termed the *focus* of a usage. For example, $(\mathbf{x}(\mathbf{John}))^s$: $[\text{John}, x]$ is a usage. This definition makes a usage a modernization of Bloomfield's notion of the function of a linguistic form. Any $\langle$meaning, tree$\rangle$ pair which can fill the slots in the usage to make a term-labeled sentence tree in the TLT language at hand is a Bloomfieldian *form* which is said to *have the usage*, and to *instantiate the focus.* All the forms in a language which have a particular

usage constitute a *form class* determined by that usage. Under our definitions, a form is then a pair consisting of a tree of words and a lambda term representing the meaning of the word tree.

Let us consider how usages can be considered equivalent in a term-labeled language. We begin by defining what it means for respective subterms of lambda terms M and N to function equivalently in their terms. The notion of two lambda-terms $M[\mathbf{x}]$ and $N[\mathbf{y}]$ having respective subterms $\mathbf{x}$ and $\mathbf{y}$ which *function equivalently* is defined inductively: 1. Any lambda terms M and N are subterms of themselves, and they function equivalently as such. 2. Two application terms $(M_1(M_2))[\mathbf{x}]$ and $(N_1(N_2))[\mathbf{y}]$ have respective subterms $\mathbf{x}$ and $\mathbf{y}$ which function equivalently just when either $M_1[\mathbf{x}]$ and $N_1[\mathbf{y}]$ have $\mathbf{x},\mathbf{y}$ functioning equivalently, or $M_2[\mathbf{x}]$ and $N_2[\mathbf{y}]$ have $\mathbf{x},\mathbf{y}$ functioning equivalently. 3. Two abstraction terms $(\lambda z.M)[\mathbf{x}]$ and $(\lambda z.N)[\mathbf{y}]$ have subterms $\mathbf{x},\mathbf{y}$ which function equivalently just when $M[\mathbf{x}]$ and $N[\mathbf{y}]$ have $\mathbf{x},\mathbf{y}$ functioning equivalently.

We now consider a similar notion for syntax trees, and define inductively when two subtrees are *positioned equivalently* within their parent trees. 1. Any two syntax trees S and T are subtrees of themselves, and are positioned equivalently as such. 2. Any two binary-branching syntax trees (S_1,S_2) and (T_1,T_2) have respective subtrees γ and δ positioned equivalently if and only if for either $i=1$ or $i=2$ the two trees S_i and T_i have the respective subtrees γ and δ positioned equivalently.

We can put these two together to define when two usages are equivalent in shape. Two usages $M_1[\mathbf{x}]:\mathsf{Struc}_1[x]$ and $M_2[\mathbf{y}]:\mathsf{Struc}_2[y]$ are said to be *shape-equivalent* just when: 1. The lambda terms $M_1[\mathbf{x}]$ and $M_2[\mathbf{y}]$ have respective subterms $\mathbf{x}$ and $\mathbf{y}$ which function equivalently. 2. The syntactic trees $\mathsf{Struc}_1[x]$ and $\mathsf{Struc}_2[y]$ have respective subtrees x and y positioned equivalently. These definitions clearly make shape-equivalence of usages reflexive, symmetric, and transitive, and thus the relation is a formal equivalence.

Any set of shape-equivalent usages will be called a *usage class*; a usage class which contains all possible usages that can be formed from a sample $\mathscr{D}$ of TLTs is termed a *complete* usage class *relative to* $\mathscr{D}$. The members of a usage class are each said to *illustrate* the class. When all the usages in a class determine the same form class, the usage class itself is held to also determine that form class; otherwise, the usage class does not determine a form class.

Example 1. *Here is a term-labeled tree language with four elements:*

(7.3.1)
$$\begin{array}{rlrl}
(\textbf{\textit{sings}}(\textbf{\textit{John}}))^s: & [\mathit{John},\mathit{sings}] & (\mathbf{x}(\textbf{\textit{John}}))^s: & [\mathit{John},x]\\
(\textbf{\textit{sings}}(\textbf{\textit{Mary}}))^s: & [\mathit{Mary},\mathit{sings}] & (\mathbf{x}(\textbf{\textit{Mary}}))^s: & [\mathit{Mary},x]\\
(\textbf{\textit{laughs}}(\textbf{\textit{Susan}}))^s: & [\mathit{Susan},\mathit{laughs}] & (\mathbf{x}(\textbf{\textit{Susan}}))^s: & [\mathit{Susan},x]\\
(\textbf{\textit{laughs}}(\textbf{\textit{Tom}}))^s: & [\mathit{Tom},\mathit{laughs}] & (\mathbf{x}(\textbf{\textit{Tom}}))^s: & [\mathit{Tom},x]
\end{array}$$

The four usages on the right can be formed from the respective term-labeled trees in the sample on the left. They are all shape-equivalent, and they comprise a complete *shape-equivalent set of usages which can be formed from the sample.*

Any form class determined by a complete usage class relative to a particular language will be termed a *part of speech* in that language, in the spirit of Bloomfield. In other words, a part of speech in a term-labeled tree language L is an equivalence class of trees of words (in fact, word occurrences) which is the form class determined by each of the usages in a usage class which is complete relative to L. The parts of speech of a term-labeled language are thus well-defined if and only if the members of each complete usage class are guaranteed to determine the same form class, throughout the language. In the example above, for instance, the four usages are a complete class relative to the four-sentence sample taken as a language. However, the first two usages form a usage class whose form class is different from that of the last two usages, as mentioned above. There is thus not a properly defined system of parts of speech in this four-sentence language. A term-labeled language L will be termed *syntactically homogeneous* if and only if it has well-defined parts of speech, meaning every complete usage class that can be formed from its trees determines a form class.

There is a second important property of TLT languages to be defined, which is entwined with the ways they are generated by their grammars. Let us say that a TLT $N_1 : \Gamma_1$ is a *recursively redundant extension* of a second TLT $N_2 : \Gamma_2$ just when: 1. they are shape-equivalent (i.e. instantiations of shape-equivalent usages); 2. there are corresponding foci within the two TLTs which are instantiated by respective subtrees $\nu_1 : \gamma_1$ and $\nu_2 : \gamma_2$ where the first is larger than the second; 3. the subtree $\nu_1 : \gamma_1$ in turn contains a subtree $\nu_1' : \gamma_1'$ which is shape-equivalent to $\nu_2 : \gamma_2$; 4. if a third TLT $N_3 : \Gamma_3$ is created from $N_2 : \Gamma_2$ by replacing $\nu_2 : \gamma_2$ with the result of removing $\nu_1' : \gamma_1'$ from $\nu_1 : \gamma_1$ (i.e. their difference), then $N_3 : \Gamma_3$ is generated by a grammar for the other two by means of a proof precisely equivalent to one which generates $N_2 : \Gamma_2$.

Example 2. *The first TLT below is a recursively redundant extension of the second TLT, as can by seen by instantiating the subtrees of the definition as shown.*

(7.3.2) $\boldsymbol{died}(\boldsymbol{the}((\boldsymbol{from}(\boldsymbol{the}(((\boldsymbol{by}(\boldsymbol{the}(\boldsymbol{sea})))\boldsymbol{town}))))\boldsymbol{man})):$

[[The [man [from [the [town [by [the sea]]]]]]] died]

(7.3.3) $\boldsymbol{died}(\boldsymbol{the}((\boldsymbol{from}(\boldsymbol{the}(\boldsymbol{town})))\boldsymbol{man})):$

[[The [man [from [the town]]]] died]

v_1: $\gamma_1 = \textbf{\textit{from}}(\textbf{\textit{the}}((\textbf{\textit{by}}(\textbf{\textit{the}}(\textbf{\textit{sea}})))\textbf{\textit{town}})$:
[from [the [town [by [the sea]]]]]
v_2: $\gamma_2 = \textbf{\textit{from}}(\textbf{\textit{the}}(\textbf{\textit{town}}))$: *[from [the town]]*
v_1': $\gamma_1' = \textbf{\textit{by}}(\textbf{\textit{the}}(\textbf{\textit{sea}}))$: *[by [the sea]]*

Now, a (possibly infinite) TLT language L is said to be *finitely illustratable* if, and only if, some (nonempty) finite set $\mathcal{C}$ of usages together illustrate the maximal set of usage classes which can be formed from trees of L so that no usage in $\mathcal{C}$ is a recursively redundant extension of any other usage in $\mathcal{C}$. Such a set $\mathcal{C}$ is called an *illustration set* for the language. If L is infinite, then the remaining infinite number of its usage classes not illustrated in $\mathcal{C}$ must all consist of usages that are recursively redundant extensions of members of usage classes illustrated by $\mathcal{C}$.

7.4 Discovering structurally unified lexicons

Since we are willing to provide our human language learning model with a Universal Type Logic, what remains to be learned of a particular language is precisely its lexicon. This includes learning the syntactic and semantic categories that are at work in the language. We would like to induce this information from an American Structuralist-style analysis of the language sample to determine which items are distributionally equivalent, or intersubstitutable in some sense.

An algorithm is presented below, dubbed SUTL, which induces a set of lexicons from a sample of term-labeled trees. It is based on the GFTL procedure outlined in Fulop (2003, 2002b), which learns general form type-logical lexicons from such a sample, and also employs the optimal unification procedure of Buszkowski and Penn (1990).

A *general form* lexicon I_{gf} for a TLT sample is defined to be one in which distinct variable primitive types (drawn from a denumerable set *Var*) each occur only once as an atomic type (though they may occur additionally as subtypes of complex types), and which generates exactly the sample as its TLT language when used together with R_G. The notion of a general form lexicon and its role as an intermediate step in grammar discovery was elucidated by Buszkowski and Penn (1990) in their work on the discovery of classical categorial grammars from syntactic skeletons.

Lemma 3. *For any two shape-equivalent term-labeled trees $T_1 = M[\mathbf{x}]$: $Struc[x]$ and $T_2 = M'[\mathbf{y}]$: $Struc'[y]$ in the sample data, in which form occurrences $\langle \mathbf{x}, x \rangle$ and $\langle \mathbf{y}, y \rangle$ occur equivalently (they may be occurrences of any possibly distinct trees of words), at least one general form lexicon I will be discovered by GFTL which assigns unifiable types to the syntactic word occurrences x and y if indeed they are*

words. More generally if x or y or both are not words, the respective proofs Π *and* Π' *of* T_1 *and* T_2 *using some such I will nonetheless be such that the word trees x in* Π *and y in* Π' *have types which are unifiable. Moreover, there will be some general form lexicon* I_{gf} *discovered by GFTL which has these properties for every such pair* x, y *of forms positioned equivalently in shape-equivalent trees.*

Proof. This lemma follows from the definition of the GFTL algorithm of Fulop (2003), which is guaranteed to find all possible proofs Π and Π' which can be obtained by using the lambda terms M, M' as proof-building recipes, in whatever type logic R is being used, of the respective term-labeled trees. It is a property of the principal typing carried out in GFTL that **x** and **y** are assigned to semantic types which are alphabetic variants. Since the lambda terms M and M' are homomorphic constructions of the respective proofs Π and Π' of the syntax trees which they label, it must then be possible (though not necessary) to prove each of them in such a way that the types of x and y are the same in the two proofs, up to alphabetic variation, and are thus unifiable.

Turning to the final statement in the lemma, because GFTL is defined to output a lexicon from every possible combination of ways of proving each term-labeled tree in the sample, at least one such set of proofs for the sample will be such that every set of shape-equivalent trees will have unifiable types assigned to every set of respective subtrees which are positioned equivalently. □

We now present the SUTL algorithm:

1. The input is a sample of term-labeled trees; some permitted type logic must also be chosen. Use GFTL to obtain the set $GF = \{I_1, \ldots, I_m\}$ of m general form lexicons for the sample. These have the property that every occurrence of a primitive type other than s in their categories is a new variable—they are totally ununified.

2. Create a set U of all usages which can be formed from the term-labeled sentence trees in the sample by replacing single lexical items (words, in the simplest view) with placeholders $\langle \mathbf{x}, x \rangle$.

3. Create a set UC of all the complete usage classes in U. Let $n = |UC|$.

4. For each usage class $\mathscr{C}_i$ in UC, form a set P_i of word instances which is the union of all the word instances in the form classes determined by the members of $\mathscr{C}_i$.

5. Select the first general form lexicon I_1.

6. Do for all $1 \leq i \leq n$: Each of the words p in P_i will have a set T_p of syntactic types that are assigned to *its instances in proofs of those sample trees that are shape-equivalent to the usage class* $\mathscr{C}_i$, using I_1. Create the set T_{P_i} which is the union $\bigcup_p T_p$.

7. Find all the optimal unifiers of the family $\mathscr{T} = \{T_{P1}, \ldots, T_{Pn}\}$ using the optimal unification procedure of Buszkowski and Penn (1990).

8. Discard any unifiers of the family which are incomplete, i.e. that fail to completely unify each set T_{Pi}. For a given general form lexicon I this may leave no unifiers, but we are guaranteed by Lemma 3 to have at least one lexicon that will produce a complete unifier, for any finite sample of TLTs.

9. A unifier of a lexicon formed in the above fashion relative to a set of term-labeled trees will be termed a *structural unifier.* Find all the unifications of the general form lexicon that result from applying each structural unifier in turn, if there are any. Add these unified lexicons to the set of lexicons to be output.

10. Select the next general form lexicon in the set GF and repeat from step 6, until there are no more lexicons in GF.

11. At last, output the complete set of *structurally unified* lexicons that has been built up through the iteration.

Structurally unified grammars have the interesting property that they generate their TLT languages so that distinct syntactic categories are assigned to syntactically distinct positions, while syntactically equivalent positions all have the same category. This property turns out to be equivalent to syntactic homogeneity of TLT languages.

Consider the class $\bigcup_R \Lambda TL(R)_{\text{homfin}}$ of finitely illustratable homogeneous TLT languages generated by permitted logics, obviously a proper subset of all the TLT languages of permitted logics. How does it relate to the class $\bigcup_R \Lambda TL(R)_{\text{sutl}}$ of all TLT languages of SUTL-range grammars under permitted logics?

Proposition 4.

$$\bigcup_R \Lambda TL(R)_{sutl} \subseteq \bigcup_R \Lambda TL(R)_{homfin}.$$

Lemma 5. *Any SUTL grammar G generates a TLT language whose usage classes are either illustrated in the learning sample or are recursively redundant extensions of those which are.*

Lemma 6. *For any SUTL grammar G, all lexical items in equivalently positioned foci in generated shape-equivalent TLTs will have the same type there.*

This lemma follows immediately from steps 6–8 of the SUTL algorithm together with the preceding lemma, whose proof we leave to the reader. Proposition 4 follows from these lemmas using an induction on the structure of the lambda term labels to establish the extension of the second lemma to all syntactically equivalent positions, no matter whether they are occupied by lexical items or larger trees.

Proposition 7.

$$\bigcup_R \Lambda TL_{homfin} \subseteq \bigcup_R \Lambda TL(R)_{sutl}.$$

Let us sketch a proof. Suppose a language $L \in \bigcup_R \Lambda TL(R)_{\text{homfin}}$. There is a finite set $\mathscr{D}$ of trees drawn from L from which every usage in an illustration set $\mathcal{C}$ can be formed, and containing every syntactically distinct word use in L at least once. There will then be some general form lexicon I_{gf} discovered by applying GFTL to $\mathscr{D}$ using some logic R (viz. a logic suitable to generate L), such that I_{gf} can be structurally unified using SUTL to obtain a lexicon I_{su} that generates L.

Corollary 8.

$$\bigcup_R \Lambda TL(R)_{homfin} = \bigcup_R \Lambda TL(R)_{sutl}.$$

It remains an empirical question, as Chomsky might put it, whether the class of possible human languages can in reality be construed to be either finitely illustratable or syntactically homogeneous. They are, at least, clear conditions which together substantially limit the class of languages, and we invite serious argument on the conjecture from an empirical linguistic perspective. Let us provisionally claim that no recognized syntactic phenomenon requires languages to be otherwise in any obvious way—suggesting that any human language can be generated by an SUTL grammar.

Bibliography

Bloomfield, L. (1933). *Language*. Allen and Unwin, London.

Buszkowski, W. and G. Penn (1990). Categorial grammars determined from linguistic data by unification. *Studia Logica*, **49**:431–454.

Carpenter, B. (1999). The Turing-completeness of multimodal categorial grammars. In J. Gerbrandy, M. Marx, M. de Rijke, and Y. Venema, eds., *JFAK: Essays dedicated to Johan van Benthem on the occasion of his 50th birthday*. Institute for Logic, Language, and Computation, University of Amsterdam. Available on CD-ROM at http://turing.wins.uva.nl.

Fulop, S. A. (2002a). On the logic and learning of language. Manuscript, University of Chicago.

Fulop, S. A. (2002b). Semantic bootstrapping of type-logical grammars. Manuscript, University of Chicago.

Fulop, S. A. (2003). Learnability of type-logical grammars. In L. Moss and G.-J. Kruijff, eds., *Proceedings of Formal Grammars / Mathematics of Language*, volume 53 of *Electronic Notes in Theoretical Computer Science*. Elsevier. To appear.

Gentzen, G. (1934). Untersuchungen über das logische Schliessen. *Math. Zeitschrift*, **39**:176–210, 405–431. English translation in Szabo (1969).

Howard, W. A. (1980). The formulas-as-types notion of construction. In J. P. Seldin and J. R. Hindley, eds., *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pp. 479–490. Academic Press, New York.

Jäger, G. (2002). Residuation, structural rules and context freeness. Manuscript, University of Potsdam.

Jäger, G. (2003). On the generative capacity of multi-modal categorial grammars. *Journal of Language and Computation*. To appear.

Lambek, J. (1961). On the calculus of syntactic types. In R. Jakobson, ed., *Structure of Language and its Mathematical Aspects*, Proceedings of Symposia in Applied Mathematics, pp. 166–178. American Mathematical Society, Providence, RI.

Moortgat, M. (1997). Categorial type logics. In J. van Benthem and A. ter Meulen, eds., *Handbook of Logic and Language*. Elsevier.

Morrill, G. V. (1994). *Type Logical Grammar: Categorial Logic of Signs*. Kluwer, Dordrecht.

Szabo, M. (1969). *The Collected Papers of Gerhard Gentzen*. North-Holland, Amsterdam.

Wansing, H. (1992). Formulas-as-types for a hierarchy of sublogics of intuitionist propositional logic. In D. Pearce and H. Wansing, eds., *Non-classical Logics and Information Processing*, volume 619 of *Lecture Notes in Artificial Intelligence*, pp. 125–145. Springer-Verlag, Berlin.

Wells, R. S. (1947). Immediate constituents. *Language*, **83**:81–117.

Chapter 8

m-Linear Context-Free Rewriting Systems as Abstract Categorial Grammars

PHILIPPE DE GROOTE* AND SYLVAIN POGODALLA*

ABSTRACT. This paper presents a coding of *m*-linear context-free rewriting systems (*m*-LCFRS) into abstract categorial grammars (ACG). Thus, it shows the latter formalism, which offers a powerful grammatical framework based on a small set of computational primitives, is able to reach some interesting classes of languages w.r.t. natural language modeling.

Introduction

Abstract categorial grammars (ACG) (de Groote 2001) have the property of explicitly generating two languages: an abstract one and an object one. The former may appear as a set of abstract grammatical structures and the latter as the set of the corresponding concrete forms. It then offers a framework in which other grammatical models can be encoded, both in the structures and in the expressions they allow.

This encoding has been done for any G in the class of CFGs (de Groote 2001) or in the classe of TAGs (de Groote 2002). This paper shows such an encoding for *m*-linear context-free rewriting systems (*m*-LCFRS). This enables ACGs to cover important (w.r.t. natural language modeling) classes of languages such as the ones generated by, because of the weak equivalence between them, multicomponent tree adjoining grammars (MCTAGs) (Weir 1988), multiple context-free grammars (MCFG) (Seki et al. 1991) or minimal grammars (MG) (Michaelis 2001).

*INRIA, LORIA, Nancy, France; {Philippe.deGroote,Sylvain.Pogodalla}@loria.fr

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

8.1 Abstract Categorial Grammars

This section defines the notion of an abstract categorial grammar. We first introduce the notions of *linear implicative types*, *higher-order linear signature*, *linear λ-terms* built upon a higher-order linear signature, and *lexicon*.

Definition. *Let A be a set of atomic types. The set $\mathcal{T}(A)$ of* linear implicative types *built upon A is inductively defined as follows:*

1. *if $a \in A$, then $a \in \mathcal{T}(A)$;*
2. *if $\alpha, \beta \in \mathcal{T}(A)$, then $(\alpha \multimap \beta) \in \mathcal{T}(A)$.*

Definition. *A* higher-order linear signature *consists of a triple $\Sigma = \langle A, C, \tau \rangle$, where:*

1. *A is a finite set of atomic types;*
2. *C is a finite set of constants;*
3. *$\tau : C \to \mathcal{T}(A)$ is a function that assigns to each constant in C a linear implicative type in $\mathcal{T}(A)$.*

Definition. *Let X be a infinite countable set of λ-variables. The set $\Lambda(\Sigma)$ of* linear λ-terms *built upon a higher-order linear signature $\Sigma = \langle A, C, \tau \rangle$ is inductively defined as follows:*

1. *if $c \in C$, then $c \in \Lambda(\Sigma)$;*
2. *if $x \in X$, then $x \in \Lambda(\Sigma)$;*
3. *if $x \in X$, $t \in \Lambda(\Sigma)$, and x occurs free in t exactly once, then $(\lambda x.t) \in \Lambda(\Sigma)$;*
4. *if $t, u \in \Lambda(\Sigma)$, and the sets of free variables of t and u are disjoint, then $(t\,u) \in \Lambda(\Sigma)$.*

As usual, $\Lambda(\Sigma)$ is provided with notion of capture avoiding α-conversion, substitution and β-reduction (Barendregt 1984).

Given a higher-order linear signature $\Sigma = \langle A, C, \tau \rangle$, each linear λ-term in $\Lambda(\Sigma)$ may be assigned a linear implicative type in $\mathcal{T}(A)$. This type assignment obeys an inference system whose judgements are sequents of the following form:

$$\Gamma \vdash_{\Sigma} t : \alpha$$

where:

1. Γ is a finite set of λ-variable typing declarations of the form '$x : \beta$' (with $x \in X$ and $\beta \in \mathcal{T}(A)$), such that any λ-variable is declared at most once;
2. $t \in \Lambda(\Sigma)$;
3. $\alpha \in \mathcal{T}(A)$.

The axioms and inference rules are the following:

$$\vdash_\Sigma c : \tau(c) \quad \text{(cons)} \qquad \frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma (\lambda x.t) : (\alpha \multimap \beta)} \quad \text{(abs)}$$

$$x : \alpha \vdash_\Sigma x : \alpha \quad \text{(var)} \qquad \frac{\Gamma \vdash_\Sigma t : (\alpha \multimap \beta)}{\Gamma, \Delta \vdash_\Sigma (t\,u) : \beta} \quad \text{(app)}$$

Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be two higher-order linear signatures, a lexicon $\mathcal{L} : \Sigma_1 \to \Sigma_2$ is a realization of Σ_1 into Σ_2, i.e., an interpretation of the atomic types of Σ_1 as types built upon A_2 together with an interpretation of the constants of Σ_1 as linear λ-terms built upon Σ_2. These two interpretations must be such that their homomorphic extensions commute with the typing relations. More formally:

Definition. *a* lexicon $\mathcal{L}$ *from* $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ *to* $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ *is defined to be a pair* $\mathcal{L} = \langle F, G \rangle$ *such that:*

1. $F : A_1 \to \mathcal{T}(A_2)$ *is a function that interprets the atomic types of* Σ_1 *as linear implicative types built upon* A_2*;*
2. $G : C_1 \to \Lambda(\Sigma_2)$ *is a function that interprets the constants of* Σ_1 *as linear* λ*-terms built upon* Σ_2*;*
3. *the interpretation functions are compatible with the typing relation, i.e., for any* $c \in C_1$*, the following typing judgement is derivable:*

$$\vdash_{\Sigma_2} G(c) : \hat{F}(\tau_1(c)),$$

where $\hat{F}$ *is the unique homomorphic extension of* F*. Similarly,* $\hat{G}$ *is the unique* λ*-term homomorphism from* $\Lambda(\Sigma_1)$ *to* $\Lambda(\Sigma_2)$ *that extends* G*.*

In the sequel, when '$\mathcal{L}$' will denote a lexicon, it will also denote the homomorphisms $\hat{F}$ and $\hat{G}$ (the intended meaning will be clear from the context).

We are now in a position of defining the notion of abstract categorial grammar.

Definition. *An* abstract categorial grammar *is a quadruple* $\mathcal{G} = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$ *where:*

1. Σ_1 *and* Σ_2 *are two higher-order linear signatures; they are called the* abstract vocabulary *and the* object vocabulary*, respectively ;*

2. $\mathcal{L} : \Sigma_1 \to \Sigma_2$ *is a lexicon from the abstract vocabulary to the object vocabulary;*

3. *s is an atomic type of the abstract vocabulary; it is called the* distinguished type *of the grammar.*

The abstract language $\mathscr{A}(\mathcal{G})$ *generated by* $\mathcal{G}$ *is defined as follows:*

$$\mathscr{A}(\mathcal{G}) = \{t \in \Lambda(\Sigma_1) \mid \vdash_{\Sigma_1} t : s \text{ is derivable}\}$$

In words, the abstract language generated by $\mathcal{G}$ *is the set of closed linear* λ*-terms, built upon the abstract vocabulary* Σ_1*, whose type is the distinguished type s. On the other hand, the* object language $\mathscr{O}(\mathcal{G})$ *generated by* $\mathcal{G}$ *is defined to be the image of the abstract language by the term homomorphism induced by the lexicon* $\mathcal{L}$*:*

$$\mathscr{O}(\mathcal{G}) = \{t \in \Lambda(\Sigma_2) \mid \exists u \in \mathscr{A}(\mathcal{G}).\, t = \mathcal{L}(u)\}$$

8.2 *m*-Linear Context-Free Rewriting Systems

In this section, we directly define the class of *m-linear context-free rewriting systems* (Vijay-Shanker et al. 1987; Weir 1988), even if it can be defined as a proper subclass of the class of *multiple context-free grammars* (Seki et al. 1991; Michaelis 2001), the latter themselves being a subclass of the *generalized context-free grammars* introduced by Pollard (1984).

Definition. *A five-tuple* $G = \langle N, O, F, R, S\rangle$ *is a* m-linear context-free rewriting system (m-LCFRS) *if:*

1. N *is a finite non-empty set of nonterminal symbols;*

2. $O = \bigcup_{i=1}^{m} \langle \Sigma^* \rangle^i$ *for some finite non-empty set* Σ *of terminal symbols with* $\Sigma \cap N = \emptyset$*. O is thet set of all non-empty finite tuples of finite strings in* Σ *such that each tuple has at most m components;*

3. F *is a finite subset of* $\bigcup_{n \leq m} F_n \setminus \{\emptyset\}$ *where* F_n *is the set of partial functions from* $\langle O \rangle^n$ *into O. Moreover, for each* $f \in F$*, there exist* $n(f) \in \mathbb{N}$*,* $d(f) \in \mathbb{N}$ *and* $d_1(f), \ldots, d_{n(f)}(f) \in \mathbb{N}$ *such that:*

$$f : \langle \langle \Sigma^* \rangle^{d_1(f)}, \ldots, \langle \Sigma^* \rangle^{d_{n(f)}(f)} \rangle \mapsto \langle \Sigma^* \rangle^{d(f)} \text{ and}$$

$$f(\langle\langle x_{11},\dots x_{1d_1(f)}\rangle,\dots,\langle x_{n(f)1},\dots x_{n(f)d_{n(f)}(f)}\rangle\rangle = \langle f_1(\langle y_{11},\dots y_{1n(f_1)}\rangle),\dots, f_{d(f)}(\langle y_{d(f)1},\dots y_{d(f)n(f_{d(f)})}\rangle)\rangle$$

with $\bigcup_{i=1}^{d(f)}\bigcup_{j=1}^{n(f_i)}\{y_{ij}\} = \bigcup_{i=1}^{n(f)}\bigcup_{j=1}^{d_i(f)}\{x_{ij}\}$ *and every* f_i *is linear in each of the* y_{jk}*, i.e.:* $\forall f \in F, \forall i \in [1,d(f)], \exists \xi_{i0},\dots,\xi_{in(f_i)} \in \Sigma^*$ *such that*

$$f_i(\langle y_{i1},\dots y_{in(f_i)}\rangle) = \xi_{i0} y_{i\sigma_i(1)} \xi_{i1} y_{i\sigma_i(2)} \cdots y_{i\sigma_i(n(f_i))} \xi_{in(f_i)}$$

with σ_i *permutation on* $[1,n(f_i)]$*. We call* Ξ *the (finite) set of all the* ξ_{ij} *that are defined in that way.*

4. $R \subset \bigcup_{n\le m}(F\cap F_n)\times N^{n+1}$ *is a finite set of* rewriting rules.
   We usually write a rule $r = \langle f, X_0, X_1,\dots,X_n\rangle \in (F\cap F_n)\times N^{n+1}$ *for some* $n \in \mathbb{N}$ *as* $X_0 \to f(\langle X_1,\dots,X_n\rangle)$*, and* $X_0 \to f()$ *if* $n = 0$*. If* $n = 0$*,* r *is* terminating*, else it is* nonterminating*;*

5. $S \in N$ *is the* distinguished start symbol*;*

6. *there is a function* d_G *from* N *to* $\mathbb{N}$ *such that if* $X_0 \to f(\langle X_1,\dots,X_n\rangle) \in R$*, then* $d(f) = d_G(X_0)$ *and* $d_i(f) = d_G(X_i)$ *where* $d(f)$ *and* $d_i(f)$ *are as in 3;*

7. $d_G(S) = 1$.

We can now define the languages these grammars generate:

Definition. *For each* $X \in N$ *and* $k \in \mathbb{N}$*, the set* $L_G^k(X) \subset O$ *is defined as follows:*

- $L_G^0(X) = \{\theta | X \to f() \in R \text{ and } f() = \theta\}$
- *let* $F_X^n = \{f \in F | \exists X \to f(\langle X_1,\dots,X_n\rangle) \in R\}$*. Then* $L_G^{k+1}(X) = L_G^k(X) \bigcup_{n\le m}\bigcup_{f\in F_X^n} f(\langle L_G^k(X_1),\dots,L_G^k(X_n)\rangle)$

X derives θ in G *if there exist* $X \in N$ *and* $k \in \mathbb{N}$ *such that* $\theta \in L_G^k(X)$*.* θ *is called an* X-phrase in G*. For each* $X \in N$*, the* language derivable from X by G *is* $L_G(X) = \bigcup_{k\in\mathbb{N}} L_G^k(X)$*, and* $L_G(S)$ *is the* language derivable by G*.*

In addition, we need the definition of the associated parse trees.

Definition. $T = (D_\gamma, V)$ *is a* tree over V *iff* γ *is a function from* D_γ *into* V *where the domain* D_γ *is a finite subset of* $\mathbb{N}^\star$ *such that:*

1. *if* $q \in D_\gamma, p < q,$ *then* $p \in D_\gamma$*;*

2. *if* $p\cdot j \in D_\gamma, j \in \mathbb{N},$ *then* $p\cdot 1, p\cdot 2,\dots,p\cdot(j-1) \in D_\gamma$

where $\mathbb{N}^\star$ *is the free monoid generated by* $\mathbb{N}$, $\cdot$ *is the binary operation,* 0 *is the identity and for* $q \in \mathbb{N}^\star$, $p \leq q$ *iff there is a* $r \in \mathbb{N}^\star$ *such that* $q = p \cdot r$, *and* $p < q$ *iff* $p \leq q$ *and* $p \neq q$.

We say that $D_\gamma = Dom(T)$ *and* $\gamma = Label(T)$.

Definition. *For each* $X \in N$ *and* $k \in \mathbb{N}$, *the set* $PT_G^k(X)$ *of the* parse trees derived from X *is defined as follows:*

- $PT_G^0(X) = \{(\{0\}, \{0 \mapsto (X, f)\}) | X \to f() \in R\}$
- *let* $r_X^n = \{\langle f, X, X_1, \ldots, X_n\rangle \in R\}$. *Then* $PT_G^{k+1}(X) = \bigcup_{n \leq m} \bigcup_{r \in r_X^n} T_r$ *with* $T_r = \{(\{0\} \bigcup \cup_i \{i \cdot D_i\}, \{0 \mapsto (X, f)\} \bigcup \cup_i \{i \cdot \omega \mapsto \gamma_i(\omega)) | (D_i, \gamma_i) \in PT_G^k(X_i)\}$

X derives T in G *if there exists* $X \in N$ *and* $k \in \mathbb{N}$ *such that* $T \in PT_G^k(X)$. T *is called a* X-parse tree in G. *For each* $X \in N$, *the* parse trees derivable from X by G *is* $PT_G(X) = \bigcup_{k \in \mathbb{N}} PT_G^k(X)$, *and* $PT_G = \bigcup_{X \in N} PT_G(X)$ *is the* set of parse trees of G.

Note that from PT_G, we can obviously recover $L_G(X)$ with a linearization function Lin, for all $X \in N$. Indeed, by induction, if $T \in PT_G^0(X)$, it exists $X \to f() \in R$, and with $\mathrm{Lin}(T) = f() = \theta$, $\theta \in L_G^0(X) \subset L_G(X)$. If $T \in PT_G^{k+1}(X)$ there exists $\langle f, X, X_1, \ldots, X_n\rangle \in R$ and $T_i \in PT_G^k(X_i)$. By induction hypothesis, for each $i \leq n$, $\mathrm{Lin}(T_i) \in L_G^k(X_i)$, then $\mathrm{Lin}(T) = f(\langle \mathrm{Lin}(T_1), \ldots, \mathrm{Lin}(T_n)\rangle) = \theta$ is such that $\theta \in L_G^{k+1}(X) \subset L_G(X)$.

8.3 Building an ACG Equivalent to an *m*-LCFRS

In this section, we present the main result of this paper.

Theorem. *For every m-LCFRS* $G = \langle N, O, F, R, S\rangle$, *there exists an ACG* $\mathcal{G}_G$ *such that:*

- *the abstract language* $\mathscr{A}(\mathcal{G}_G)$ *of normal terms is isomorphic to the set of parse-trees of* G;
- *the language generated by* G *coincides with the object language of* $\mathcal{G}_G$, *i.e.* $\mathscr{O}(\mathcal{G}_G) = L_G(S)$.

Proof. First, we add to G a new symbol S' and a new rule $S' \to f_{S'}(S)$ with $f_{S'}(\langle x\rangle) = x$. This is because we model tuples with higher-order functions, and we need to come back to strings at the end. Nevertheless, the generated language is unchanged (we could avoid this by garanteeing that G is not recursive in S).

Then, we define $\mathcal{G}_G = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S'\rangle$ with the abstract vocabulary $\Sigma_1 = \langle A_1, C_1, \tau_1\rangle$ such that:

- $A_1 = N \cup \{S'\}$
- the set of constants C_1 is a set of symbols in one-to-one correspondance with R
- for $c \in C_1$ and $\langle f, X_0, \ldots, X_n \rangle$ the corresponding rule, $\tau_1(c) = X_1 \multimap \cdots \multimap X_n \multimap X_0$

Using the usual encoding of the type σ of strings from an arbitrary atomic type $*$ with $\sigma = * \multimap *$, the empty string being $\lambda x.x$, the string made from one character a being $\lambda x.xa$ and the concatenation operation $+$ being defined as $\lambda f.\lambda g.\lambda x.f(gx)$ (which is an associative operator that admits the identity function as a unit), we can define the object vocabulary as follows (considering σ as an atomic type):

- $A_2 = \{\sigma\}$;
- $C_2 = \{f_1(), \ldots, f_{d(f)}() | \exists X \to f() \in R, f() = \langle f_1(), \ldots, f_{d(f)}() \rangle\} \cup \Xi$;
- τ_2 is defined as assigning the type σ to each $c \in C_2$.

Then we define the lexicon $\mathcal{L}$ with:

- $\mathcal{L}(S') = \sigma$, then for every $X \in N$, $\mathcal{L}(X) = (\underbrace{\sigma \multimap \ldots \multimap \sigma}_{d_G(X) times} \multimap \sigma) \multimap \sigma$ (note that $\mathcal{L}(S) = (\sigma \multimap \sigma) \multimap \sigma$);
- for every $c \in C_1$ that corresponds to a rule $\langle f, X_0, X_1, \ldots, X_n \rangle$ with $X_0 \neq S'$ and

$$f(\langle\langle x_{11}, \ldots x_{1d_1(f)}\rangle, \ldots, \langle x_{n(f)1}, \ldots x_{n(f)d_{n(f)}(f)}\rangle\rangle) = \langle f_1(\langle y_{11}, \ldots y_{1n(f_1)}\rangle), \ldots, f_{d(f)}(\langle y_{d(f)1}, \ldots y_{d(f)n(f_{d(f)})}\rangle)\rangle$$

with $f_i(\langle y_{i1}, \ldots y_{in(f_i)}\rangle) = \xi_{i0} y_{i\sigma_i(1)} \xi_{i1} y_{i\sigma_i(2)} \cdots y_{i\sigma_i(n(f_i))} \xi_{in(f_i)}$

Let $u_i = \xi_{i0} + y_{i\sigma_i(1)} + \xi_{i1} + y_{i\sigma_i(2)} \cdots y_{i\sigma_i(n(f_i))} + \xi_{in(f_i)}$ for each $i \in [1, d(f)]$, then, with $\vec{x}_i = x_{i1} \cdots x_{id_i(f)}$, we have:

$$\mathcal{L}(c) = \lambda T_1 \cdots T_n g.T_1(\lambda \vec{x}_1.T_2(\lambda \vec{x}_2.T_3(\cdots T_n(\lambda \vec{x}_{n(f)}.g u_1 \cdots u_{d(f)}) \cdots)))$$

Indeed, this is a term of $\Lambda(\Sigma_2)$ because of the linearity condition on f and the f_i.

Note that if $c : X$ and X is an atomic type, then f comes from a terminating rule, $f() = \langle f_1(), \ldots, f_{d(f)}() \rangle$ and $\mathcal{L}(c) = \lambda g.g f_1() \cdots f_{d(f)}()$.

If c correspond to the rule $\langle f_{S'}, S', S \rangle$, then $\mathcal{L}(c) = \lambda t.t(\lambda x.x)$.

Then we build I a mapping from the normal terms of $\Lambda(\Sigma_1)$ that are of *atomic types* onto the derivation trees of G by induction as follows:

- if $c \in C_1$, c of type $\alpha \in N$ and correspond to the rule $\alpha \to f()$, $I(c) = (\{0\}, \{0 \mapsto (\alpha, f)\})$
- if $t = cu_1 \cdots u_n$ (in head normal form) is of type $\alpha \in N$ with $c \in C_1$ corresponding to the rule $\langle f, \alpha, \alpha_1, \cdots, \alpha_m \rangle$ where u_i is of type $\alpha_i \in N$, then $m = n$ (because t is of atomic type) and $I(t) = (\{0\} \bigcup_{i=1}^{n} i \cdot \mathrm{Dom}(I(u_i)), \{0 \mapsto (\alpha, f),$ and for all $i \leq n$, $i \cdot w \mapsto \mathrm{Label}(I(u_i))(w))$
- no other case has to be considered since we consider only terms with atomic type.

By induction on the parse trees of G, is it easy to prove that I is an isomorphism. Induction hypothesis $H(n)$: $\forall k \leq n$, $\forall X \in N$, $T \in PT_G^k(X)$ iff there exists a unique $t \in \Lambda(\Sigma_1)$, in normal form and of atomic type, such that $T = I(t)$.

- $n = 0$: $\forall X \in N$, $T \in PT_G^0(X)$ iff there exists $X \to f() \in R$, iff there exists a unique $c \in C_1$ corresponding to $X \to f()$ and c of atomic type X, iff there exists a unique $t = c \in C_1$ such that $T = I(t)$ (by definition of T and $I(t)$);
- $n > 1$: if $k < n$, its trivial by induction hypothesis. If $k = n$, $\forall X \in N$, $T \in PT_G^k(X)$ iff there exists $\langle f, X, X_1, \ldots, X_n \rangle \in R$ and $T_i \in PT_G^{k-1}(X_i)$, iff there exists a unique $c \in C_1$ corresponding to $\langle f, \alpha, \alpha_1, \cdots, \alpha_n \rangle$ and (induction hypothesis) for all $i \leq n$, $\exists! u_i \in \Lambda(\Sigma_1)$ such that $T_i = I(u_i)$, iff there exists a unique $t = cu_1 \cdots u_n$ such that $T = I(t)$ (by definition of T and $I(t)$).

We prove $\mathscr{O}(\mathcal{G}_G) = L_G(S)$ in two steps:

- any tuple $\langle x_1, \ldots, x_n \rangle$ is modeled in $\Lambda(\Sigma_2)$ by $\lambda f. f x_1 \cdots x_n$
- by induction, we prove that for every $t \in \Lambda(\Sigma_1)$ that is of atomic type, $\mathcal{L}(t) = \lambda g. g x_1 \cdots x_n$ where $\langle x_1, \ldots, x_n \rangle = \mathrm{Lin}(I(t))$.

  It is clear if $t = c \in C_1$. If $t = ct_1 \ldots t_n$, with $c \in C_1$ corresponding to the rule $\langle f, X_0, X_1, \cdots, X_n \rangle$, and by induction hypothesis, for each $i \leq n$, $\mathcal{L}(t_i) = \lambda g. g x_{i1} \cdots x_{id_i(f)}$ with $\mathrm{Lin}(I(t_i)) = \langle x_{i1} \cdots x_{id_i(f)} \rangle$.

  So, using the same notations as in the definition of $\mathcal{L}$, we have

$$\begin{aligned}
\mathcal{L}(t) &= \mathcal{L}(c)\mathcal{L}(t_1) \cdots \mathcal{L}(t_n) \\
&= (\lambda T_1 \cdots T_n g. T_1 (\lambda \vec{x}_1. T_2(\cdots)))(\lambda h. h \vec{x_1}) \cdots \mathcal{L}(t_n) \\
&= (\lambda T_2 \cdots T_n g. T_2 (\lambda \vec{x}_2. T_3(\cdots)))\mathcal{L}(t_2) \cdots \mathcal{L}(t_n) \\
&\vdots \\
&= \lambda g. g u_1 \cdots u_{d(f)}
\end{aligned}$$

Since $\mathrm{Lin}(I(t)) = \langle u_1, \ldots, u_n \rangle$ when identifying strings and their coding in $\Lambda(\Sigma_2)$, this ends the proof (by definition of the u_i).

□

8.4 Example

This section provides an example from $G = \langle N, O, F, R, S \rangle$ the 5-LCFRS defined as follows:

- $N = \{A, S\}$
- we have the following rules:

$$\begin{array}{lll} r_0: & S' \to S & f_0(\langle x \rangle) = x \\ r_1: & S \to A & f_1(\langle x_1, \ldots, x_5 \rangle) = \langle x_1 + \cdots + x_5 \rangle \\ r_2: & A \to A & f_2(\langle x_1, \ldots, x_5 \rangle) = \langle x_1 + a, \ldots, x_5 + e \rangle \\ r_3: & A & f_3() = \langle a, b, c, d, e \rangle \end{array}$$

G generates the language $L_G(S') = \{a^n b^n c^n d^n e^n | n > 0\}$.

Following the rules given in the previous section to build the ACG $\mathcal{G}_G$, we have:

$A_1 = \{A, S, S'\}$ $\qquad$ $A_2 = \{\sigma\}$

$C_1 = \{r_0, r_1, r_2, r_3\}$ $\qquad$ $C_2 = \{a, b, c, d, e\}$

τ_1 is such that $\tau_1(r_0) = S \multimap S'$, $\tau_1(r_1) = A \multimap S$, $\tau_1(r_2) = A \multimap A$, $\tau_1(r_3) = A$ $\qquad$ τ_2 is the constant function σ

$$\begin{array}{l} \mathcal{L}(S') = \sigma \\ \mathcal{L}(S) = (\sigma \multimap \sigma) \multimap \sigma \\ \mathcal{L}(A) = (\sigma \multimap \sigma \multimap \sigma \multimap \sigma \multimap \sigma \multimap \sigma) \multimap \sigma \\ \mathcal{L}(r_0) = \lambda t.t(\lambda x.x) \\ \mathcal{L}(r_1) = \lambda T.\lambda g.T(\lambda x_1 x_2 x_3 x_4 x_5.g(x_1 + x_2 + x_3 + x_4 + x_5)) \\ \mathcal{L}(r_2) = \lambda T.\lambda g.T(\lambda x_1 x_2 x_3 x_4 x_5.g(x_1 + a)(x_2 + b)(x_3 + c)(x_4 + d)(x_5 + e)) \\ \mathcal{L}(r_3) = \lambda g.gabcde \end{array}$$

For instance, we can compute:

$$\begin{array}{ll} \mathcal{L}(r_0(r_1(r_2(r_3)))) & = \mathcal{L}(r_0)(\mathcal{L}(r_1)(\mathcal{L}(r_2)(\mathcal{L}(r_3)))) \\ & = \mathcal{L}(r_0)(\mathcal{L}(r_1)(\lambda g.g(a+a)(b+b)(c+c)(d+d)(e+e))) \\ & = \mathcal{L}(r_0)(\lambda g.g(a+a+b+b+c+c+d+d+e+e)) \\ & = a+a+b+b+c+c+d+d+e+e \end{array}$$

Conclusion

This paper gives a coding of *m*-linear context-free rewriting systems into abstract categorial grammars. After the coding of CFGs and TAGs, it shows ACGs to cover still a larger class of languages. Identifying their exact expressive power remains an open problem.

Importantly, it also outlines the ability of ACGs to appear as the kernel of a grammatical framework in which other existing grammatical models may be encoded.

Bibliography

Barendregt, H. P. (1984). *The lambda calculus, its syntax and semantics*. North-Holland. Revised edition.

de Groote, P. (2001). 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.

de Groote, P. (2002). Tree-adjoining grammars as abstract categorial grammars. In *TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pp. 145–150. Università di Venezia.

Michaelis, J. (2001). Transforming linear context-free rewriting systems into minimalist grammars. In *Proceedings of the conference Logical Aspects of Computational Linguistics (LACL '01)*, volume 2099 of *LNCS/LNAI*.

Pollard, C. (1984). *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*. Ph.D. thesis, Stanford University, CA.

Seki, H., T. Matsumura, M. Fujii, and T. Kasami (1991). On multiple context-free grammars. *Theoretical Computer Science*, **223**:87–120.

Vijay-Shanker, K., D. J. Weir, and A. K. Joshi (1987). Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th ACL*, pp. 104–111. Stanford, CA.

Weir, D. J. (1988). *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania.

Chapter 9

Learning local transductions is hard

MARTIN JANSCHE*

ABSTRACT. Local deterministic string-to-string transductions are generalizations of morphisms on free monoids. Learning local transductions reduces to inference of monoid morphisms. However, learning a restricted class of morphisms, the so-called fine morphisms, is an intractable problem, because the decision version of the empirical risk minimization problem contains an **NP**-complete subproblem.

9.1 Introduction

Symbolic approaches to natural language processing (NLP) based on finite automata (Roche and Schabes, 1997) suffer from a shortage of robust, practical inference procedures. If inductive inference is understood as 'identification in the limit' (Gold, 1967), then regular languages cannot be inferred on the basis of positive data alone. Most learning algorithms proposed for NLP tasks therefore employ different notions of inference, or aim at more restricted classes of languages, and they generally have to work with imperfect data.

This paper is about the problem of learning local transducers, a restricted subclass of the generalized sequential machines (Eilenberg, 1974), and inference is understood as empirical risk minimization. The general problem is illustrated by a specific NLP task, namely learning letter-to-sound rules (see for example van den Bosch, 1997) from a pronunciation dictionary. In this task, the training samples are pairs of strings, consisting of a string of letters – for example ⟨shoes⟩ – and a string of phonemes – for example /ʃuz/. Note that no relation between individual letters and phonemes is specified, which is to say one does **not** know whether the second symbol in /ʃuz/, the phoneme /u/, corresponds to the second symbol in ⟨shoes⟩,

*Linguistics, The Ohio State University, Columbus, OH, U.S.A.;jansche@ling.ohio-state.edu

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

the letter ⟨h⟩. In this sense the present task is markedly different from other common NLP tasks – such as learning part-of-speech assignment rules – where explicit correspondences between input and output symbols exist.

In general, a letter string may correspond to a longer phoneme string, for example[1]

⟨mutualism⟩ (9 letters) /mjuʧəwəlɪzəm/ (12 phonemes),

or to a shorter phoneme string, such as

⟨featherweight⟩ (13 letters) /fɛðɚwet/ (7 phonemes);

and even if the two strings happen to have the same length, as in

⟨parliamentarianism⟩ (18 letters) /paɹləməntɛɹiənɪzəm/ (18 phonemes),

no alignment is implied. One usually assumes that letter strings are of equal length or longer than their corresponding phoneme strings. While clearly false in an absolute sense, this assumption is true for most English words (more than 98% of the entries in the CMU pronouncing dictionary), and workarounds for cases where it seems to break down have been suggested, for example the transcription system used by NETtalk (Sejnowski and Rosenberg, 1987).

Learning letter-to-sound rules can be conceptualized as grammatical inference of specific subclasses of rational transductions. For the class of subsequential transductions, limit-identification is possible (Oncina et al., 1993) and has been applied to the closely related problem of phonemic modeling (Gildea and Jurafsky, 1996), but only after modifications and incorporation of domain-specific knowledge. It can be shown that the algorithm proposed by Oncina et al. (1993) has poor out-of-class behavior and is brittle in the presence of imperfect data; furthermore its hypothesis space, the class of subsequential transductions, is arguably too general for the present task. Almost all approaches to learning letter-to-sound rules assume, justifiably, that the hypothesis space is restricted to the analog of the locally testable languages in the strict sense (McNaughton and Papert, 1972), which are limit-identifiable (García and Vidal, 1990). We call the analogous class of transductions *local transductions*.

Local transductions are computed by scanner transducers, which move a sliding window of fixed size across the input string and produce a string of output symbols for each window position; concatenating these output strings yields the overall output of the transducer. Since the size of the sliding window is fixed, one can assume without loss of generality that it is 1. If a larger window size n is

[1] The following examples are taken from the CMU pronouncing dictionary (Weide, 1998). Phonemic transcriptions have been changed to use IPA (International Phonetic Association, 1999).

needed, one can simply change the input alphabet to consist of *n*-tuples of symbols, and such a modified alphabet is obviously still finite; alternatively, one can think of this modification as a preprocessing step that applies a simple subsequential transducer to each input string. Functional transductions that examine individual input symbols (letters, or *n*-tuples of letters) without taking any context into account (a finite amount of history or lookahead can be incorporated into the modified symbols created by the preprocessing step) can be realized by generalized sequential machines with a trivial one-state topology and correspond exactly to morphisms on free monoids (Eilenberg, 1974, p. 299).

The subsequent discussion will refer to a finite set Σ of input symbols and a finite set Γ of output symbols. The free monoid generated by Σ is called Σ^* and has the property that every element (string) $x \in \Sigma^*$ has a unique factorization in terms of elements of Σ. This means that a morphism $g : \Sigma^* \to \Gamma^*$ on free monoids is completely characterized by $g|_\Sigma$, its restriction to Σ. Conversely, this allows us to define the following notion:

Definition 1 (Free monoid morphism). Given a function $f : \Sigma \to \Gamma^*$ define f^* to be the unique monoid morphism $f^* : \Sigma^* \to \Gamma^*$ such that $f^*(x) = f(x)$ for all $x \in \Sigma$; $f^*(\varepsilon) = \varepsilon$; and $f^*(yz) = f^*(y)\ f^*(z)$ for all $y, z \in \Sigma^*$.

At the core of the learning task is then the problem of finding a suitable function $f : \Sigma \to \Gamma^*$ mapping from individual input symbols to output strings. In this paper we focus on two classes of functions. The first class restricts the codomain to strings of length one. If $f : \Sigma \to \Gamma$ is such a function – an alphabetic substitution – then f^* is a *very fine morphism*, according to Eilenberg (1974, p. 6). The second class is a superset of the first and allows the empty string in the codomain. Eilenberg (1974) calls the morphism f^* a *fine morphism* if its underlying function f is of type $\Sigma \to \{\varepsilon\} \cup \Gamma$. For the specific problem of learning letter to sound rules we can restrict our attention to fine morphisms, since by our previous assumption letter strings are never shorter than their corresponding phoneme strings, so a fine morphism is formally adequate. In general we may want to consider other kinds of morphisms, for example those arising from functions of type $\Sigma \to \{\varepsilon\} \cup \Gamma \cup \Gamma^2$. However, most practically relevant classes of morphisms will probably contain the class of fine morphisms, and therefore many of their properties will carry over to more general settings. By restricting our attention to fine morphisms we have narrowed down the initial learning task considerably, as the hypothesis space H is now the set of functions of type $\Sigma \to \{\varepsilon\} \cup \Gamma$, which is always finite (though usually very large) for fixed finite alphabets Σ and Γ. Moreover, since $\ln |H| = |\Sigma| \ln(1 + |\Gamma|)$ the sample complexity of this hypothesis space is polynomial.

Our conceptualization of learning is not limit-identification, but empirical risk minimization. Although empirical risk minimization is somewhat problematic

(Minka, 2000), particularly if the training data are not representative of the distribution of future data, it underlies most symbolic approaches to learning letter-to-sound rules, as well as many other NLP tasks. The empirical risk R of a hypothesis $h : \Sigma^* \to \Gamma^*$ is its average loss on a set of training samples $D \subseteq \Sigma^* \times \Gamma^*$, namely

$$R = \frac{1}{|D|} \sum_{\langle x,y \rangle \in D} L(h(x), y)$$

where $L : \Gamma^* \times \Gamma^* \to \mathbb{R}_{\geq 0}$ is the loss function. The most commonly used generally applicable loss functions for comparing strings are the zero-one loss

$$L_{\text{identity}}(y', y) = \begin{cases} 0 & \text{if } y' = y \\ 1 & \text{otherwise} \end{cases}$$

and $L_{\text{evenshtein}}$, the string edit distance (see for example Kruskal, 1983). Both kinds of loss play a role in the evaluation of letter-to-sound rules: for example, Damper et al. (1999, p. 164) use zero-one loss, and Fisher (1999) uses string edit distance. One generally requires that $L(y, y) = 0$, which is obviously the case for L_{identity}, and also holds for $L_{\text{evenshtein}}$ provided the cost for matching symbols is zero.

Empirical risk minimization under zero-one loss can mean one of two things: minimizing the total number of mistakes a hypothesis makes on the training data, or maximizing the number of correct predictions. These two notions are equivalent if optimal solutions can be found exactly, but differ for approximate solutions.[2]

9.2 Exact optimization

The problem of finding a function $f : \Sigma \to \{\varepsilon\} \cup \Gamma$ such that the empirical risk of f^* is minimal is fundamentally a combinatorial optimization problem. Like all such problems it can be stated formally in different ways (Papadimitriou and Steiglitz, 1998, p. 345f.): the optimization version asks for the optimal f for a given set of samples $D \subseteq \Sigma^* \times \Gamma^*$; the evaluation version asks for the total loss incurred on D by the optimal f^*; and the decision version asks whether there exists an f^* such that the total loss incurred by it on D is less than or equal to a given budget k. A solution to the optimization version could be used to construct an answer to the evaluation version, which in turn could be used to solve the decision version. Contrapositively, if the decision version is hard to solve, so are the other two versions.

[2] Suppose the true global optimum among 100 samples is 10 mistakes, and the optimum can be approximated within a ratio of 1.2. Approximate maximization would find a solution with at most $100 - 90/1.2 = 25$ mistakes, but approximate minimization yields a solution with at most $10 \times 1.2 = 12$ mistakes.

Because $L(y,y) = 0$ for the loss functions L considered here, there is a common subproblem of the decision version which is independent of the loss function used: the restricted decision version asks whether there exists an f^* such that the total loss incurred by it on D is exactly zero. We call this the consistency problem. Obviously, if the decision version can be solved efficiently, so can the consistency problem.

Before we can formally state this key problem underlying the learning task, we need another auxiliary definition:

Definition 2 (Graph of a relation). Given a relation $R : A \to B$ on sets, define $\#R$, the *graph of R*, to be the set $\{\langle a,b\rangle \in (A \times B) \mid aRb\}$.

The consistency problems for the two classes of morphisms are stated in a format similar to the one used by Garey and Johnson (1979). An answer to the questions asked by these problem would tell us whether a suitable morphism exists that perfectly fits the training data D.

Problem 9.1 (Very Fine Morphism Consistency – VFMC)
Instance: A finite (multi)set $D \subseteq \Sigma^* \times \Gamma^*$ of training samples.

Question: Does there exist a very fine morphism consistent with all elements of D, i. e., is there a function $f : \Sigma \to \Gamma$ such that $D \subseteq \#(f^*)$?

Problem 9.2 (Fine Morphism Consistency – FMC)
Instance: A finite (multi)set $D \subseteq \Sigma^* \times \Gamma^*$ of training samples.

Question: Does there exist a fine morphism consistent with all elements of D, i. e., is there a function $f : \Sigma \to \{\varepsilon\} \cup \Gamma$ such that $D \subseteq \#(f^*)$?

The size of an instance of one of these problems is the total length of all strings in the training dictionary D:

Definition 3 (Dictionary size). Define the size $\|D\|$ of a dictionary $D \subseteq \Sigma^* \times \Gamma^*$ as

$$\|D\| = \sum_{\langle x,y\rangle \in D} |x| + |y|$$

where $|x|$ is the length of string x.

Of the two problems formulated here, FMC is intuitively more difficult than VFMC, since one has to decide which input symbols are mapped to the empty string, or equivalently, how the output strings should be aligned relative to the inputs. This issue does not arise with VFMC, since only strings of equal length need to be considered (if D contained a pair of strings with different lengths, then no very

```
1:  {Input: instance D, certificate f}
2:  for all ⟨x,y⟩ ∈ D do
3:      a_1 ··· a_n ← x
4:      b_1 ··· b_m ← y
5:      j ← 1
6:      for i ← 1 to n do
7:          if f(a_i) ≠ ε then
8:              if j > m then
9:                  return false
10:             else if f(a_i) ≠ b_j then
11:                 return false
12:             else {f(a_i) matches b_j}
13:                 j ← j + 1
14:     if j ≠ m + 1 then
15:         return false
16: return true
```

Figure 9.1: Certificate verification algorithm for FMC.

fine morphism can be consistent with D). It will be shown that FMC is a complete problem for the complexity class **NP** (see for example Garey and Johnson, 1979). Membership of FMC in **NP** can be established straightforwardly:

Theorem 1. *Problem* FMC *has succinct certificates that can be verified in polynomial time.*

Proof: A certificate for FMC is a partial function $f : \Sigma \to \Gamma \cup \{\varepsilon\}$, which can be represented in space linear in $\|D\|$ (because f only needs to mention elements of Σ that occur in D). Verification amounts to applying f^* to each input string in D and comparing the results to the corresponding reference output. The verification procedure, shown in Figure 9.1, runs in linear time and logarithmic space. □

As an aside, note that problem VFMC for very fine morphisms can be solved efficiently in linear time and space by the following procedure: iterate over D, and for each input symbol σ set $f(\sigma) \leftarrow \gamma$, where γ is the output symbol aligned[3] with σ; then run the verification algorithm from Figure 9.1 on D and f, and return its answer.

NP-hardness of FMC is established by a reduction from 3SAT, the decision problem asking whether there is a satisfying truth assignment for a set of disjunc-

[3] A consistent very fine morphism can only exist if $|x| = |y|$ for all $\langle x,y\rangle \in D$, which means that x and y are aligned, in the sense that the nth symbol of x corresponds to the nth symbol of y.

tive clauses with at most three literals each. We first define the construction and then prove that it correctly preserves the structure of 3SAT.

Definition 4 (Boolean variable gadget). For any Boolean variable v, the set $\mathscr{V}(v)$ contains the following pairs (a_v and b_v are two new symbols dependent on v):

$$\langle a_v v \overline{v} b_v, FTF \rangle,$$
$$\langle a_v b_v, F \rangle.$$

Definition 5 (3SAT clause gadget). For any 3SAT clause C_i of the form $(l_{i1} \vee l_{i2} \vee l_{i3})$ (where each l_{ij} is a literal of the form v or $\overline{v}$) the set $\mathscr{C}(C_i)$ contains the following pairs (c_{ij}, d_{ij}, e_i and f_i for $1 \leq j \leq 3$ are eight new symbols dependent on i):

$$\langle c_{i1} l_{i1} d_{i1}, FT \rangle,$$
$$\langle c_{i2} l_{i2} d_{i2}, FT \rangle,$$
$$\langle c_{i3} l_{i3} d_{i3}, FT \rangle,$$
$$\langle d_{i1} d_{i2} d_{i3} e_i f_i, TT \rangle.$$

Definition 6 (Reduction from 3SAT). Given an instance $\varphi = \bigwedge_{i=1}^{n} C_i$ of 3SAT, define $\mathscr{D}(\varphi)$ as the collection $\bigcup_{i=1}^{n} \mathscr{C}(C_i) \cup \bigcup \{\mathscr{V}(v) \mid \text{variable } v \text{ occurs in } \varphi\}$.

Theorem 2. *The reduction from* 3SAT *to* FMC *can be computed in logarithmic space and creates an instance whose size is polynomial in the size of the original instance.*

Proof: The reduction $\mathscr{D}$, which can be made to run in linear time, builds a collection $\mathscr{D}(\varphi)$ with the following properties: let m be the number of distinct variables of φ (so $m \leq 3n$); then $\|\mathscr{D}(\varphi)\| = 10m + 22n \leq 52n$, $|\mathscr{D}(\varphi)| = 2m + 4n \leq 10n$, $|\Sigma| = 4m + 8n \leq 20n$, and $|\Gamma| = 2$. Only counters need to be stored for computing the reduction (in order to keep track of clauses and variables represented by integers), which requires logarithmic space. □

Theorem 3. *Problem* FMC *is* **NP***-hard.*

Proof: We show that $\varphi = \bigwedge_{i=1}^{n} C_i$ is satisfiable iff there exists a fine morphism f^* consistent with $\mathscr{D}(\varphi)$. It will be convenient to let V denote the set of distinct variables of φ.

($\Rightarrow$) Assume that φ is satisfiable, i. e., there exists a satisfying assignment $\tau : V \to \{T, F\}$. Incrementally define a fine morphism f^* consistent with $\mathscr{D}(\varphi)$ as

follows: for all $v \in V$, let $f(v) = \tau(v)$ and $f(\overline{v}) = \overline{\tau(v)}$. If $\tau(v) = T$, let $f(a_v) = F$ and $f(b_v) = \varepsilon$, which makes f^* consistent with $\mathscr{V}(v)$; otherwise, if $\tau(v) = F$, let $f(a_v) = \varepsilon$ and $f(b_v) = F$ to make f^* consistent with $\mathscr{V}(v)$. In either case f^* can be made consistent with $\mathscr{V}(v)$, and because a_v and b_v do not occur outside the gadget for v, f^* can be made consistent with all variable gadgets.

The fact that τ is a satisfying assignment means that in each clause C_i at least one literal is made true by τ. So f will map at most two d_{ij} in $\mathscr{C}(C_i)$ to T, and therefore the definition of f^* can always be extended to make it consistent with the fourth pair in $\mathscr{C}(C_i)$ and hence consistent with the entire clause gadget for C_i. Since all symbols in a clause gadget other than literals of φ occur only in that gadget, the definition of f^* can be extended to make it consistent with all gadgets and therefore consistent with $\mathscr{D}(\varphi)$. Hence there exists a consistent fine morphism f^* constructible from τ.

($\Leftarrow$) Conversely, assume that a fine morphism g consistent with $\mathscr{D}(\varphi)$ exists. Show that $g|_V$, i. e. g restricted to the variables of φ, is a satisfying truth assignment for φ. The morphism g being consistent with $\mathscr{D}(\varphi)$ means that g is consistent with all variable gadgets and all clause gadgets.

Pick any variable gadget $\mathscr{V}(v)$. Then, because of the second pair in $\mathscr{V}(v)$, g must map exactly one of a_v and b_v to F: if $g(a_v) = F$ then $g(b_v) = \varepsilon$, and for the first pair $g(v) = T$ and $g(\overline{v}) = F$; otherwise if $g(b_v) = F$ then $g(a_v) = \varepsilon$, $g(v) = F$, and $g(\overline{v}) = T$. Note in particular that $(g|_V)(v) \in \{T, F\}$, so $g|_V$ is formally a truth assignment.

Now pick any clause gadget $\mathscr{C}(C_i)$ and suppose that g maps no l_{ij} in $\mathscr{C}(C_i)$ to T. Then all d_{ij} in $\mathscr{C}(C_i)$ are mapped to T because of the first three pairs in that clause gadget. But this would make g inconsistent with the fourth pair, contradicting the assumption that g is consistent with all clause gadgets. So g must map at least one l_{ij} in $\mathscr{C}(C_i)$ to T, which means that $g|_V$ makes the clause C_i true, and is therefore a satisfying truth assignment for φ. □

The preceding three theorems together imply that the consistency problem FMC is **NP**-complete. The existence of efficient algorithms for solving FMC is therefore unlikely. Since FMC is a subproblem of empirical risk minimization, the decision version of this optimization problem is also **NP**-complete.[4]

The evaluation and optimization version of empirical risk minimization do not

[4] Strictly speaking, the previous discussion only establishes **NP**-hardness of the decision version. Showing membership in **NP** is straightforward, but requires separate proofs depending on which loss function is used. For zero-one loss only a few minor modifications to the certificate verification algorithm in Figure 9.1 are required, which now has to aggregate the number of mistakes and compare it to the budget k. For loss based on edit distance, using the standard dynamic programming algorithm (Kruskal, 1983) ensures that certificates can be verified in polynomial time.

seem to fall within the analogous class **FNP** of function problems. The reason for this is that an optimal solution f only certifies the existence of a feasible solution (namely f) within a certain budget k (namely the aggregate loss of f^* on the training data), but does not seem to provide enough information to verify in polynomial time that no better solution within a budget of $k-1$ can exist. It is doubtful whether there are any polynomial-length certificates of optimality. We conjecture that these problems are in fact $\mathbf{FP}^{\mathbf{NP}}$-complete, just like TSP (Papadimitriou, 1994).

9.3 Approximations and heuristics

Since the existence of exact efficient algorithms for solving the overall optimization problem is unlikely, one should consider the alternatives: approximate, heuristic, and/or inefficient algorithms.

Even for highly restricted problems the prospects are rather bleak. The optimization problem that maximizes empirical string-level classification accuracy (the dual of empirical zero-one loss, i. e. string-level classification error) for very fine morphisms will be called MAX-VFMC. It is far from clear whether MAX-VFMC is an easy or a hard problem, as we had shown earlier that VFMC can be solved very efficiently. We define the decision version of MAX-VFMC as follows:

Problem 9.3 (Very Fine Morphism Maximization – MAX-VFMC)
Instance: A finite sequence $D = \langle s_1, \ldots, s_n \rangle$ where each $s_i \in \bigcup_{j \in \mathbb{N}} \Sigma^j \times \Gamma^j$ for $1 \leq i \leq n$; and a natural number k with $k \leq n$.

Question: Does there exist a very fine morphism consistent with at least k elements of D, i. e., is there a function $f : \Sigma \to \Gamma$ and a length k unordered subsequence $\langle t_1, \ldots, t_k \rangle$ of D such that $t_i \in \#(f^*)$ for all $1 \leq i \leq k$?

We show that MAX-VFMC has probably (unless $\mathbf{P} = \mathbf{NP}$) no polynomial time approximation schemes (PTAS, which would allow us to find arbitrarily good approximations efficiently). In the best case, there may be an approximation algorithm for MAX-VFMC with a fixed approximation ratio, which would make MAX-VFMC a member of the class **APX** (Ausiello et al., 1999); whether or not this is the case is an open question.

Theorem 4. *Problem* MAX-VFMC *is* **APX***-hard.*

Proof: Show this by exhibiting an **AP**-reduction from an **APX**-complete problem. It suffices to show that MAX-k-CSP is L-reducible (Papadimitriou, 1994, 309ff.) to MAX-VFMC. MAX-k-CSP is a constraint satisfaction problem (Khanna et al., 1997) with conjunctive constraints containing at most k literals (see also Ausiello et al., 1999).

Given an instance $\varphi = \langle (l_{11} \wedge \cdots \wedge l_{1k}), \ldots, (l_{n1} \wedge \cdots \wedge l_{nk}) \rangle$ of MAX-k-CSP, construct an instance of MAX-VFMC by mapping the ith constraint $(l_{i1} \wedge \cdots \wedge l_{ik})$ to the pair $\langle l_{i1}\overline{l_{i1}} \ldots l_{ik}\overline{l_{ik}}, TF \ldots TF \rangle$ to form D (if a literal l is negative, i. e. of the form $\overline{v}$, then $\overline{l}$ is simply v). So Σ consist of the negated and unnegated variables of φ, and $\Gamma = \{T, F\}$. This construction ensures that there is a truth assignment τ that makes exactly m constraints of φ true iff there exists a very fine morphism f^* which is consistent with exactly m elements of D. One can construct f from τ (and vice versa) via $f(v) = \tau(v)$ and $f(\overline{v}) = \overline{\tau(v)}$ where v is a variable occurring in φ. □

Exact global optimization of MAX-VFMC is theoretically possible via branch-and-bound search. While this inefficient algorithm can be used for very small problem instances (learning English letter-to-sound rules with no conditioning context, for which only a few trillion morphisms have to be explored), it becomes intractable for even slightly larger problems (for English letter-to-sound rules conditioned on one letter of context there are more than one trequadragintillion feasible solutions). Heuristic algorithms, especially those based on local search (Papadimitriou and Steiglitz, 1998), are efficient and do in practice improve on greedily constructed initial solutions, but offer no performance guarantees.

9.4 Conclusions

We have reduced the problem of learning local transductions to the problem of learning morphisms on free monoids (the reduction may involve deterministic preprocessing of the training data). The restricted problem of deciding whether there exists a fine morphism consistent with a set of training samples was shown to be **NP**-complete. Since this problem is a specialization of the decision version of empirical risk minimization under any loss function L for which $L(y,y) = 0$, the larger optimization problems which generalize the consistency problem are generally intractable.

Acknowledgements

This paper is based, occasionally verbatim, on the author's dissertation (Jansche, to appear 2003), which contains further details. Many thanks to Chris Brew, Gerald Penn, and Richard Sproat, as well as two anonymous reviewers, for comments and feedback. The usual disclaimers apply.

Bibliography

Ausiello, G., P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi (1999). *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Berlin.

Damper, R. I., Y. Marchand, M. J. Adamson, and K. Gustafson (1999). Evaluating the pronunciation component of text-to-speech systems for English: A performance comparison of different approaches. *Computer Speech and Language*, **13**(2):155–176.

Eilenberg, S. (1974). *Automata, Languages, and Machines*, volume A. Academic Press, New York.

Fisher, W. M. (1999). A statistical text-to-phone function using ngrams and rules. In *International Conference on Acoustics, Speech, and Signal Processing*, pp. 649–652. Phoenix, AZ.

García, P. and E. Vidal (1990). Inference of *k*-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, **12**(9):920–925.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco.

Gildea, D. and D. Jurafsky (1996). Learning bias and phonological-rule induction. *Computational Linguistics*, **22**(4):497–530.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, **10**(5):447–474.

International Phonetic Association (1999). *Handbook of the International Phonetic Association: A Guide to the Use of the International Phonetic Alphabet*. Cambridge University Press, Cambridge, England.

Jansche, M. (to appear 2003). *Inference of String Mappings*. Ph.D. thesis, The Ohio State University, Columbus, OH.

Khanna, S., M. Sudan, and D. P. Williamson (1997). A complete classification of the approximability of maximization problems derived from Boolean constraint satisfaction. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pp. 11–20. El Paso, TX.

Kruskal, J. B. (1983). An overview of sequence comparison. In D. Sankoff and J. Kruskal, eds., *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pp. 1–44. Addison-Wesley, Reading, MA. Reissued by CSLI Publications, Stanford, CA, 1999.

McNaughton, R. and S. Papert (1972). *Counter-Free Automata*. MIT Press, Cambridge, MA.

Minka, T. P. (2000). Empirical risk minimization is an incomplete inductive principle. `http://www.stat.cmu.edu/~minka/papers/erm.html`.

Oncina, J., P. García, and E. Vidal (1993). Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, **15**(5):448–458.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley, Reading, MA.

Papadimitriou, C. H. and K. Steiglitz (1998). *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineola, NY. Originally published by Prentice Hall, Englewood Cliffs, NJ, 1982.

Roche, E. and Y. Schabes, eds. (1997). *Finite-State Language Processing*. Language, Speech and Communication. MIT Press, Cambridge, MA.

Sejnowski, T. J. and C. R. Rosenberg (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, **1**(1):145–168.

van den Bosch, A. P. J. (1997). *Learning to Pronounce Written Words: A Study in Inductive Language Learning*. Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands.

Weide, R. L. (1998). The Carnegie Mellon pronouncing dictionary version 0.6. Electronic document, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. `ftp://ftp.cs.cmu.edu/project/fgdata/dict/`.

Chapter 10

Querying Linguistic Treebanks with Monadic Second-Order Logic in Linear Time

STEPHAN KEPSER*

10.1 Introduction

In recent years large amounts of electronic texts have become available providing a new base for empirical studies in linguistics and offering a chance to linguists to compare their theories with large amounts of utterances from "the real world". While tagging with morphosyntactic categories has become a standard for almost all corpora, more and more of them are nowadays annotated with refined syntactic information. Examples are the Penn Treebank (Marcus et al., 1993) for American English annotated at the University of Pennsylvania, the French treebank (Abeillé and Clément, 1999) developed in Paris, the NEGRA Corpus (Brants et al., 1999) for German annotated at the University of Saarbrücken, the Tübingen Treebanks (Hinrichs et al., 2000) for Japanese, German and English from the University of Tübingen, and the German newspaper corpus TIGER (Brants et al., 2002). To make these rich syntactic annotations accessible for linguists the development of powerful query tools is an obvious need and has become an important task in computational linguistics.

Consequently, a number of query tools for syntactically annotated corpora have been developed in recent times. Amongst the most important ones are CorpusSearch (Randall, 2000), fsq (Kepser, 2003), ICECUP III (Wallis and Nelson, 2000), TGrep2 (Rohde, 2001), TIGERsearch (König and Lezius, 2000), and VIQTORYA (Kallmeyer and Steiner, 2002). All of them face a fundamental problem in the design of a query system namely the definition of the expressive power of the query language. The problem lies in balancing out user demands for a high

* SFB 441, University of Tübingen, Germany; kepser@sfs.uni-tuebingen.de

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

expressive power on the one hand and complexity problems on the other that may arise when query languages become quite powerful. The difficulty of this problem grows with the fact that the so-called treebanks to be queried are very often not just collections of proper trees. Demands of linguists have introduced additional features such as crossing branches and secondary relations. In consequence, some treebanks have more or less become collections of finite structures. Most of the above mentioned query tools (namely CorpusSearch, ICECUP III, TGrep2, and VIQTORYA) ignore this additional challenge completely, they are designed to query trees only. Still, they typically offer query languages of limited expressive power with the existential fragment of first-order logic being a kind of upper bound. A notable exception is fsq, which was particularly developed to query arbitrary finite first-order structures with full first-order logic. The disadvantage of fsq is the complexity of the implemented algorithm: Evaluation time of a query is polynomial in the size of the treebank. The size of the lead polynome is the quantifier depth of the query. Hence the evaluation of complex queries can take quite a long time.

Building on insights from theoretical informatics we show here that it is possible to query linguistic treebanks with a monadic second-order logic, powerful query language, in time linear in the size of the treebank.

It is known that the evaluation of a first-order sentence on a finite structure with a carrier of just two elements is already PSPACE-complete. To differentiate between the contribution of the logic or query language on the one hand and the contribution of the size of the finite structure on the other, Vardi (1982) introduced the notions of *query complexity* and *data complexity*. The general situation in querying linguistic treebanks is such that treebanks are large and still growing huger while most queries are relatively small. This justifies the concentration on data complexity, as we do it here.

10.2 The Query Language

The query language we propose is monadic second-order logic (MSO). It is the extension of first-order logic by set variables. As stated above, "trees" in a treebank may contain unconnected subparts and directed as well as undirected secondary edges. We therefore see a tree in a treebank as a finite relational structure. Technically, we follow the exposition by Arnborg, Lagergren, and Seese (1991). The signature of a tree consists of the unary predicate symbols $V, E, D, P_1, \ldots, P_p$ for some $p \in \mathbb{N}$ and of the three binary predicates R_1, R_2, R_3 with the intended meaning that

- V designates the set of vertices,
- E designates the set of undirected edges,

- D designates the set of directed edges,
- $R_1(a,b)$ holds if and only if a is a vertex incident with the edge b,
- $R_2(a,b)$ holds if and only if a is the source or origin of the directed edge b,
- $R_3(a,b)$ holds if and only if a is the target or end point of the directed edge b,
- $P_1,\ldots,P_p$ are (linguistic) labels of vertices and edges.

Let $\mathscr{V}_1 = \{x,y,z,x_1,x_2,x_3,\ldots\}$ and $\mathscr{V}_2 = \{X,Y,Z,X_1,X_2,X_3,\ldots\}$ be two disjoint denumerable sets of individual variables (vertices or edges) and set variables. We use lower case letters for individual variables and upper case letters for set variables. The syntax of MSO contains two binary logical relations, namely $=$ (equality) and $\in$ (membership). Formulae are defined as follows. For all $x,y \in V_1, X \in \mathscr{V}_2, 1 \leq j \leq p$: $V(x), E(x), D(x), P_j(x), R_1(x,y), R_2(x,y), R_3(x,y), x = y, x \in X$ are atomic formulae. Let φ and ψ be formulae. Then $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \rightarrow \psi, \exists x\varphi, \forall x\varphi, \exists X\varphi, \forall X\varphi$ are formulae.

A tree is a a finite relational structure $\mathscr{T} = (U,V,E,D,R_1,R_2,R_3,P_1,\ldots,P_p)$ where U is a finite nonempty set of vertices and edges, V,E,D are unary predicates of vertices and (undirected and directed) edges, $P_1,\ldots P_p$ are unary predicates, and R_1,R_2,R_3 are binary predicates. We call such a structure also a graph. The size of the graph is $|U|$, the number of vertices and edges of the graph.

The semantics of MSO over theses structures is an extension of the classical first-order logic semantics. A variable assignment a now consists of two functions $a_1 : \mathscr{V}_1 \rightarrow U$ and $a_2 : \mathscr{V}_2 \rightarrow \wp(U)$. Formulae not involving set variables have the same semantics as in the first-order case. The formula $x \in X$ is true iff $a(x) \in a(X)$. For set quantification, $\exists X\varphi$ is true in $(\mathscr{T},a)$ iff there is a $W \subseteq U$ such that φ is true in $(\mathscr{T},a[X/W])$ where $a[X/W]$ is a variable assignment that is equal to a except that it assigns W to X. The formula $\forall X\varphi$ is true in $(\mathscr{T},a)$ iff for all $W \subseteq U$ the formula φ is true in $(\mathscr{T},a[X/W])$.

Not every finite structure of the given signature is suitable to designate a tree. To ensure the intended meaning of the predicates some axioms have to be added that all structures should respect.

$$\forall x\, V(x) \vee E(x) \vee D(x), \qquad \forall x\, \neg(V(x) \wedge E(x)),$$
$$\forall x\, \neg(E(x) \wedge D(x)), \qquad \forall x\, \neg(V(x) \wedge D(x)),$$
$$\forall x,y\, R_1(x,y) \rightarrow (V(x) \wedge E(y)), \qquad \forall x,y\, R_2(x,y) \rightarrow (V(x) \wedge D(y)),$$
$$\forall x,y\, R_3(x,y) \rightarrow (V(x) \wedge D(y)).$$

The axioms state that the relations V,E and D partition the domain U and that the left hand side argument of a relation $R_{1,2,3}$ is always a vertex while the right hand side is an undirected edge for R_1 and a directed edge for $R_{2,3}$.

A linguistic treebank in our sense is a finite set of finite structures defined as above. Restricting ourselves to finite treebanks is justified on two grounds. Firstly, any now or in the future existing treebank is finite. Secondly, querying an infinite treebank makes no sense since a person posing a query expects an answer at least in finite time.

10.3 Linear Time Complexity of MSO Queries

In the general case, the data complexity of MSO queries on arbitrary classes of finite structures is PSPACE (see, e.g., Ebbinghaus and Flum (1995)). Thus there is little hope to find efficient algorithms for MSO queries on arbitrary classes of finite structures. But there is a class of structures for which a linear-time algorithm exists. As was shown independently by Arnborg, Lagergren, and Seese (1991) and by Courcelle (1990a,b, 1992), the class of graphs *with bounded treewidth* possesses such an algorithm.

The notion of *treewidth* was introduced by Robertson and Seymour (1986) as a way to measure how close to a tree a graph is. Bodlaender (1993) provided a general introduction that the interested reader is referred to.

Definition 1. A *tree decomposition* of a graph $(A, V, E, D, R_1, R_2, R_3, P_1, \ldots, P_p)$ is a pair (T,S), where T is a tree and S is a family of sets indexed by the vertices of T such that

1. $\bigcup_{X_t \in S} X_t = A$.
2. For all $c \in A$ such that $E(c)$ there is a unique $X_t \in S$ such that $c \in X_t$, and if $a \in A$ satisfies $R_1(a,c)$ then $a \in X_t$.
3. For all $c \in A$ such that $D(c)$ there is a unique $X_t \in S$ such that $c \in X_t$, and if $a \in A$ satisfies $R_2(a,c)$ or $R_3(a,c)$ then $a \in X_t$.
4. For all $a \in A$ the subgraph of T induced by $\{t \mid a \in X_t\}$ is connected.

The *width* of such a decomposition is $\max_{X_t \in S} |\{a \mid a \in X_t, V(a)\}| - 1$, i.e., the largest number of vertices in a single set of the decomposition minus 1.
A graph G is of *treewidth* k if and only if the smallest width of a tree decomposition of G is k.

There are different ways in which a graph can divert from a tree. A clique, for example, is a structure which is kind of an opposite of a tree. Hence it is simple to see that the size of the largest clique is a lower bound of the treewidth of a graph. An important property of a tree is that every pair of vertices of a tree is connected by a *unique* path. A graph in which many pairs of vertices are connected by many

different independent paths is therefore also kind of an opposite of a tree. The largest number of independent paths between two vertices gives an upper bound for the treewidth.

Proposition 1. Arnborg et al. (1991); Courcelle (1990a,b, 1992) *For every class K of graphs of universally bounded treewidth, every MSO sentence can be decided in time linear in the size of G for* $G \in K$.

A fortiori, every finite set of graphs has a bounded treewidth.

Corollary 2. *Therefore MSO queries on linguistic treebank can be evaluated in linear time in the size of the treebank.*

The above results were enhanced by several authors showing that MSO can be extended by cardinality predicates or simple counting. The perhaps most general result is given by Courcelle and Mosbah (1993) who add a certain type of evaluation functions to MSO.

The core of these results is achieved by a reduction to classical formal language theory. Using a method of semantic interpretation of one structure in another proposed by Rabin (1977), Arnborg et al. (1991), and Courcelle (1990a,b, 1992) provide a method for interpreting finite graphs of bounded treewidth by finite trees. MSO sentences can then be evaluated over these trees using tree automata techniques proposed by Doner (1970) and Thatcher and Wright (1968).

A bit more detailed, Arnborg, Lagergren, and Seese (1991) provide a general construction by which an MSO sentence S on graphs is translated into an MSO sentence $\tau(S)$ on binary trees. This construction also transforms a general labelled graph G with a suitable tree decomposition into a labelled binary tree $T(G)$ in time linear in the number of vertices of G in such a way that S holds in G if and only if $\tau(S)$ holds in $T(G)$. The step of the computation of a suitable tree decomposition can be done also in linear time on graphs with bounded treewidth, as was shown by Bodlaender (1996).[1] After the application of this transformation, classical tree automata techniques (Doner, 1970; Thatcher and Wright, 1968) can be applied.

The computation of a tree decomposition, although possible in time linear in the size of graph, is an expensive step. In a recent analysis of the original algorithm by Bodlaender (1996), Hagerup (2002) presents a variant which works three orders of magnitude faster. Still the algorithm is exponential in the square of the treewidth. This seems to indicate that it can hardly be used in practice. But there are two important facts to keep in mind that make this approach feasible. Firstly, most trees in current treebanks have a small treewidth. The capabilities of secondary

[1] Remarkably, Bodlaender found his result after the publication of the works by Arnborg et al. (1991) and Courcelle (1990a,b).

relations are only sparsely used by annotators. To give an extreme example of how rarely secondary edges are used, consider the German Tübingen Treebank (Hinrichs et al., 2000), in which more than 99.9% of the trees have treewidth 1, the small rest having treewidth 2. Secondly, and more importantly, the computation of a tree decomposition has to be done only once. It is a part of the preprocessing step that transforms tree-like graphs of the input treebank into proper trees suitable for the application of tree automata techniques. Obviously, such a preprocessing is performed once and off-line. As such, it is not a relevant factor in the actual query response time, i.e., the time from the posing of a query till the presentation of the answer. Therefore, longer preprocessing times are indeed tolerable.

10.4 On the Expressive Power of MSO Queries

As stated in the introduction, the development of query systems that employ powerful query languages is a relatively new one. An important reason therefore is certainly the fact that corpora with rich syntactic annotations came to exist only recently. And only if there is a rich structure to query it makes sense to provide powerful query languages.

On the other hand, there is now a growing need for powerful query languages for the following reasons. Suppose a linguist is interested in finding a particular syntactic phenomenon in a large treebank. In most query languages it is a trivial task to write a query the answer set to which will contain all instances of the phenomenon that can be found in the corpus. Just write a query which is rather general. The answer set will be big and certainly contain what the linguist is looking for. But it will mainly consists of undesirable "garbage", trees that do not exhibit the phenomenon sought. Hence, the real task in querying is not so much to produce an answer set that contains instances of what you are searching for. The task is rather to weed out the garbage, to keep answer sets as small as possible. Looking at things this way, a query is a kind of a filter for the corpus. And in order to retain small answer sets it is necessary to make that filter strong. A linguist should be able to spell out as exactly as possible the phenomenon he is looking for. And that requires powerful query languages. Treebanks have already gained quite a size, e.g., the German Tübingen Treebank contains more than 38.000 trees. There is hardly any chance to manually check big answer sets any more.

Let us illustrate these arguments by linguistically motivated examples. Suppose we are looking for trees in a German or English treebank where a clause lacks the subject. It is known that Germanic languages require the subject to be lexically realised under normal circumstances. It is therefore interesting to see whether there are any exceptions from this rule, and if, what they look like. An example of

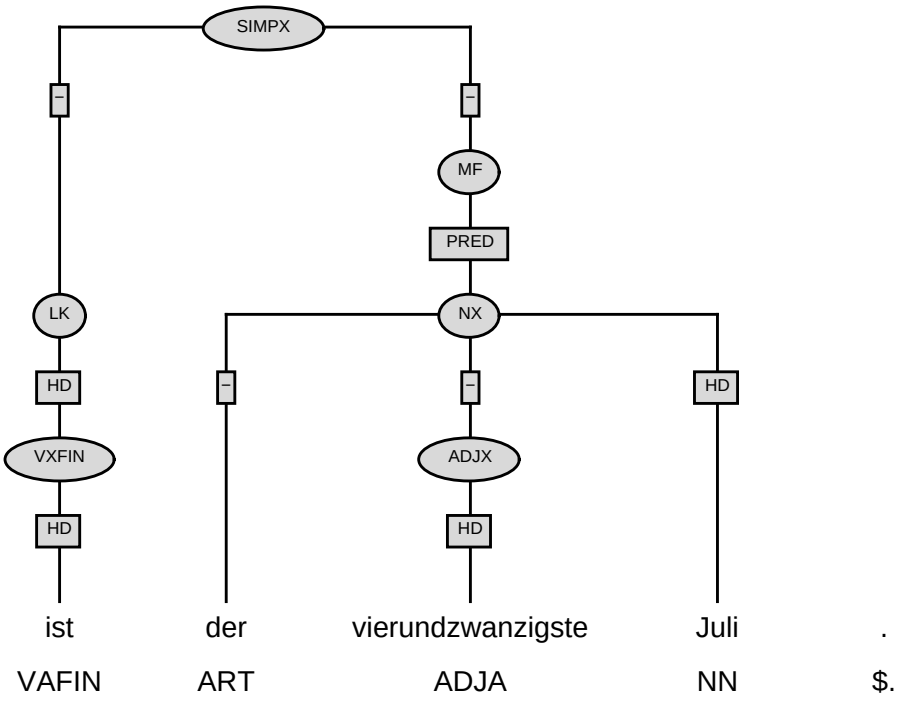


Figure 10.1: A subject-free clause from the German Tübingen Treebank

a clause where the subject is missing can be seen in Figure 10.1, which displays a tree from the German Tübingen Treebank. It reads *is the twenty fourth of July*. Without going into details of the annotation we note that the grey shaded ellipses represent part-of-speech tags or syntactic categories and the grey shaded rectangles represent grammatical functions or edge labels.

In order to find trees that lack the subject, we have to find a clause node, which is a node of category (POS tag) `SIMPX` below which there is no subject. In this treebank, a subject is a node of grammatical function `ON` which stands for *Object in the Nominative*. The treebank contains a parent relation to indicate the tree structure. In our queries, we will use the dominance relation, the reflexive transitive closure of the parent relation, and designate it by the relation symbol $>>$. A discussion of transitive closures in queries follows the present illustration of searching for subject-free clause.

We can now pose the following query:

$\exists x \mathrm{SIMPX}(x) \wedge \forall y((x >> y) \rightarrow \neg \mathrm{ON}(y))$

The formula reads "*There is a clause node (node of category* SIMPX*) such that no*

node below it is a subject node (node of function ON *(Object in the Nominative)).*"

An example result is the tree in Figure 10.1. Another result from the same corpus would be "*aber gut, wir können ja mal fragen, was gegeben wird.*" (*All right, we can ask, what's on play.*) where there is no subject in the German subordinate clause. If one is interested in finding only those trees where the subject is lacking in a subordinate clause, the above query has to be extended to

$\exists x \exists y\, \mathrm{SIMPX}(x) \wedge \mathrm{SIMPX}(y) \wedge (x >> y) \wedge (x \neq y) \wedge (\forall z \neg((y >> z) \wedge \mathrm{ON}(z)))$

"*There are two different clause nodes, one dominating the other, and no node below the lower clause node is a subject node.*"

This is a query of quantifier depth 3 (number of deepest nestings of quantifiers). On second thought one can see that this query is still too simple to find all subordinate clauses without subject. It does not reflect the possibility of having a subordinate clause with subject as a subclause of a subordinate clause without subject. Here is a query that does:

$$\begin{aligned}\exists x \exists y\ & \mathrm{SIMPX}(x) \wedge \mathrm{SIMPX}(y) \wedge (x >> y) \wedge (x \neq y) \wedge \\ & (\forall z\, \mathrm{ON}(z) \rightarrow\ \ (\neg(y >> z) \vee \\ & \qquad \exists w\, \mathrm{SIMPX}(w) \wedge (y >> w) \wedge (y \neq w) \wedge (w >> z)))\end{aligned}$$

"There are two different clause nodes, one dominating the other, and every subject node is either not dominated by the lower clause node or there is a further clause node intervening."

This query is even of quantifier depth 4. Another complicated query must be used if we want to find all trees in which the main clause lacks the subject, but subordinate clauses may have one. The query looks like this:

$$\begin{aligned}\exists x\ & \mathrm{SIMPX}(x) \wedge (\forall y\, \mathrm{SIMPX}(y) \rightarrow (x >> y \wedge x \neq y)) \wedge \\ & (\forall y\ \ ((x >> y) \wedge \mathrm{ON}(y)) \rightarrow \\ & \qquad \exists z\, (\mathrm{SIMPX}(z) \wedge (x >> z) \wedge (x \neq y) \wedge (z >> y)))\end{aligned}$$

"*There is a highest clause node such that for every subject node dominated by it there is a second clause node intervening.*"

These examples show that once a linguist is interested in more advanced phenomena a powerful query language is necessary to specify as closely as possible what it is that the linguist seeks.

One of the advantages of MSO as a query languages is the fact, shown by Courcelle (1990a), that the transitive closure of any MSO-definable relation is also MSO-definable. Transitive closures play an important role in formal definitions of linguistic structures. Although the term is rarely literally used, many definitions contain it tacitly. One such example is the definition of dominance as given in the above discussion. Another example is the lexical head of a phrase. Here we look at the transitive closure of the head-daughter relation. Any notion of government, c-command or barriers contains indirectly a transitive closure, as well as notions of maximal or minimal categories. Since it is unforeseeable which particular variant

of these notions a user querying a corpus has in mind, it is not a feasible approach to try to precompile these transitive closures during the preprocessing of the corpus. To provide a query language that allows the definition of transitive closures seems to be the more promising way.

There is a second field of applications of powerful queries, namely in the *development* of treebanks. As was pointed out by Dickinson and Meurers (2003), even treebanks that are annotated by hand and not automatically can contain quite a number of misannotations and inconsistencies. To enhance the quality of the treebank an annotator can check wether his annotations are consistant by defining the environment of an annotation, querying the treebank with this definition, and inspecting if the annotations in the answer set are the way they should be. The expressive power of the query language is important for an annotator because fine-grained annotations are typically very sensitive to environments, and thus environments should be definable rather precisely.

10.5 Conclusion

In this paper we showed that linguistic treebanks can be queried with a very powerful query language, namely monadic second-order logic, in time linear in the size of the treebanks. We thus give an argument for that at least on a theoretical level the question of a choice of a query language for treebanks can be settled. We hardly expect the arise of a need of an even more powerful query language. And the fact that a large part of costly computations can be done in an offline preprocessing step to be performed only once lets us believe that the described approach is practically feasible.

It would certainly be nice, if one would be able to show that the types of finite structures one can find in linguistic treebanks are such that they have a bounded treewidth by their nature. But at least some of the corpus formats currently being used do not as such warrant a bound for the treewidth of its instances. A simple example is the addition of free indexation to syntax trees in GB theory such as coindexing anaphora and antecedent or moved constituents and their traces. If there is no bound on the number of coindexations, the structures have an unbounded treewidth. An inspection of some of the available treebanks reveals on the other hand, that typically only a subset of the capabilities provided by the corpus formalism is actually in use. We thus think it is an intersting research goal to see if one can find an abstract characterisation of linguistic trees as found in treebanks that is general enough to cover most existing corpora but also that specific that it provides boundedness of the treewidth of its instance structures.

Acknowledgements

This research was funded by a grant of the German Science Foundation (DFG SFB 441-2). I would like to thank Uwe Mönnich for interesting and helpful discussions.

Bibliography

Abeillé, A. and L. Clément (1999). A tagged reference corpus for French. In *Proceedings of EACL-LINC*.

Arnborg, S., J. Lagergren, and D. Seese (1991). Easy problems for tree-decomposable graphs. *Journal of Algorithms*, **12**:308–340.

Bodlaender, H. L. (1993). A tourist guide through treewidth. *Acta Cybernetica*, **11**:1–23.

Bodlaender, H. L. (1996). A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, **25**:1305–1317.

Brants, S., S. Dipper, S. Hansen, W. Lezius, and G. Smith (2002). The TIGER Treebank. In K. Simov, ed., *Proceedings of the Workshop on Treebanks and Linguistic Theories*. Sozopol.

Brants, T., W. Skut, and H. Uszkoreit (1999). Syntactic annotation of a German newspaper corpus. In *Proceedings of the ATALA Treebank Workshop*, pp. 69–76.

Courcelle, B. (1990a). Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, volume B, chapter 5, pp. 193–242. Elsevier.

Courcelle, B. (1990b). The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, **85**:12–75.

Courcelle, B. (1992). The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues. *Informatique Théoretique et Applications*, **26**:257–286.

Courcelle, B. and M. Mosbah (1993). Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, **109**:49–82.

Dickinson, M. and D. Meurers (2003). Detecting errors in part-of-speech annotations. In A. Copestake and J. Hajič, eds., *Proceedings EACL 2003*, pp. 107–114.

Doner, J. (1970). Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, **4**:406–451.

Ebbinghaus, H.-D. and J. Flum (1995). *Finite Model Theory*. Springer-Verlag.

Hagerup, T. (2002). Simpler and faster tree decomposition. Manuscript, University of Frankfurt a. M.

Hinrichs, E., J. Bartels, Y. Kawata, V. Kordoni, and H. Telljohann (2000). The VERBMOBIL treebanks. In *Proceedings of KONVENS 2000.*

Kallmeyer, L. and I. Steiner (2002). Querying treebanks of spontaneous speech with VIQTORYA. *Traitement Automatique des Langues*, **43**(3):155–179.

Kepser, S. (2003). Finite Structure Query: A tool for querying syntactically annotated corpora. In A. Copestake and J. Hajič, eds., *Proceedings EACL 2003*, pp. 179–186.

König, E. and W. Lezius (2000). A description language for syntactically annotated corpora. In *Proceedings of the COLING Conference*, pp. 1056–1060.

Marcus, M., B. Santorini, and M. A. Marcinkiewicz (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, **19**(2):313–330.

Rabin, M. (1977). Decidable theories. In J. Barwise, ed., *Handbook of Mathematical Logic*, pp. 595–629. North-Holland.

Randall, B. (2000). CorpusSearch user's manual. Technical report, University of Pennsylvania. `http://www.ling.upenn.edu/mideng/ppcme2dir/`.

Robertson, N. and P. Seymour (1986). Graph minors II. Algorithmic aspects of treewidth. *Journal of Algorithms*, **7**(309–322).

Rohde, D. (2001). Tgrep2. Technical report, Carnegie Mellon University. `http://tedlab.mit.edu/~dr/Tgrep2/`.

Thatcher, J. and J. Wright (1968). Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, **2**(1):57–81.

Vardi, M. (1982). The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pp. 137–146.

Wallis, S. and G. Nelson (2000). Exploiting fuzzy tree fragment queries in the investigation of parsed corpora. *Literary and Linguistic Computing*, **15**(3):339–361.

Chapter 11

Some remarks on arbitrary multiple pattern interpretations

C. MARTÍN-VIDE* AND V. MITRANA*

ABSTRACT. A word w is obtained by an arbitrary n-pattern interpretation of a word x if there are n homomorphisms $h_1, h_2, \ldots, h_n$ and a positive integer k such that $w = h_{i_1}(x)h_{i_2}(x)\cdots h_{i_k}(x)$ with $1 \leq i_j \leq n$ for all $1 \leq j \leq k$. This arbitrary multiple pattern interpretation of words is naturally extended to languages. We investigate some closure properties of the families of languages obtained by arbitrary multiple pattern interpretations of finite, regular, and context-free languages, respectively. We show that the first of these families forms an infinite hierarchy and give a characterization of the arbitrary multiple pattern interpretation of finite languages. Two concepts of ambiguity and inherent ambiguity of multiple pattern interpretation are defined. It is shown that both properties are decidable for multiple pattern interpretations on finite languages but strong ambiguity is not decidable for multiple pattern interpretations on the class of context-free languages. The paper also contains a series of open problems.

11.1 Introduction

In Angluin (1980), a new way for defining a language is considered. Instead of identifying completely a language by generative devices as formal grammars or by recognition devices as automata, sometimes it is useful to consider less strict definitions. In the aforementioned work, the notion of *pattern* is defined as a word containing variables and constants, and then the language defined by a pattern α consists of all words obtained from α by substituting a string of constants for each variable. The substitution has to be uniform in the sense that the multiple occurrences of a variable must be replaced with the same string.

In the seminal work of Angluin 1980 the variables have to be replaced with nonempty strings, while in Jiang et al. (1994) substitutions by empty strings are allowed, which makes an essential difference. In Restivo and Salemi (2002), one

*Research Group in Mathematical Linguistics, University Rovira i Virgili, Tarragona, Spain; cmv@correu.urv.es, vmi@fll.urv.es

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

proposes a generalization of this definition: the distinction between variables and constants is discarded. Given two strings x and w, possibly over the same alphabet, x is a *pattern description* of a string w (w is obtained by a *pattern interpretation* of x) if w is the homomorphic image of x: in other words, there exists a homomorphism h such that $w = h(x)$.

Programming languages can be viewed as pattern interpretations of some languages. For instance, the main non-context-free features of programming languages are the necessity to define labels and the necessity to declare identifiers. The necessity of defining labels can be expressed by pattern interpretation in the following way. A correct program containing labels should be the pattern interpretation of the following general description:

$$(part\_pro)_1 (label)\ :\ (statement)\ (part\_pro)_2\ goto\ (label)\ (part\_pro)_3,$$

where *(label)*, *(statement)* and $(part\_pro)_i$, $i = 1,2,3$, are variables representing labels, statements or other parts of a program, respectively. By interpreting this pattern, we have to substitute the two occurrences of the variable $(label)$ by the same string of constants, hence observing the semantic restriction regarding labels definition.

In Kudlek et al. (2003) we propose a new pattern interpretation, namely *multiple pattern interpretation*. A word w is obtained by an arbitrary n-pattern interpretation of a word x if there are some homomorphisms $h_1, h_2, \ldots, h_n$, and $k \geq 1$, such that $w = h_{i_1}(x)h_{i_2}(x)\cdots h_{i_k}(x)$, $1 \leq i_j \leq n$ for all $1 \leq j \leq k$. This arbitrary multiple pattern interpretation of words is naturally extended to languages, namely a language L is the arbitrary multiple pattern interpretation of another language E if L contains all words which are obtained by the same arbitrary multiple pattern interpretation of the words in E.

Multiple pattern interpretations seem to be of basic concern for linguists. Indeed, one may say that each sentence follows a pattern which is an element of a finite or infinite set, that is a language.

Let us first consider the following aspect of language acquisition: two-word utterances in the speech of a two-year old child, in accordance with Owens (2001). To understand them, linguists proposed several strategies actually based on ordered or arbitrary multiple pattern interpretations. First, some words are used without any positional consistency (*agent+action, action+object, agent+object*): the so-called *grouping pattern* in Brown and Leonard (1986). For example, the child may say "*Eat cookie*" or "*Cookie eat*". Secondly, the utterance is characterized by a consistent word order which reflects patterns heard in adult speech: the so-called *positional associative pattern*, see Braine (1976). A third strategy, called *positional productive pattern* (Braine (1976)), is characterized by consistent word order and creative combinations. That is, children hypothesize a mini-language of patterns

(*attribute+entity, possessor+possession, demonstrative+entity, agent+action*, etc.) and then interpret these patterns by words repeatedly heard in specific locations in adult speech. It has been suggested that positional rules rather than semantic rules are the basis for early multiword utterances (Pine and Lieven (1993)). This strategy applies to adult speech as well, especially for nonnative speakers. For instance, a part of a speech can be constructed by an ordered interpretation of the word:

article+adjective+noun+verb+pronoun+noun+adverb.

By interpreting this word through a two-pattern interpretation, one gets the sentences:

The young man ate his hamburger quickly.
A mad racer drove his car recklessly.

On the other hand, syntactic theory is concerned, unlike traditional grammar, not with just describing specific languages but also with developing a general, universal theory. According to Borsley (1999), this means that other languages are always potentially relevant when one is describing a particular language. Thus, following Chomsky (1965, 1975); Kolb and Mönnich (1999), there exists a nontrivial set of axioms and a learnable extension of it that specify a possible natural language, and every natural language has a theory which is a learnable extension of the initial set. One has to determine a set of primitive blocks, operations which act on these blocks, an initial set and a learning procedure which maps the (primitive blocks of the) initial set onto the (utterances of the) steady state.

Furthermore, the *principles-and-parameters-model* discussed in Vogler (1999) has been established as a grammar formalism, based on the GB theory (Chomsky (1981, 1986)), aimed at describing the syntactical knowledge in a way that gives answers to questions concerning language acquisition and universal properties of languages. Thus, one may assume that the kernel of GB theory consists of a set of principles (= wellformedness conditions) and a way of interpreting them. In this respect, our multiple pattern interpretation of a language may be viewed as a particular case of such a general model.

11.2 Basic definitions

Let V and U be two alphabets. For a given integer $n \geq 1$, we denote by $\Omega_{n,V,U}$ an n-tuple $(h_1, h_2, \ldots, h_n)$ of homomorphisms from V^* to U^*, and call it an *n-pattern interpretation*. The subscripts indicating the alphabets will be omitted when the two alphabets are self-understood. A multiple pattern interpretation is said to be *non-erasing* if all its components are non-erasing homomorphisms. For the rest

of this paper, if not otherwise stated, all multiple pattern interpretations are non-erasing ones.

The *arbitrary multiple pattern interpretation* of $L \subseteq V^*$ through $\Omega_{n,V,U}$ is the language:

$$\begin{aligned}\Omega^*_{n,V,U}(L) &= \{h_{i_1}(w)h_{i_2}(w)\cdots h_{i_r}(w) \mid w \in L, r \geq 1, 1 \leq i_j \leq n, \\ &\quad \text{for all } 1 \leq j \leq r\}.\end{aligned}$$

If $n = 1$, hence Ω consists of a single homomorphism h from V^* to U^*, we write $h^*_{V,U}(L)$, or $h^*(L)$ provided that the alphabets V and U are self-understood from the context.

A homomorphism $h : V^* \longrightarrow U^*$ is termed a *letter-to-letter homomorphism* if $h(a) \in U$ for any $a \in V$. A multiple pattern interpretation whose all components are letter-to letter homomorphism is called a multiple pattern letter-to-letter interpretation. The following families of languages are defined:

$$\begin{aligned}\mathbf{HOM}^*_n(\mathbf{X}) &= \{\Omega^*_{n,V,U}(L) \mid \text{ for some } n\text{-pattern interpretation } \Omega_{n,V,U} \\ &\quad \text{and } L \in \mathbf{X}\}, \\ \mathbf{LHOM}^*_n(\mathbf{X}) &= \{\Omega^*_{n,V,U}(L) \mid \text{ for some } n\text{-pattern letter-to-letter} \\ &\quad \text{interpretation } \Omega_{n,V,U} \text{ and } L \in \mathbf{X}\},\end{aligned}$$

where $\mathbf{X} \in \{\mathbf{FIN}, \mathbf{REG}, \mathbf{CF}\}$. Here **FIN**, **REG**, **CF**, stand for the families of finite, regular, and context-free languages, respectively.

The above definition of a multiple pattern interpretation remembers the definition of a DT0L scheme, a very well-known type of Lindenmayer system. A DT0L scheme may be viewed as a multiple pattern interpretation $\Omega_{n,V,V}$, hence the n homomorphisms are actually endomorphisms on V^*. However, the way of interpreting a word through a DT0L scheme is different. A word w is obtained by a the DT0L scheme $\Omega_{n,V,V}$ interpretation of a word x if there exists a positive integer k such that $w = h_{i_1} \circ h_{i_2} \circ \cdots \circ h_{i_k}(x)$, $1 \leq i_j \leq n$, $1 \leq j \leq k$. Note the main difference: an arbitrary multiple pattern interpretation of a word is defined by a concatenation of the homomorphic images of that word while the interpretation through a DT0L scheme is a composition of the homomorphic images of that word. For more details about Lindenmayer systems the reader is referred to Rozenberg and Salomaa (1980). This makes an essential difference with respect to the families of languages obtained by these interpretation, namely the two families are incomparable. Indeed, the language $\{a^{3^n} \mid n \geq 0\}$ can be obtained by interpreting the singleton set $\{a\}$ through the DT0L scheme formed by the homomorphism $h(a) = aaa$, but it cannot be the arbitrary multiple pattern interpretation of any language since $2 \cdot 3^n$ cannot be written as a power of 3. On the other hand, the regular

language $R = \{a^{2n} \mid n \geq 1\}$ is the arbitrary 1-pattern interpretation of the same singleton set $\{a\}$ but it cannot be obtained by interpreting any finite language through a DT0L scheme. Assume the contrary, namely R is obtained by interpreting a finite language through the DT0L scheme $\Omega_n = (h_1, h_2, \ldots, h_n)$. Since a^{2p} with p an arbitrarily large prime number is in R, it follows that $a^{2p} = h_i(a^{2k})$ for some $1 \leq i \leq n$ and $k \geq 1$. This implies that $k = 1$ and $h_i(a) = a^p$. The contradiction follows from the fact that there are many prime numbers.

11.3 Some properties of the languages obtained by arbitrary multiple pattern interpretations

Clearly, for any alphabet $V = \{a_1, a_2, \ldots, a_n\}$ we have $V^* = \Omega^*_{n,\{a\},V}(\{a\})$, where $\Omega_{n,\{a\},V} = (h_1, h_2, \ldots, h_n)$, each h_i being defined by $h_i(a) = a_i$. It is worth mentioning here a similar fact observed for pattern descriptions (Restivo and Salemi (2002)), namely the "worst" description of any word w (in the sense that this description gives the least information about the structure of w) is the word a. On the other hand, it is easy to note that any language in $\mathbf{HOM}^*_n(\mathbf{FIN})$ is either $\{\varepsilon\}$ or infinite.

Note that if L is a finite language, then any language obtained by an arbitrary multiple pattern interpretation of L is a regular language. Indeed, for any $\Omega_n = (h_1, h_2, \ldots, h_n)$

$$\Omega^*_n(L) = \bigcup_{w \in L} \{h_1(w), h_2(w), \ldots, h_n(w)\}^+ \qquad (*)$$

holds. However, there are regular languages that are not the arbitrary multiple pattern interpretation of any language, finite or not. Such a language is a^+b^+. This follows immediately from a simple observation: if a word w lies in a language L defined by a multiple pattern interpretation of an arbitrary language, then ww must lie in L, too. Therefore, the following problem naturally arises: Is it decidable whether or not a given a context-free (regular) language is the arbitrary multiple pattern interpretation of a finite language?

First, we give a characterization of regular languages that can be obtained by an arbitrary multiple pattern interpretation of a finite language.

Proposition 11.3.1. *A regular language is the arbitrary multiple pattern interpretation of a finite language if and only if it is a finite union of finitely generated semigroups w.r.t. concatenation.*

Proof : Clearly, if any arbitrary multiple pattern interpretation of a finite language is a finite union of finitely generated semigroups w.r.t. concatenation, see (*) above.

Assume now that $L = \bigcup_{i=1}^{k} F_i^+ \subseteq U^*$ for some $k \geq 1$ and finite sets $F_1, F_2, \ldots, F_k$. Let $p = \max\{\text{card}(F_i) \mid 1 \leq i \leq k\}$. It is plain that for each $1 \leq i \leq k$, one can find F_i' such that $\text{card}(F_i') = p$ and $(F_i')^+ = F_i^+$. Suppose that $F_i' = \{x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)}\}$, $1 \leq i \leq k$. We define the alphabet $V = \{a_1, a_2, \ldots, a_k\}$ and the homomorphisms $h_j : V^* \longrightarrow U^*$, $1 \leq j \leq p$, $h_j(a_i) = x_j^{(i)}$. The equality $\Omega_p^*(V) = L$, where $\Omega_p = (h_1, h_2, \ldots, h_p)$, concludes the proof. □

Now the aforementioned problem can be reformulated as follows: Is it decidable whether or not a given regular language can be written as a finite union of finitely generated semigroups w.r.t. concatenation? Despite the problem seems to be "classic", we were not able to find any result regarding this matter either to solve it. The problem is likely decidable for subclasses of regular languages, like slender regular languages (Păun and Salomaa (1995)) but the general case remains *open*.

For the class of context-free languages the problem was solved in Kudlek et al. (2003) by a usual reduction to the Post Correspondence Problem:

Theorem 11.3.1. *Is it undecidable whether or not a given a context-free language is the arbitrary multiple pattern interpretation of a finite language.*

It is worth mentioning that there are non-context-free languages in $\mathbf{LHOM}_n^*(\mathbf{REG})$, for any $n \geq 1$. However, for each $k \geq 1$, the family $\mathbf{HOM}_k(\mathbf{CF})$ contains context-sensitive languages only.

Proposition 11.3.2. *For any* $k \geq 1$, $\mathbf{HOM}_n(\mathbf{CF}) \subset \mathbf{NSPACE}(n)$.

Proof : Given $L \subseteq V^*$ (by a context-free grammar or a pushdown automaton) and $\Omega_{k,V,U} = (h_1, h_2, \ldots, h_k)$ for some alphabet U, it is easy to construct an on-line Turing machine M with one storage tape which works as follows:

- The read-only input tape contains the string w of length n which is to be analyzed. M guesses a pair (x, i), where $x \in V^*$ is written on the storage tape, $|x| \leq n$, and $1 \leq i \leq k$ such that $h_i(x)$ is a prefix of w.

- Then, M checks whether or not $x \in L$. If $x \in L$, then M chooses nondeterministically $1 \leq j \leq k$ and checks whether or not $h_j(x)$ is the next factor of w, and continues in this way until the input string is completely read. When the input string is completely read, M accepts w.

- If there is no pair (x, i) as above, then M rejects w. Clearly, M accepts w iff $w \in \Omega_n^*(L)$ and the total space used on the storage tape is bounded by $|w|$. □

It is easy to note that the family **CF** in the above proof can be replaced by the family of context-sensitive languages.

As far as the possibility of having infinite hierarchies of families of languages defined by arbitrary multiple pattern interpretations of finite, regular, or context-free languages is concerned, we state the following partial result:

Theorem 11.3.2. *Both hierarchies* $\mathbf{HOM}_n^*(\mathbf{FIN}) \subseteq \mathbf{HOM}_{n+1}^*(\mathbf{FIN})$ *and* $\mathbf{LHOM}_n^*(\mathbf{FIN}) \subseteq \mathbf{LHOM}_{n+1}^*(\mathbf{FIN})$ *are infinite.*

Proof : For a given n we consider the alphabet $V_n = \{a_1, a_2, \ldots, a_n\}$. It is obvious that $V_n^+ \in \mathbf{LHOM}_n^*(\mathbf{FIN})$. Assume now that $V_n^+ = \Omega_{n-1}^*(L)$, where $\Omega_{n-1} = (h_1, h_2, \ldots, h_{n-1})$ and L is a finite language. We take $z = a_1^{p_1} a_2^{p_2} \ldots a_n^{p_n} \in V_n^+$ for arbitrarily large $p_1, p_2, \ldots, p_n$. Assume that $z = h_{(i,1)}(x) h_{(i,2)}(x) \ldots h_{(i,k)}(x)$ for some $k \geq 1$. Since L is finite and $p_1, p_2, \ldots, p_n$ are arbitrarily large, it follows that there are $1 \leq j_1, j_2, \ldots, j_n \leq k$ such that $h_{(i,j_t)}(x) \in a_t^+$ for all $1 \leq t \leq n$, which is contradictory. □

We do not know whether any of the hierarchies $\mathbf{HOM}_n^*(\mathbf{X}) \subseteq \mathbf{HOM}_{n+1}^*(\mathbf{X})$, $\mathbf{LHOM}_n^*(\mathbf{X}) \subseteq \mathbf{LHOM}_{n+1}^*(\mathbf{X})$, $\mathbf{X} \in \{\mathbf{REG}, \mathbf{CF}\}$, is infinite.

11.3.1 Closure properties

Theorem 11.3.3. *1. For each* $n \geq 1$, *and* $\mathbf{X} \in \{\mathbf{FIN}, \mathbf{REG}, \mathbf{CF}\}$ *the family* $\mathbf{HOM}_n^*(\mathbf{X})$ *is closed under union and homomorphisms but fails to be closed under concatenation, intersection with regular sets, complement, and set difference.*

Proof : Union: Let

$$U_1^* \supseteq L_1 = \overline{\Omega}_n^*(X_1), X_1 \subseteq V_1^*, X_1 \in \mathbf{FIN},$$
$$U_2^* \supseteq L_2 = \widetilde{\Omega}_n^*(X_2), X_2 \subseteq V_2^*, X_2 \in \mathbf{FIN},$$
where $n \geq 1$,
$\overline{\Omega}_n = (h_1, h_2, \ldots, h_n)$ and $\widetilde{\Omega}_n = (g_1, g_2, \ldots, g_n)$.

Without loss of generality we may assume that V_1 and V_2 are disjoint. We define $\Omega_n = (s_1, s_2, \ldots, s_n)$, where each homomorphism $s_i : (V_1 \cup V_2)^* \longrightarrow (U_1 \cup U_2)^*$, $1 \leq i \leq n$, is defined by

$$s_i(a) = \begin{cases} h_i(a), & \text{if } a \in V_1 \\ g_i(a), & \text{if } a \in V_2 \end{cases}$$

Obviously, $L_1 \cup L_2 = \Omega_{n+m}^*(X_1 \cup X_2)$.

Homomorphisms: Let $\Omega_n = (h_1, h_2, \ldots, h_n)$ be an n-pattern interpretation, $h_i : V^* \longrightarrow U^*$ for all $1 \leq i \leq n$, and $g : U^* \longrightarrow W^*$ be an arbitrary homomorphism. It is plain that $\Omega_n^*(L) = \overline{\Omega}_n^*(L)$ holds for any language $L \subseteq V^*$ and $\overline{\Omega}_n = (g \circ h_1, g \circ h_2, \ldots, g \circ h_n)$.

Concatenation: Both languages a^+ and b^+ are multiple pattern interpretation of finite languages, but a^+b^+ cannot be the multiple pattern interpretation of any language.

Intersection with regular sets: It follows immediate from the fact that V^* is a multiple pattern interpretation of a finite language whereas there are regular languages which cannot be obtained by a multiple pattern interpretation of any language.

Complement and set difference: We take the language $L = \{a^{2n} \mid n \geq 1\} = h^*(\{a\})$, where $h(a) = aa$. But $\overline{L} = \{a\}^* \setminus L$ cannot be the arbitrary multiple pattern interpretation of any language since if $w \in \overline{L}$, then ww is also in $\overline{L}$, hence both $|w|$ and $|ww|$ must be odd, which is contradictory. □

We do not know whether or not a family $\mathbf{HOM}_n(\mathbf{X})$ as above is closed under Kleene closure, but the next result may be interpreted as follows: The Kleene closure of an arbitrary multiple pattern interpretation looses, in some cases, information about the structure imposed by the pattern interpretation. Formally,

Theorem 11.3.4. *Let* **F** *be a family of languages closed under homomorphisms and union. Then, the Kleene closure of any arbitrary multiple pattern interpretation of a language in* **F** *is in* **F**.

Proof : Let $L \subseteq V^*$ be a language in **F** and $\Omega_n = (h_1, h_2, \ldots, h_n)$ be an n-pattern interpretation, for some $n \geq 1$ and $h_i : V^* \longrightarrow U^*$, $1 \leq i \leq n$. We construct the new alphabets $V_i = \{a_i \mid a \in V\}$ and define the letter-to-letter homomorphisms $c_i : V^* \longrightarrow V_i^*$, $c_i(a) = a_i$, $1 \leq i \leq n$. We now consider the language

$$R = (\bigcup_{i=1}^{n} c_i(L))^*,$$

which is still in **F**, and the homomorphism $g : (\cup_{i=1}^{n} V_i)^* \longrightarrow U^*$, defined by $g(a_i) = h_i(a)$ for all $1 \leq i \leq n$, and $a \in V$. We claim that

$$g(R) = (\Omega_n^*(L))^*$$

holds. Indeed, if $z = g(y) \in g(R)$, then $y = x_1 x_2 \ldots x_p$ with $x_j = c_{(i,j)}(z_j)$, $z_j \in L$, $1 \leq j \leq p$. But

$$z = g \circ c_{(i,1)}(z_1) \ldots g \circ c_{(i,p)}(z_p) = h_{(i,1)}(z_1) \ldots h_{(i,p)}(z_p) \in (\Omega_n^*(L))^*.$$

Conversely, if $y = z_1 z_2 \ldots z_r \in (\Omega_n^*(L))^*$, then there are the strings $x_1, x_2, \ldots, x_r$ in L and the positive integers $k_1, k_2, \ldots, k_r$ such that $z_i = h_{(i,1)}(x_i) \ldots h_{(i,k_i)}(x_i)$, $1 \leq (i,1), (i_2), \ldots, (i,k_i) \leq n$, $1 \leq i \leq r$. Hence

$$y = g(c_{(1,1)}(x_1) \ldots c_{(1,k_1)}(x_1) c_{(2,1)}(x_2) \ldots c_{(2,k_2)}(x_2) \ldots c_{(r,1)}(x_r) \ldots c_{(r,k_r)}(x_r)),$$

therefore $y \in g(R)$. □

Corollary 11.3.1. *The Kleene closure of any arbitrary multiple pattern interpretation of a regular (context-free) language is regular (context-free).*

11.3.2 Ambiguity

Given an n-pattern interpretation $\Omega_n = (h_1, h_2, \ldots, h_n)$ and a language L, we say that Ω_n is *weakly ambiguous* on L if there exists a word $x \in L$ such that

$$h_{i_1}(x)h_{i_2}(x)\cdots h_{i_k}(x) = h_{j_1}(x)h_{j_2}(x)\cdots h_{j_p}(x),$$

holds for some $k, p \geq 1$.

Given an n-pattern interpretation $\Omega_n = (h_1, h_2, \ldots, h_n)$ and a language L, we say that Ω_n is *strongly ambiguous* on L if there exist two different words x and y in L such that:

$$h_{i_1}(y)h_{i_2}(y)\cdots h_{i_k}(y) = h_{j_1}(x)h_{j_2}(x)\cdots h_{j_p}(x),$$

holds for some $k, p \geq 1$.

If Ω_n is weakly/strongly ambiguous on any language L in a family of languages F, then Ω_n is said to be *inherently weak/strong ambiguous* on F.

Theorem 11.3.5. *It is decidable whether or not an arbitrary multiple pattern interpretation is weakly/strongly ambiguous on a finite language L.*

Proof : First we discuss how the strong ambiguity can be algorithmically checked. Let $L = \{x_1, x_2, \ldots, x_k\} \subseteq V^*$ and $\Omega_{n,V,U}$ an arbitrary multiple pattern interpretation. We set $L_i = L \setminus \{x_i\}$ for any $1 \leq i \leq k$. It is plain that Ω_n is not strongly ambiguous on L if and only if $\Omega_n^*(L_i) \cap \Omega_n^*(\{x_i\}) = \emptyset$ for all $1 \leq i \leq k$. Since $\Omega_n^*(L_i) \cap \Omega_n^*(\{x_i\})$ is a regular language which can be effectively constructed and the emptiness problem is decidable for regular languages, we are done.

Let $h_i(x_j) = w_{(i,j)}$, $1 \leq i \leq n$, $1 \leq j \leq k$; clearly Ω_n is not weakly ambiguous if and only if for each $1 \leq i \leq n$ the following two conditions are satisfied:

(i) $w_{(i,j)} \neq w_{(i,r)}$, $1 \leq j \neq r \leq k$,

(ii) $\{w_{(i,1)}, w_{(i,2)}, \ldots, w_{(i,k)}\}$ is a code, or equivalently the semigroup $\{w_{(i,1)}, w_{(i,2)}, \ldots, w_{(i,k)}\}^+$ is free.

Obviously, the first condition can be algorithmically checked while the second condition can be checked by Sardinas-Paterson algorithm (Berstel and Perrin (1984)) for testing injectivity of the homomorphism $f : \{a_1, a_2, \ldots, a_k\}^* \longrightarrow U^*$ defined by $f(a_j) = w_{(i,j)}$, $1 \leq j \leq k$. It is known that, provided (i) holds, (ii) is satisfied if and only if f is injective (see, Shyr and Thierrin (1977)). □

Theorem 11.3.6. *Let* **F** *be a family of languages having the following two properties:*

1. It is effectively closed under union, letter-to-letter homomorphisms and concatenation with symbols.

2. The problem "Given two languages $L_1, L_2 \in \mathbf{F}$, is $L_1 \cap L_2$ empty?" is undecidable.

Then, given an n-pattern interpretation Ω_n, $n \geq 1$, and a language $L \in \mathbf{F}$, one cannot algorithmically decide whether or not Ω_n is strongly ambiguous on L.

Proof : Let $L_1 \subseteq V_1^*$ and $L_2 \subseteq V_2^*$ be two arbitrary languages in $\mathbf{F}$. We construct the letter-to-letter homomorphism $g : V_2^* \longrightarrow U^*$, where $U = \{X_a \mid a \in V_2\}$ such that $U \cap V_1 = \emptyset$, defined by $g(a) = X_a$ for each $a \in V_2$, and then consider the language

$$L = \{\$\}L_1\{\#\} \cup \{\$\}g(L_2)\{\#\},$$

where \$ and # are two new symbols. Now we take the homomorphism $h : (V_1 \cup U \cup \{\$, \#\})^* \longrightarrow (V_1 \cup V_2 \cup \{\$, \#\})^*$, defined by $h(a) = a$ for $a \in V_1$, $h(X_b) = b$ for all $b \in V_2$, and $h(\$) = \$$, $h(\#) = \#$.

Clearly, h is strongly ambiguous on L if and only if $L_1 \cap L_2 \neq \emptyset$. Indeed, if $w \in L_1 \cap L_2$, then $\$g(w)\# \in \{\$\}g(L_2)\{\#\} \subseteq L$ and $\$g(w)\#$ is different than $\$w\#$. But $h(\$w\#) = h(\$g(w)\#)$, hence h is strongly ambiguous on L.

Conversely, if h is strongly ambiguous on L, then there are $x, y \in L$, $x \neq y$, such that $h^n(x) = h^m(y)$ for some positive integers n, m. As $h^k(z)$ contains exactly k occurrences of \$ for any $k \geq 1$ and $z \in L$, it follows that $n = m$. By the definition of h, one cannot have both strings either from $\{\$\}L_1\{\#\}$ or from $\{\$\}g(L_2)\{\#\}$. Assume that $x = \$z\#$, with $z \in L_1$, and $y = \$g(w)\#$, with $w \in L_2$ (the other case may be treated similarly). For $h^n(x) = h^n(y)$, it follows that $(\$z\#)^n = (\$w\#)^n$, hence $z = w$, that is $L_1 \cap L_2 \neq \emptyset$. □

Since the family of context-free languages has all properties above, we get:

Corollary 11.3.2. *Given an n-pattern interpretation Ω_n, $n \geq 1$, and a context-free language L, one cannot algorithmically decide whether or not Ω_n is strongly ambiguous on L.*

The decidability status of the weak ambiguity remains *open*.

11.4 Conclusion and further work

We have investigated some properties of the families of languages obtained by arbitrary multiple pattern interpretations of finite, regular, and context-free languages. Some closure properties, most of them being negative results, of these families were presented. In spite of the fact that a characterization of the arbitrary multiple pattern interpretation of finite languages was given the problem of deciding whether or not a regular language is such a language remained open. Two concepts of ambiguity and inherent ambiguity of multiple pattern interpretation were

defined. It was shown that both properties were decidable for multiple pattern interpretations on finite languages but strong ambiguity was not decidable for multiple pattern interpretations on the class of context-free languages.

We finish with a brief discussion about some directions for further research. A multiple pattern interpretation is said to be *inherently weakly/strongly ambiguous* on a family of languages $\mathbf{X}$ if it is weakly/strongly ambiguous on any language in $\mathbf{X}$. Clearly, there exist multiple pattern interpretations which are inherently weakly/strongly ambiguous on a given family $\mathbf{X}$; it suffices to take all the homomorphic images as being power of a common word. A language $L \in \mathbf{HOM}_n^*(\mathbf{X})$ is said to be inherently weakly/strongly ambiguous if for any multiple pattern interpretation Ω_n such that $\Omega_n^*(E) = L$, for some $E \in \mathbf{X}$, then Ω_n is weakly/strongly ambiguous on E. Note that it is not obligatory Ω_n be inherently ambiguous on $\mathbf{X}$. We hope to return to this topic in a further work.

Bibliography

Angluin, D. (1980) Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62.

Berstel, J. and Perrin, D. (1984) *The Theory of Codes*. Academic Press, New York.

Borsley, R. (1999) *Syntactic Theory. A Unified Approach*. Edward Arnold, London.

Braine, M. (1976) Children's first word combinations. *Monographs of the Society for Research in Child Development*, 41 (Serial N0. 164).

Brown, B. and Leonard, L. (1986) Lexical influences on children's early positional patterns. *Journal of Child Language*, 13:219–229.

Chomsky, N. (1965) *Aspects of the Theory of Syntax*. MIT Press, Cambridge MA.

Chomsky, N. (1975) *The Logical Structure of Linguistic Theory*. Plenum, New York.

Chomsky, N. (1981) *Lectures on Government and Binding*. Foris, Dordrecht.

Chomsky, N. (1986) *Knowledge of Language. Its Nature, Origin and Use*. Praeger, New York.

Jiang, T., Kinber, E., Salomaa, A., Salomaa, K., and Yu, S. (1994) Pattern languages with and without erasing. *International Journal of Computer Mathematics*, 50:147–163.

Kolb, H.P. and Mönnich, U. (1999) Introduction. In *The Mathematics of Syntactic Structure*, H.P. Kolb and U. Mönnich, (eds.). Mouton de Gruyter, Berlin.

Kudlek, M., Martín-Vide, C., and Mitrana, V. (2003) Multiple pattern interpretations. *GRAMMARS*, 5:223–238.

Owens, R.E. (2001) *Language Development: An Introduction*. Allyn and Bacon, Boston.

Păun, G. and Salomaa, A. (1995) Thin and slender languages. *Discrete Appl. Math.*, 61:257–270.

Pine, J.M. and Lieven, E.V.M. (1993) Reanalysing rote-learned phrases: individual differences in the transition to multiword speech. *Journal of Child Language*, 20:551–571.

Restivo, A. and Salemi, S. (2002) Words and patterns. *Lecture Notes in Computer Science 2295*. Springer, Berlin, 117–129.

Rozenberg, G. and Salomaa, A. (1980) *The Mathematical Theory of L Systems*. Academic Press, New York.

Rozenberg, G. and Salomaa, A. (eds.) (1997) *Handbook of Formal Languages*. 3 vols. Springer, Berlin.

Shyr, H.J. and Thierrin, G. (1977) Languages and MOL schemes. *R.A.I.R.O. Informatique Theorique/Theoretical Computer Science*, 1,4: 293–301.

Vogler, H. (1999) Principle languages and principle-based parsing. In *The Mathematics of Syntactic Structure*, H.P. Kolb and U. Mönnich, (eds.). Mouton de Gruyter, Berlin, 83–111.

Chapter 12

A set-theoretical investigation of Pāṇini's *Śivasūtras*

WIEBKE PETERSEN*

ABSTRACT. In Pāṇini's grammar one finds the *Śivasūtras*, a table which defines the natural classes of phonological segments in Sanskrit by intervals. We present a formal argument which shows that, using his representation method, Pāṇini has chosen an optimal way of ordering the phonological segments to represent the natural classes. The argument is based on a strict set-theoretical point of view depending only on the set of natural classes and does not explicitly take into account the phonological features of the segments, which are, however, implicitly given in the way a language clusters its phonological inventory. Moreover, the argument is so general that it allows one to decide for each set of sets whether it can be represented with Pāṇini's method. Actually, Pāṇini had to modify the set of natural classes in order to define it by the *Śivasūtras* (the segment *h* plays a special role). We show that this modification was necessary and, in fact, the best possible modification. We discuss how every set of classes can be modified in such a way that it can be defined in a *Śivasūtra*-style representation.[1]

12.1 Pāṇini's *Śivasūtras*

Pāṇini's grammar is recognized as a consistent theoretical analysis of spoken Sanskrit (*bhāṣā*) of the time of its origin (ca. 350 BC). The *Śivasūtras* form the first part of it (a short survey of the structure of Pāṇini's grammar can be found in Kiparsky, 1994) and define the phonological segments of the language and their grouping in natural phonological classes, called *pratyāhāras*. In the *Aṣṭādhyāyī*, a system of about 4000 grammatical rules or rule elements forming the central part of his grammar, Pāṇini refers to 42 of the *pratyāhāras* in hundreds of rules.

*Institut für allgemeine Sprachwissenschaft, Heinrich-Heine-Universität Düsseldorf, Germany; petersew@uni-duesseldorf.de

[1]This approach fits naturally in the framework of Formal Concept Analysis, since the investigated graphs are formal concept lattices. The proofs of the presented propositions can be found in my thesis, which will appear in 2003, and are sketched in an the extended version of the present paper.

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

1.	a	i	u			Ṇ
2.				ṛ	ḷ	K
3.		e	o			Ṅ
4.		ai	au			C
5.	h	y	v	r		Ṭ
6.					l	Ṇ
7.	ñ	m	ṅ	ṇ	n	M
8.	jh	bh				Ñ
9.			gh	ḍh	dh	Ṣ
10.	j	b	g	ḍ	d	Ś
11.	kh	ph	ch	ṭh	th	
			c	ṭ	t	V
12.	k	p				Y
13.		ś	ṣ	s		R
14.	h					L

Table 12.1: Pāṇini's *Śivasūtras*

The *Śivasūtras* state 42 phonological segments and consist of 14 *sūtras* (rows in table 12.1), each of which consists of a sequence of phonological segments (transcribed with small letters) bounded by a marker (transcribed with a capital letter), called *anubandha*. Phonological classes are denoted by abbreviations, called *pratyāhāras*, consisting of a phonological segment and an *anubandha*. The elements of such a class are defined by the *Śivasūtras* given in table 12.1 and are the continuous sequence of phonological segments starting with the given segment and ending with the last segment before the *anubandha*. Table 12.2 gives an example of a *pratyāhāra*.

1.	a	**i**	**u**			Ṇ
2.				**ṛ**	**ḷ**	K
3.		**e**	**o**			Ṅ
4.		**ai**	**au**			C
5.	h	y	v	r		Ṭ

Table 12.2: Example of a *pratyāhāra*: iC = {i,u,ṛ,ḷ,e,o,ai,au}

In this way 285 *pratyāhāras* can be constructed, which is more than the 42 actually needed by Pāṇini, but it is still a small number compared to the number of all classes that can be formed from the phonological segments, which is $2^{42} > 4 \cdot 10^{12}$.

As Kornai (1993) points out clearly, the task of characterizing a phonological

system of a language is to specify the segmental inventory, phonological rules and the set of natural classes of phonological segments which allow generalized rules. The set of natural classes is externally given by the phonological patterning of a language and it always meets two conditions: it is small compared to the set of unnatural classes and the nonempty intersection of two natural classes is a natural class itself.

Kornai stresses that the representation device used for the notation of the classes must make it easier to use natural classes than unnatural ones (e.g. the complement of a natural class is generally unnatural). Contemporary phonological theories build up a structured system of phonological features which are used to characterize the natural classes. Instead of referring to phonological features in order to define a phonological class, Pāṇini refers to intervals in a linear order of the phonological segments. His method of defining the natural classes by *pratyāhāras* – intervals of the *Śivasūtras* – meets the required conditions.

The phonological classes of a grammar are mutually related: classes can be subclasses of other classes, two or more classes can have common elements, etc. These connections are naturally represented in a hierarchy. A *Śivasūtra*-style representation encodes such connections in a linear form.[2] An aim of this paper is to determine the conditions under which a set of sets does have a *Śivasūtra*-style linear representation.

The rest of the paper is organized as follows: In section 12.2 a general formalization of Pāṇini's *Śivasūtra*-style representation of phonological classes is given. Furthermore, the main questions which will be answered in the course of the paper are raised. Section 12.3 explains how the Hasse-diagrams of sets of subsets determine whether a *Śivasūtra*-style representation of natural classes exists. Since some results of graph theory are needed, a brief introduction to planar graphs is given. Finally, in section 12.4 a procedure is presented which constructs an optimal *Śivasūtra*-style representation of a set of natural classes if it exists. This section ends with the proof that Pāṇini has chosen a perfect *Śivasūtra*-style representation. The whole approach is based only on a set-theoretical investigation of the set of natural classes used in Pāṇini's grammar of Sanskrit. No external – phonological – arguments are involved.

12.2 General definitions and the main questions

Definition 12.2.1. *A* well-formed *Śivasūtra*-alphabet *(short* S-alphabet*) is a triple* $(\mathscr{A}, \Sigma, <)$ *consisting of a finite alphabet* $\mathscr{A}$ *and a finite set of markers* Σ *(such that*

[2] Since Pāṇini's grammar was designed for oral tradition, the linear form of the *Śivasūtras* was a prerequisite.

$\mathscr{A} \cap \Sigma = \emptyset$*), and a total order* $<$ *on* $\mathscr{A} \cup \Sigma$*.*

Definition 12.2.2. *A subset* T *of the alphabet* $\mathscr{A}$ *is* S-encodable in $(\mathscr{A}, \Sigma, <)$ *iff there exists* $a \in \mathscr{A}$ *and* $M \in \Sigma$*, such that* $T = \{b \in \mathscr{A} | a \leq b < M\}$*.* aM *is called the* pratyāhāra *or* S-encoding *of* T *in* $(\mathscr{A}, \Sigma, <)$*.*

Definition 12.2.3. *An S-alphabet* $(\mathscr{A}', \Sigma, <)$ corresponds *to a system of sets* $(\mathscr{A}, \Phi)$ *(where* Φ *is a set of subsets of* $\mathscr{A}$*) iff* $\mathscr{A} = \mathscr{A}'$ *and each element of* Φ *is S-encodable in* $(\mathscr{A}', \Sigma, <)$*. An S-alphabet which corresponds to* $(\mathscr{A}, \Phi)$ *is called an* S-alphabet of $(\mathscr{A}, \Phi)$*. A system of sets for which a corresponding S-alphabet exists is said to be* S-encodable*.*

For example, take the set of subsets

(12.2.1) $\Phi = \{\{d,e\},\{b,c,d,f,g,h,i\},\{a,b\},\{f,i\},\{c,d,e,f,g,h,i\},\{g,h\}\}$

of the alphabet $\mathscr{A} = \{a,b,c,d,e,f,g,h,i\}$: it is S-encodable and

(12.2.2) $a\ b\ M_1\ c\ g\ h\ M_2\ f\ i\ M_3\ d\ M_4\ e\ M_5$

is one of the corresponding S-alphabets. The *pratyāhāras* of Φ are: dM_5, bM_4, aM_1, fM_3, cM_5 and gM_2.

Definition 12.2.4. *An S-alphabet* $(\mathscr{A}, \Sigma, <)$ *of* $(\mathscr{A}, \Phi)$ *is said to be* optimal *iff there exists no other S-alphabet* $(\mathscr{A}, \Sigma', <')$ *of* $(\mathscr{A}, \Phi)$ *such that the set of markers* Σ' *has fewer elements than* Σ*.*

Looking at Pāṇini's *Śivasūtras* it is striking that the phonological segment *h* occurs twice, namely in *sūtra* 5 and *sūtra* 14. To model this phenomenon we will introduce the concept of *enlarging* an alphabet by duplicating some of its elements.[3]

$\hat{\mathscr{A}}$ is said to be an *enlarged alphabet* of $\mathscr{A}$ if there exists a surjective map $\vartheta : \hat{\mathscr{A}} \to \mathscr{A}$. It is clear that for every system of sets $(\mathscr{A}, \Phi)$ we can find an enlarged alphabet $\hat{\mathscr{A}}$ and a set of subsets $\hat{\Phi}$ with $\Phi = \{\vartheta(\varphi') : \varphi' \in \hat{\Phi}\}$ such that $(\hat{\mathscr{A}}, \hat{\Phi})$ is S-encodable. To achieve such an S-encodable system of sets $(\hat{\mathscr{A}}, \hat{\Phi})$ we enlarge $\mathscr{A}$ so that the sets of $\hat{\Phi}$ are disjoint. Then we arrange the sets of $\hat{\Phi}$ in a sequence and separate them by markers. The induced S-alphabet $(\hat{\mathscr{A}}, \hat{\Sigma}, \hat{<})$ then obviously corresponds to $(\hat{\mathscr{A}}, \hat{\Phi})$.

An S-alphabet of $(\hat{\mathscr{A}}, \hat{\Phi})$ will sometimes be called an *enlarged S-alphabet of* $(\mathscr{A}, \Phi)$. Since we always find a finite, enlarged S-alphabet of $(\mathscr{A}, \Phi)$, a minimally enlarged S-alphabet exists.

[3] Duplicating an element a means adding a new element a' to $\mathscr{A}$ and changing some of the occurrences of a in Φ to a'.

Definition 12.2.5. *An enlarged S-alphabet* $(\hat{\mathscr{A}}, \hat{\Sigma}, \hat{<})$ *of* $(\mathscr{A}, \Phi)$ *is said to be* perfect *iff it fulfills the following conditions: First, there exists no other enlarged S-alphabet* $(\tilde{\mathscr{A}}, \tilde{\Sigma}, \tilde{<})$ *of* $(\mathscr{A}, \Phi)$*, the alphabet* $\tilde{\mathscr{A}}$ *of which has fewer elements than* $\hat{\mathscr{A}}$ *and furthermore, as a secondary condition,* $(\hat{\mathscr{A}}, \hat{\Sigma}, \hat{<})$ *is optimal.*[4]

After this introduction of basic concepts the following questions will be investigated in the present paper:

1. Given a system of sets, is it possible to decide whether it is S-encodable?
2. If a system of sets is S-encodable, how can we construct an optimal corresponding S-alphabet?

And with respect to the special case of the phonological classes defined by Pāṇini's *Śivasūtras*:

3. Is the duplication of *h* in Pāṇini's *Śivasūtras* necessary?
4. Are Pāṇini's *Śivasūtras* perfect?

Kiparsky (1991) discusses questions 3 and 4 and affirms both, as I will do in what follows, using a different approach.

12.3 The existence of *Śivasūtra*-style representations of systems of sets

12.3.1 A Brief introduction to the theory of planar graphs

Throughout this paper we will need some basic knowledge about planar graphs, which will be briefly introduced in this section.[5] A *graph G* is a pair (V, E) consisting of a set of *vertices V* and a set of *edges* $E \subseteq V \times V$. *Paths* in graphs are defined in the natural way and *circles* are closed paths, as usual. *Directed graphs* are graphs the edges of which are directed.

A graph is a *plane graph* if its vertices are points in the Euclidean plane $\mathbb{R} \times \mathbb{R}$ and its edges are polygonal arcs in $\mathbb{R} \times \mathbb{R}$, such neither a vertex nor a point of an edge lies in the inner part of another edge. The Euclidean plane $\mathbb{R} \times \mathbb{R}$ is subdivided by a plane graph into *faces* (areas). Exactly one of this faces, the *infinite face*, is of unlimited size.

[4] Note that for every system of sets a perfect S-alphabet exists.

[5] The full details can be found in every introductory book on graph theory. In writing this paper Diestel (1997) proved to be especially helpful.

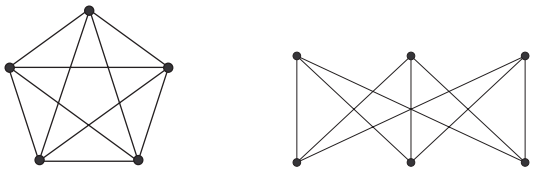


Figure 12.1: The graphs K^5 and $K_{3,3}$

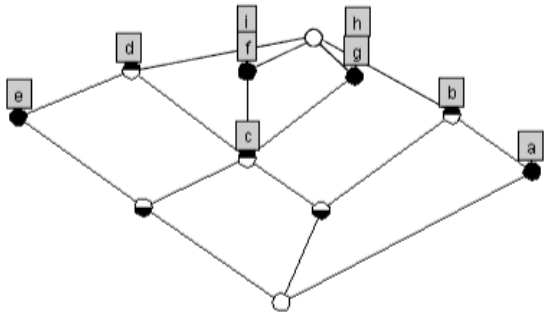


Figure 12.2: Hasse diagram of $(\mathscr{H}(\Phi),\supseteq)$, $\Phi = \{\{d,e\},\{b,c,d,f,g,h,i\},\{a,b\},\{f,i\},\{c,d,e,f,g,h,i\},\{g,h\}\}$

If a graph is isomorphic to a plane graph, it is said to be *planar*. One of the most important criteria for the planarity of graphs is the criterion of Kuratowski, which is based on the notion of minors of a graph. A graph *M* is said to be a *minor* of a graph *G* if it can be arrived from *G* by first removing a number of vertices and edges from *G* and then contracting some of the remaining edges.

Proposition 12.3.1 (Criterion of Kuratowski). *A graph G is planar iff G contains neither a K^5 nor a $K_{3,3}$ as a minor (see figure 12.1).*

12.3.2 Plane Hasse-diagrams and S-encodability

Let $(\mathscr{A},\Phi)$ be a system of sets as above, and let $\mathscr{H}(\Phi)$ be the set of all intersections of elements of $\Phi\cup\{\mathscr{A}\}$. $\mathscr{H}(\Phi)$ is partially ordered by the superset relation. A Hasse-diagram of a partially ordered set is a drawing of a directed graph whose vertices are the elements of the set and whose edges correspond to the upper neighbor relations determined by the partial order. The drawing must meet the following condition: if an element of the partially ordered set is an upper neighbor of another element, then its vertex lies above the vertex of the other one. In this paper we will stipulate that all edges are directed upwards.

Figure 12.2 shows a drawing of the Hasse-diagram $(\mathscr{H}(\Phi),\supseteq)$ of our example

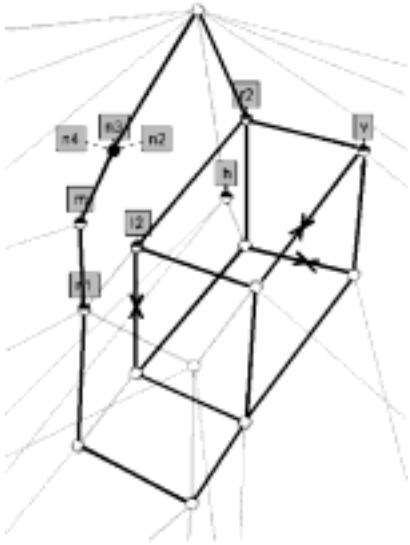


Figure 12.3: A section of the Hasse-diagram of the *pratyāhāras* used in the *Aṣṭādhyāyī* which has K^5 as a minor. The figure shows that the class memberships of the phonological segments *h*,*v* and *l* (denoted by *l2*) are independent of each other.

system of sets $(\mathscr{A},\Phi)$ from (12.2.1). All Hasse-diagrams in this paper are labeled economically as follows: for every $b \in \mathscr{A}$ the smallest set of $\mathscr{H}(\Phi)$ containing *b* (its *b*-set) is labeled. The set corresponding to a vertex of the diagram can be reconstructed by collecting all elements which are labeled to the vertex itself or to a vertex above. For example, in figure 12.2 the vertex labeled with *c* corresponds to the set $\{c,d,f,g,h,i\}$.

The Hasse-diagram of $(\mathscr{H}(\Phi),\supseteq)$ gives a first hint of the question whether $(\mathscr{A},\Phi)$ is S-encodable:

Proposition 12.3.2. *If* $(\mathscr{A},\Phi)$ *is S-encodable, then the Hasse-diagram of* $(\mathscr{H}(\Phi),\supseteq)$ *is a planar graph.*[6]

It follows as a corollary that a system of sets is not S-encodable whenever the Hasse-diagram of the corresponding set of intersections is not planar.

Together with Kuratowski's criterion 12.3.1 this answers question 3, since figure 12.3 shows a section of the Hasse-diagram of the *pratyāhāras* and their intersections, which has K^5 as a minor.[7] Hence, Pāṇini was forced to duplicate at

[6]Due to the limited space, no proofs are given in this paper, but the results are illustrated by a number of examples.

[7]The emphasized lines in the figure mark a way to arrive at the minor K^5: remove all edges which

least one of the phonological segments. But it remains to prove that *h* is the best candidate for the duplication; this discussion will be postponed.

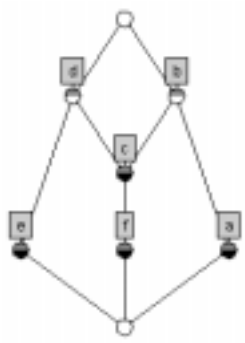


Figure 12.4: Hasse-diagram of the intersections of the sets $\{\{d,e\},\{a,b\},\{b,c,d\},\{b,c,d,f\},\{a,b,c,d,e,f\}\}$. It is plane, but there exists no corresponding S-alphabet.

The condition for S-encodable systems of sets given in proposition 12.3.2 is necessary but not sufficient, however. Figure 12.4 shows an example of a system of sets which is not S-encodable, although the Hasse-diagram of its intersection sets is planar. We need a second proposition to fully identify those systems of sets which are S-encodable.

If $(\mathscr{A},\Phi)$ is a system of sets which is S-encodable, then the boundary of the infinite face of a plane Hasse-diagram of $(\mathscr{H}(\Phi)\setminus\emptyset,\supseteq)$ is called the *S-graph of* $(\mathscr{A},\blacksquare)$. It can be shown that the S-graph of $(\mathscr{A},\Phi)$ is fixed up to isomorphism, and is therefore independent of the chosen embedding of the Hasse-diagram in $\mathbb{R}^2$.

Proposition 12.3.3. *If $(\mathscr{A},\Phi)$ is S-encodable, then the Hasse-diagram of $(\mathscr{H}(\Phi),\supseteq)$ is a planar graph and the S-graph of $(\mathscr{A},\Phi)$ meets the following condition: For every $b\in\mathscr{A}$ the smallest set of $\mathscr{H}(\Phi)$ containing b (its b-set) is a vertex of the S-graph.*

In the example of figure 12.4 the f-set violates the condition of proposition 12.3.3 since in a plane Hasse-diagram of $(\mathscr{H}(\Phi)\setminus\emptyset,\supseteq)$ it would only touch inner faces. It is obvious that there exists no alternative embedding of the Hasse-diagram into $\mathbb{R}^2$ which would fulfill both conditions.

---

are not emphasized and contract those edges which are marked by arrows.

12.4 The construction of *Śivasūtra*-style representations of systems of sets

12.4.1 The boundary graph determines the S-alphabet

If $(\mathscr{A}, \Phi)$ is a system of sets which is S-encodable, then an S-alphabet $(\mathscr{A}, \Sigma, <)$ of $(\mathscr{A}, \Phi)$ can be found as follows: Take the labeled S-graph of $(\mathscr{A}, \Phi)$ and a path in it, that starts and ends at the vertex corresponding to $\mathscr{A}$. The path must meet the following conditions: First, for every $a \in \mathscr{A}$ the path passes the a-set at least once; second, none of the edges occurring more than once in the path is part of a circle in the S-graph. By looking at the S-graph as a subgraph of the directed Hasse-diagram, the edges of the path can be directed.

The S-alphabet, seen as a sequence of markers and elements of $\mathscr{A}$, can be constructed from the empty sequence by traversing the path from the beginning to the end: If a vertex is reached which is labeled with an a-set, then add a to the sequence, together with all other labels of the same vertex. If an edge is passed whose direction contradicts the traversal direction, a new, previously unused, marker element is added to the sequence, unless the last added element is already a marker. Finally, after the end of the path is reached, revise the sequence as follows: If an element of $\mathscr{A}$ appears more than once in the sequence, delete all occurrences of it except one, and if two markers happen to occur next to each other, remove one.

Applied to our small example (12.2.1) and the plane graph of its Hasse-diagram given in figure 12.2, we may choose the path illustrated in figure 12.5, which fulfills the required conditions. Traversing the path, we pass first the a-set and the b-set without using an edge against its destined direction. Now we move downwards and violate the direction of the edge, and therefore we have to add a marker to our sequence, so that it starts with $a\ b\ M_1$. Now moving upwards we collect the c and the g, but since the g- and the h-sets are identical we also have to collect the h. After this we move downwards again, and that is why we add a new marker. We again reach the c-set and add c a second time to our sequence. So far our sequence is $a\ b\ M_1\ c\ g\ h\ M_2\ c$, and if we continue we end up with the S-alphabet depicted in (12.2.2).

Note that this procedure does not yield a unique S-alphabet since we have several decisions to make: (a) If a vertex is labeled with more than one element, their order in the S-alphabet is arbitrary; (b) the ultimately deleted elements are arbitrarily chosen; (c) from the vertex labeled c we can either go to the vertex labeled gh or if; (d) the path can be traversed clockwise or anti-clockwise.

It can be shown that every optimal S-alphabet of $(\mathscr{A}, \Phi)$ can be constructed in this way by finding a path which fulfills the conditions and violates the direction of the edges as seldom as possible. This proves our main theorem and answers

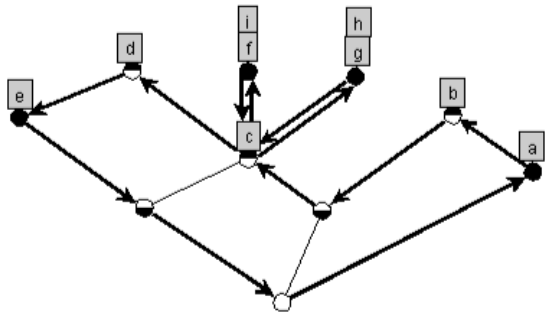


Figure 12.5: Hasse-diagram of example (12.2.1) with a possible path in its S-graph from which the S-alphabet $a\ b\ M_1\ c\ g\ h\ M_2\ f\ i\ M_3\ d\ M_4\ e\ M_5$ can be achieved.

questions 1 and 2.

Theorem 12.4.1. *$(\mathscr{A},\Phi)$ is S-encodable iff the Hasse-diagram of $(\mathscr{H}(\Phi),\supseteq)$ is isomorphic to a plane graph G and for every $b\in\mathscr{A}$ the b-set lies at the boundary of the infinite face of G', where G' is obtained from G by removing the vertex of the empty set and all corresponding edges.*

If $(\mathscr{A},\Phi)$ is S-encodable, then all optimal S-alphabets $(\mathscr{A},\Sigma,<)$ of $(\mathscr{A},\Phi)$ can be constructed systematically.

12.4.2 Pāṇini's *Śivasūtras* are perfect

Figure 12.6 shows the Hasse-diagram, with duplicated *h*, corresponding to the *pratyāhāras* which Pāṇini uses in his *Aṣṭādhyāyī*; the duplication of *h* is denoted by *h*_.[8] The 42 *pratyāhāras* actually used by Pāṇini in the *Aṣṭādhyāyī* are marked in the figure with white boxes. The black and the striped rectangles next to some of the vertices mark the places where markers have to be added, depending on the traversal direction (black: anti-clockwise [14 markers], striped: clockwise [17 markers]). It is obvious that no S-encoding can have less than 14 markers and the optimal S-alphabets are the various combinatorial variants of

$$\begin{aligned}\big\langle &\text{a,i,u},M_1,\text{ṛ,ḷ},M_2,\{\langle\{\text{e,o}\},M_3\rangle,\langle\{\text{ai,au}\},M_4\rangle\},\text{h,y,v,r},M_5,\text{l},M_6,\\ &\text{ñ,m,}\{\text{n̄,ṇ,n,}\},M_7,\text{jh,bh},M_8,\{\text{gh,ḍh,dh}\},M_9,\text{j,}\{\text{b,g,ḍ,d}\},M_{10},\\ &\{\text{kh,ph}\},\{\text{ch,ṭh,th}\},\{\text{c,ṭ,t}\},M_{11},\{\text{k,p}\},M_{12},\{\text{ś,ṣ,s}\},M_{13},\text{h},M_{14}\big\rangle.\end{aligned}$$

Kiparsky (1991) argues in detail that the order chosen by Pāṇini out of the set of possibilities is unique if one requires a subsidiary principle of restrictiveness.

[8] The drawing was done by the tool "Concept Explorer" written by Sergey Yevtushenko, which can be found at `http://www.sourceforge.net/projects/conexp`.

So far we have answered the first three questions in the affirmative. Hence, we have argued that Pāṇini was forced to enlarge the alphabet, but it remains to show why duplicating the *h* is the best choice. If *h* is entirely removed from the *pratyāhāras*, then the optimal S-alphabet has only one marker less, namely 13.

In the *pratyāhāras*, the occurrence of *h* and any two of the segments $\{s, bh, v, l\}$ are independent of each other. Take for example the three segments *h*, *l* and *v*; then there exists a *pratyāhāra* – or an intersection of *pratyāhāras* – for each subset of $\{h, l, v\}$ which contains the elements of the subset but no element of its complement in $\{h, l, v\}$. Therefore, the class memberships of the segments *h*, *l* and *v* are independent of each other. A Hasse-diagram which contains 3 independent elements has K^5 as a minor and is therefore not planar (see figure 12.3). Hence, to avoid the duplication of *h* it would be necessary to duplicate at least 3 of the segments $\{s, bh, v, l\}$, which is worse than duplicating just one. For that reason, it is necessary to duplicate *h* in order to get a perfect S-alphabet corresponding to Pāṇini's *pratyāhāras*.

Summarizing, all 4 questions at issue have to be affirmed. Pāṇini's method of representing hierarchical information in a linear form is an interesting field of further investigations. Also the fact that one does not need to refer to phonological features explicitly in order to define phonological classes is remarkable.

Kornai (1993) points out that Pāṇini's approach is generalized by *feature geometry*, and that it is genuinely weaker than the latter. Although Kornai argues that the power of feature geometry is needed in order to get the proper set of natural classes of a phonological system, for some special tasks like describing the set of *major class features* a *Śivasūtra*-style analysis seems to be quite appropriate.

Finally, it should be emphasized again: The approach presented is so general that it is not limited to the domain of phonology.

Bibliography

Diestel, R. (1997). *Graph theory* Springer, New York.

Kiparsky, P. (1991). Economy and the construction of the Śivasūtras. In M. M. Deshpande and S. Bhate, eds., *Pāṇinian Studies: Professor S D Joshi Felicitation Volume*. Center for South and Southeast Asian Studies, University of Michigan, Ann Arbor, Michigan.

Kiparsky, P. (1994). Pāṇinian Linguistics. In R. E. Asher, ed., *The Encyclopedia of Language and Linguistics*, vol. 6, Pergamon Press, Oxford, pp. 2918–2923.

Kornai, A. (1993). The generative power of feature geometry. In *Annals of Mathematics and Artificial Intelligence* **8**, 37–46.

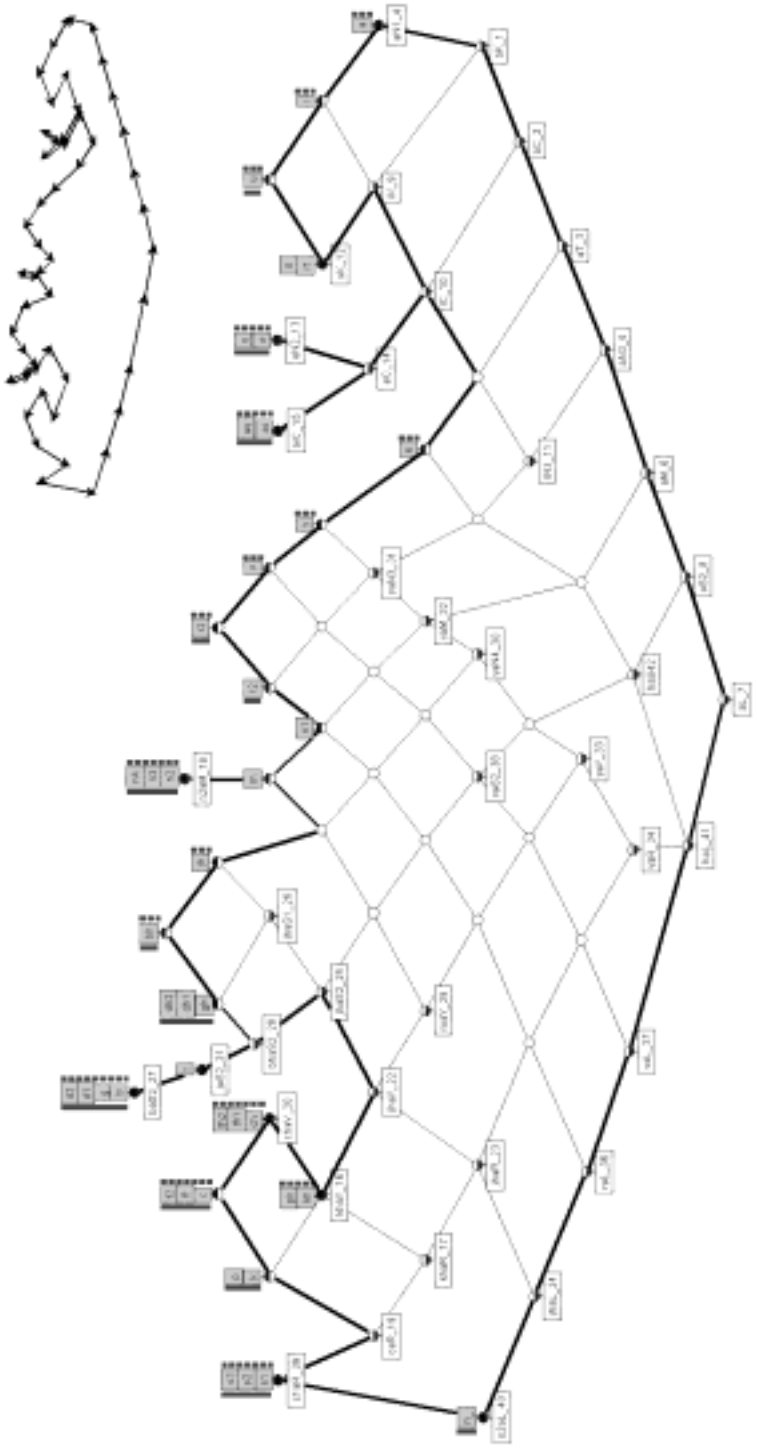


Figure 12.6: Hasse-diagram of the of the *pratyāhāras* used in the *Aṣṭādhyāyī*. The denotations in the figure are as follows: *h_* is the duplicate of *h* and r1:ṛ, l1:ḷ, r2:ṛ, l2:ḷ, n1:ñ, n2:ṅ, n3:ṇ, n4:n, dh1:ḍh, dh2:dh, d1:ḍ, d2:d, th1:ṭh, th2:th, t1:ṭ, t2:t, s1:ś, s2:ṣ, s3:s. The white boxes mark the 42 phonological classes which Pāṇini uses in the *Aṣṭādhyāyī*. The small figure on top shows the path in the S-graph which one has to choose in order to construct Pāṇini's *Śivasūtras*.

Chapter 13

The Semantic Complexity of some Fragments of English

IAN PRATT-HARTMANN*

ABSTRACT. By a fragment of a natural language we mean a subset of that language equipped with a semantics which translates its sentences into some formal system such as first-order logic. The familiar concepts of satisfiability and entailment can be defined for any such fragment in a natural way. The question therefore arises, for any given fragment of natural language, as to the computational complexity of determining satisfiability and entailment within that fragment. This paper presents some new technical results concerning a series of fragments of English for which the satisfiability problem lies in various complexity classes. The paper thus represents a case study in how to approach the problem of determining the semantic complexity of certain natural language constructions.

13.1 Introduction

This paper presents new results on the semantic complexity of various fragments of English. A *fragment* of English is a set of English sentences equipped with a semantics translating those sentences into a formal language such as first-order logic. To investigate the *semantic complexity* of such a fragment is to determine the computational complexity of the satisfiability and entailment problems for sentences in that fragment, as defined by the associated semantics.

We begin with a simple example. Consider the following context-free grammar. This grammar defines a set of English sentences by successive expansion of nonterminals, with the expressions on the right of the obliques indicating the semantic values of the corresponding phrases in the familiar way.

* Department of Computer Science, University of Manchester, U.K.; ipratt@cs.man.ac.uk

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

Syntax	Formal lexicon
$\mathrm{IP}/\phi(\psi) \to \mathrm{NP}/\phi,\ \mathrm{I}'/\psi$	$\mathrm{Det}/\lambda p \lambda q[\exists x(p(x) \wedge q(x))] \to$ some
$\mathrm{I}'/\phi \to$ is a N'/ϕ	$\mathrm{Det}/\lambda p \lambda q[\forall x(p(x) \to q(x))] \to$ every
$\mathrm{I}'/\neg\phi \to$ is not a N'/ϕ	$\mathrm{Det}/\lambda p \lambda q[\forall x(p(x) \to \neg q(x))] \to$ no
$\mathrm{NP}/\phi \to \mathrm{PropN}/\phi$	
$\mathrm{NP}/\phi(\psi) \to \mathrm{Det}/\phi,\ \mathrm{N}'/\psi$	
$\mathrm{N}'/\phi \to \mathrm{N}/\phi$.	

Content lexicon

$\mathrm{N}/\lambda x[\mathrm{man}(x)] \to$ man	$\mathrm{PropN}/\lambda p[p(\mathrm{socrates})] \to$ Socrates
$\mathrm{N}/\lambda x[\mathrm{mortal}(x)] \to$ mortal	$\mathrm{PropN}/\lambda p[p(\mathrm{diogenes})] \to$ Diogenes
...	...

Since the primary form-determining element in this fragment of English is the copula, let us denote the fragment by Cop. The content-lexicon, comprising the open word-classes of common and proper nouns, is assumed to be open-ended. Thus, Cop can be viewed as the union of a *family* of fragments, with each member of that family corresponding to a choice of content-lexicon. In this respect, the notion of a fragment of English resembles the logician's familiar notion of a fragment of a formal language such as first-order logic.

The set of sentences recognized by Cop is, in effect, the familiar language of the syllogism; and the formulas to which Cop translates those sentences are the familiar formal translations found in introductory logic courses. For example, Cop contains the sentences featured in the following argument, and assigns them the corresponding logical translations.

Every man is a mortal	$\forall x(\mathrm{man}(x) \to \mathrm{mortal}(x))$
Socrates is a man	$\mathrm{man}(\mathrm{socrates})$
Socrates is a mortal	$\mathrm{mortal}(\mathrm{socrates})$

This translation allows familiar semantic concepts to be transferred from first-order logic to the fragment Cop in the obvious way. Thus, a set of Cop-sentences E can be said to *entail* a Cop-sentence e if the formulas to which E is translated entail the formula to which e is translated in the usual sense of first-order logic; likewise, a set of Cop-sentences E can be said to be *satisfiable* if the set of formulas to which E is translated is satisfiable in the usual sense of first-order logic. The philosophical justification of this rational reconstruction of entailment and satisfiability need not detain us here, since, for the applications we have in mind, it is not open to productive doubt.

Define the *size* of an English sentence to be the number of words it contains; likewise, define the size of a set E of sentences, denoted $|E|$, to be the sum of the sizes of its members. Using this concept of size, we can formulate complexity-theoretic questions concerning fragments of English in the usual way. In particular, the computational complexity of the satisfiability problem for an English fragment is the number of steps of computation required to determine algorithmically whether a given finite set E of sentences in that fragment is satisfiable, expressed as a function of $|E|$.

Theorem 1. *The problem of determining the satisfiability of a set of sentences in Cop is in PTIME.*

Proof: Every sentence recognized by Cop is easily seen to have a first-order translation matching one of the schemata $\forall x(p(x) \rightarrow \pm \mathrm{q}(x))$, $\exists x(\mathrm{p}(x) \wedge \pm \mathrm{q}(x))$, or $\pm p(c)$, where p and q are predicates and c is a constant. (We take $\pm\phi$ to stand indeterminately for ϕ or $\neg\phi$.) Clausifying such formulas results in function-free clauses matching either of the schemata $\neg p(x) \vee \pm \mathrm{q}(x)$ or $\pm p(c)$. Since resolution only produces more clauses of this form, saturation is reached in PTIME. □

This simple observation suggests a programme of work: take a fragment of English delineated in terms which respect the syntax of the language; then determine the computational complexity of deciding satisfiability in that fragment, if, indeed, the fragment is decidable. From this standpoint, the syllogistic is just one such fragment, with very restricted syntax and a correspondingly efficient decision procedure. In the sequel, we shall investigate what happens as we expand our syntactic horizons. Throughout the paper, we make extensive use of the standard apparatus of resolution theorem-proving, including the notions of *clause*, *A-ordering*, *ordered resolution*, and *splitting*. For the definitions of these terms, the reader is referred to a standard text such as Leitsch (1997).

13.2 Relative clauses

Let Cop+Rel be the fragment defined by the grammar rules of Cop together with the following rules. For the sake of avoiding clutter, here and in the sequel, we suppress the semantic annotations on these rules, since these are completely routine.

Syntax		Formal lexicon
CP → CSpec, C′	N′ → N, CP	RelPro → who, which
C′ → C, IP	NP → RelPro	C →
CSpec →		

In addition, we assume that, following generation of an IP by these rules, relative pronouns are subject to wh-movement to produce the observed word-order. For our purposes, we may take the wh-movement rule to require: (i) the empty position CSpec must be filled by movement of a RelPro from within the IP which forms its right-sister (i.e. which it C-commands); (ii) every RelPro must move to a closest such CSpec position.

As for Cop, so too for Cop+Rel, the semantics map its sentences to first-order logic, thus inducing natural definitions of satisfiability and validity. For example, Cop+Rel contains the sentences featured in the following argument, and assigns them the corresponding logical translations.

Every man who is not a stoic is a cynic	$\forall x(\mathrm{man}(x) \wedge \neg \mathrm{stoic}(x) \rightarrow \mathrm{cynic}(x))$
Every stoic is a fool	$\forall x(\mathrm{stoic}(x) \rightarrow \mathrm{fool}(x))$
Every cynic is a fool	$\forall x(\mathrm{cynic}(x) \rightarrow \mathrm{fool}(x))$
Every man is a fool	$\forall x(\mathrm{man}(x) \rightarrow \mathrm{fool}(x))$

The following result shows us that determining satisfiability has become more difficult.

Theorem 2. *The problem of determining the satisfiability of a set of sentences in Cop+Rel is NP-complete.*

Proof: To show membership in NP, let E be a finite set of Cop+Rel-sentences, and let Φ be the corresponding first-order logic formulas. Since Φ has no nested quantifiers, it is obvious that if Φ is satisfiable, then it has a model whose size is bounded by $|\Phi|$.

To show NP-hardness, we reduce 3SAT to the problem of determining satisfiability in Cop+Rel. Let $\mathscr{C}$ be a set of propositional clauses, each of which has at most three literals. Without loss of generality, we may assume all the clauses in $\mathscr{C}$ to be of the forms $p \vee q$, $\neg p \vee \neg q$ or $\neg p \vee \neg q \vee r$. We then map each clause in $\mathscr{C}$ to a Cop+Rel-sentence as follows. (The first-order translations of these sentences are also given.)

$p \vee q$	Every el which is not a q is a p	$\forall x(\mathrm{el}(x) \wedge \neg q(x) \rightarrow p(x))$
$\neg p \vee \neg q$	No p is a q	$\forall x(p(x) \rightarrow \neg q(x))$
$\neg p \vee \neg q \vee r$	Every p which is a q is an r	$\forall x(p(x) \wedge q(x) \rightarrow r(x))$

Finally, we add the sentence Some el is an el. (Read el as 'element'.) Let the resulting set of Cop+Rel-sentences be E, and let the first-order translations of E be Φ. It is then easy to transform any satisfying assignment for $\mathscr{C}$ into a model for Φ and vice versa. □

13.3 Adding transitive verbs

Whereas adding relative clauses to Cop increases computational complexity, other additions are computationally harmless. Perhaps the simplest involves the addition of transitive verbs. Let Cop+TV be the fragment defined by the grammar rules of Cop together with the following rules. (Again, we assume the obvious semantics.)

Syntax	Formal Lexicon	Content Lexicon
$I' \to VP$	$Neg \to$ does not	$TV \to$ admires
$I' \to NegP$		$TV \to$ despises
$NegP \to Neg, VP$		...
$VP \to TV, NP$		

For the sake of clarity, we have suppressed the issue of verb-inflections as well as that of polarity effects of negative contexts on determiners. In the sequel, we will silently correct any such syntactic shortcomings as required. Accommodating these details in our grammar and ruling out otherwise awkward-sounding sentences can easily be seen not to affect the complexity-theoretic results reported below. For example, Cop+TV contains the sentences featured in the following argument, and assigns them the corresponding logical translations.

Every stoic admires every cynic	$\forall x(\text{sto}(x) \to \forall y(\text{cyn}(y) \to \text{adm}(x,y)))$
No cynic admires any stoic	$\forall x(\text{cyn}(x) \to \neg\exists y(\text{sto}(y) \wedge \text{adm}(x,y)))$
No stoic is a cynic	$\forall x(\text{sto}(x) \to \neg\text{cyn}(x))$

Theorem 3. *The problem of determining the satisfiability of a set of sentences in Cop+TV is in PTIME.*

Proof: If E is a finite set of sentences of Cop+TV, let Φ be the corresponding set of first-order formulas, and let $\mathscr{C}$ be the result of putting Φ into clausal form. It suffices to show that the satisfiability of $\mathscr{C}$ can be computed in polynomial time.

It follows easily from the semantics of Cop+TV that every $C \in \mathscr{C}$ is of one of the following forms.

$$\pm p(c) \quad \pm r(c,d) \quad \neg p(x) \vee \pm r(x, f(x)) \quad \neg p(x) \vee \neg q(y) \vee \pm r(x,y)$$
$$\neg p(x) \vee q(f(x)) \quad \neg p(x) \vee \pm q(x) \quad \neg p(x) \vee \pm r(c,x) \quad \neg p(x) \vee \pm r(x,c)$$

Call a literal *non-unary* if its predicate is not unary. Let $\mathscr{C}^\infty$ be the result of saturating $\mathscr{C}$ under resolution, *but only allowing non-unary literals to be resolved*

upon, and let $\mathscr{D}$ be the result of removing from $\mathscr{C}^\infty$ any clauses containing a non-unary literal. Since every $C \in \mathscr{C}$ contains at most 1 non-unary literal,we have $|\mathscr{D}| \leq |\mathscr{C}^\infty| \leq |\mathscr{C}| + |\mathscr{C}|^2$. We claim that $\mathscr{C}$ and $\mathscr{D}$ are equisatisfiable. Certainly, if $\mathscr{C}$ is satisfiable then $\mathscr{D}$ is. So suppose $\mathscr{C}$ is unsatisfiable, and let $\prec$ be the A-order defined by $A \prec B$ if A is unary and B is non-unary. Since $\prec$-resolution is a complete proof strategy, there must be a derivation of the empty clause from $\mathscr{C}$. But the ordering $\prec$ ensures that, in this derivation (considered as a tree), no steps of resolution on unary literals can precede any step of resolution on non-unary literals. Hence there is a derivation of the empty clause from $\mathscr{D}$, and so $\mathscr{D}$ is unsatisfiable.

Thus it suffices to show that the satisfiability of $\mathscr{D}$ can be decided in PTIME. By definition, the clauses in $\mathscr{D}$ involve only *unary* literals; and it is routine to verify that every $C \in \mathscr{D}$ satisfies the conditions:

P1: at most 2 variables and at most 4 literals occur in C;

P2: the depth of C is at most 1;

P3: if C is non-negative, then C must be of one of the forms

$$(13.3.1) \quad p(c) \qquad \neg p(x) \vee q(x) \qquad \neg p(x) \vee q(f(x)).$$

(The *depth* of any expression X, denoted $d(X)$, is the maximum level of nesting of function-symbols in X, with non-functional expressions assigned depth 0.) Consider the familiar A-ordering $\prec^d$ defined by setting $A \prec^d B$ if and only if: (i) $d(A) < d(B)$, (ii) $\mathrm{Vars}(A) \subseteq \mathrm{Vars}(B)$, and (iii) $d(x,A) < d(x,B)$ for all $x \in \mathrm{Vars}(A)$. (Leitsch 1997, p. 100). It is routine to show that, when $\prec^d$-resolution is applied to the clauses $\mathscr{D}$, properties **P1**–**P3** are preserved. It follows from **P1** and **P2** that saturation is reached in PTIME. □

If adding relative clauses increases complexity but adding transitive verbs does not, the question naturally arises as to what happens when both are added together. That is, let Cop+Rel+TV be the fragment defined by the grammar rules of Cop+Rel together with those of Cop+TV. For example, Cop+Rel+TV contains the sentences featured in the following argument, and assigns them the corresponding logical translations.

Every stoic is a philosopher	$\forall x(\mathrm{sto}(x) \rightarrow \mathrm{phil}(x))$
Every cynic whom some stoic admires is a cynic whom some philosopher admires	$\forall x(\mathrm{cyn}(x) \wedge \exists y(\mathrm{sto}(y) \wedge \mathrm{adm}(y,x)) \rightarrow \mathrm{cyn}(x) \wedge \exists y(\mathrm{phil}(y) \wedge \mathrm{adm}(y,x)))$

The result of this addition is a further jump in the complexity of reasoning.

Theorem 4. *The problem of determining the satisfiability of a set of sentences in Cop+Rel+TV is EXPTIME-complete.*

Proof: Essentially Pratt-Hartmann (forthcoming), Theorem 4.1. □

13.4 Adding ditransitive verbs

Let Cop+Rel+TV+DTV be the fragment defined by the grammar rules of Cop+Rel+TV, together with the following rules.

Syntax	Content Lexicon	
VP → DTV, NP, to, NP	DTV → prefers	...

For example, Cop+Rel+TV+DTV contains the following sentence, and assigns it the corresponding logical translation.

Every stoic whom no sceptic prefers any cynic to admires every philosopher

$$\forall x(\mathrm{sto}(x) \wedge \forall y(\mathrm{sce}(y) \rightarrow \neg\exists z(\mathrm{cyn}(z) \wedge \mathrm{pref}(y,z,x))) \rightarrow \forall y(\mathrm{phil}(y) \rightarrow \mathrm{adm}(x,y)))$$

Theorem 5. *The problem of determining the satisfiability of a set of sentences in Cop+Rel+TV+DTV is NEXPTIME-complete.*

Proof: To show membership in NEXPTIME, let E be a finite set of Cop+Rel+TV+DTV-sentences. Let Φ be the first-order logic translations of E. Since the sentences in E contain only one main verb, and since all noun-phrases in this fragment translate to expressions with only one (λ-bound) variable, it is easy to transform Φ polynomially into an equisatisfiable set of clauses $\mathscr{C}$ such that every $C \in \mathscr{C}$ contains at most one non-unary literal. Now let $\mathscr{D}$ be the set of clauses defined exactly as in the proof of Theorem 3, so that $\mathscr{C}$ and $\mathscr{D}$ are equisatisfiable with $|\mathscr{D}| \leq |\mathscr{C}| + |\mathscr{C}|^2$. Thus, it suffices to show that the satisfiability of $\mathscr{D}$ can be decided in nondeterministic exponential time.

By construction, $\mathscr{D}$ involves only unary literals. And it is easy to check that, by splitting clauses if necessary, every clause $C \in \mathscr{D}$ has the properties:

P1: if C contains a ground literal, then C is ground;

P2: if a functional ground term occurs in C, then C is ground;

P3: C contains at most two variables;

P4: if C contains two variables x and y, then $d(C) = 1$; moreover, all binary function-symbols in C occur in atoms of the form $p(h(x,y))$, and C contains at least one such literal.

Now define the A-order $\prec$ as follows. Let $\prec^g$ be the ordering on ground atoms defined by setting $A \prec^g B$ iff A is a ground atom of the form $q(g(t))$ and B a ground atom of the form $p(f(t_1,t_2))$. Define $\prec^d$ on the set of ground atoms by setting $A \prec^d A'$ iff either (i) $d(A) < d(A')$ or (ii) $d(A) = d(A')$ and $A \prec^g A'$. Finally, define $\prec$ on the set of all atoms by setting $A \prec A'$ iff $A\theta \prec^d A'\theta$ for all ground substitutions θ. It is clear that $\prec$ is an A-order.

Let $C \vee L$ and $C' \vee L'$ be clauses satisfying **P1**–**P4** which resolve under the A-ordering $\prec$ with resolvent C''. It is routine to show that $d(C'') \leq \max(d(C), d(C'))$ and that furthermore, by applying the splitting rule to C'' if necessary, we obtain clauses also satisfying **P1**–**P4**. Since the number of such clauses is exponential in $|\mathscr{D}|$, saturation is reached, via exponentially many non-deterministic choice points, in exponentially many steps. Hence, the satisfiability problem for Cop+Rel+TV+DTV is in NEXPTIME.

We show next that satisfiability in Cop+Rel+TV+DTV is NEXPTIME-hard. To simplify the proof, we will allow ourselves to use the NPs something and everything in the fragment, with the obvious interpretation. (In fact, this facility is inessential.) Let us say that a formula is a *standard two-variable formula* if it can be written either as $\forall x \forall y \alpha$ or as $\forall x \exists y \alpha$, where α is a quantifier-free formula involving only unary predicates (and no functions or constants). It is well known that the satisfiability problem for a set of standard two-variable formulas is NEXPTIME-hard (Börger et al. 1997, pp. 253 ff). Any such problem can easily be reduced to the satisfiability problem for a set of clauses $\mathscr{C}$ of the following form:

$$\begin{array}{ll}
\neg p_{i1}(x) \vee \neg p_{i2}(y) \vee \neg p_{i3}(x,y) & (1 \leq i \leq n_1) \\
\neg q_{i1}(x,y) \vee \neg q_{i2}(x,y) \vee \neg q_{i3}(x,y) & (1 \leq i \leq n_2) \\
\neg r_i(x, f_i(x)) & (1 \leq i \leq n_3) \\
s_{i1}(x) \vee s_{i2}(x) & (1 \leq i \leq n_4) \\
t_{i1}(x,y) \vee t_{i2}(x,y) & (1 \leq i \leq n_5),
\end{array}$$

where $n_1, \ldots, n_5$ are non-negative integers, the subscripted p, q, r, s and t are predicates of the indicated arities, and the f_i $(1 \leq i \leq n_3)$ are *pairwise distinct* function-symbols. It follows that the satisfiability problem for such sets of clauses is also NEXPTIME-hard.

For every binary predicate p appearing in $\mathscr{C}$, let p^+ be a new unary predicate. In addition, let n be a new unary predicate, c_0 a new individual constant and $\oplus$ a new binary function-symbol. (Think of $x \oplus y$ as denoting the ordered pair $\langle x,y \rangle$,

and think of $n(x)$ as stating that x is a 'normal' element—i.e. not an ordered pair.) Now let $\mathscr{D}$ be the corresponding set of clauses:

$$\begin{array}{ll}
\neg n(x) \vee \neg n(y) \vee \neg p_{i1}(x) \vee \neg p_{i2}(y) \vee \neg p^+_{i3}(x \oplus y) & (1 \leq i \leq n_1) \\
n(z) \vee \neg q^+_{i1}(z) \vee \neg q^+_{i2}(z) \vee \neg q^+_{i3}(z) & (1 \leq i \leq n_2) \\
\neg n(x) \vee \neg r^+_i(x \oplus f_i(x)) & (1 \leq i \leq n_3) \\
\neg n(x) \vee n(f_i(x)) & (1 \leq i \leq n_3) \\
\neg n(x) \vee s_{i1}(x) \vee s_{i2}(x) & (1 \leq i \leq n_4) \\
t^+_{i1}(z) \vee t^+_{i2}(z) & (1 \leq i \leq n_5) \\
\neg n(x) \vee \neg n(y) \vee \neg n(x \oplus y), \qquad n(c_0). &
\end{array}$$

It is routine to show that $\mathscr{C}$ and $\mathscr{D}$ are equisatisfiable. Now let us further transform the clause-set $\mathscr{D}$. For each i ($1 \leq i \leq n_2$), let q^+_{i12} be a new unary predicate and let d be a new ternary predicate. (Think of of $q^+_{i12}(x)$ as standing for $q^+_{i1}(x) \wedge q^+_{i2}(x)$.) Now let $\mathscr{E}$ be the corresponding set of clauses:

$$\begin{array}{ll}
\neg n(x) \vee \neg n(y) \vee \neg p_{i1}(x) \vee \neg p_{i2}(y) \vee \neg p^+_{i3}(z) \vee d(x,y,z) & (1 \leq i \leq n_1) \\
n(z) \vee \neg q^+_{i12}(z) \vee \neg q^+_{i3}(z) & (1 \leq i \leq n_2) \\
\neg q^+_{i1}(z) \vee \neg q^+_{i2}(z) \vee q^+_{i12}(z) & (1 \leq i \leq n_2) \\
\neg n(x) \vee \neg r^+_i(z) \vee d(x, f_i(x), z) & (1 \leq i \leq n_3) \\
\neg n(x) \vee n(f_i(x)) & (1 \leq i \leq n_3) \\
\neg n(x) \vee s_{i1}(x) \vee s_{i2}(x) & (1 \leq i \leq n_4) \\
t^+_{i1}(z) \vee t^+_{i2}(z) & (1 \leq i \leq n_5) \\
\neg n(x) \vee \neg n(y) \vee \neg n(z) \vee d(x,y,z), \qquad n(c_0) & \\
\neg n(x) \vee \neg n(y) \vee \neg d(x,y,x \oplus y). &
\end{array}$$

Again, the sets $\mathscr{D}$ and $\mathscr{E}$ are easily seen to be equisatisfiable. Finally, the set E of Cop+Rel+TV+DTV-sentences

$$\begin{array}{ll}
\text{Every p}_{i1} \text{ which is an n ds every p}_{i2} \text{ which is an n} & \\
\qquad \text{to every p}^+_{i3} & (1 \leq i \leq n_1) \\
\text{Every q}^+_{i12} \text{ which is a q}^+_{i3} \text{ is an n} & (1 \leq i \leq n_2) \\
\text{Every q}^+_{i1} \text{ which is a q}^+_{i2} \text{ is a q}^+_{i12} & (1 \leq i \leq n_2) \\
\text{Every n ds some n to every r}^+_i & (1 \leq i \leq n_3) \\
\text{Every n which is not an s}_{i1} \text{ is an s}_{i2} & (1 \leq i \leq n_4) \\
\text{Everything which is not a t}^+_{i1} \text{ is a t}^+_{i2} & (1 \leq i \leq n_4) \\
\text{Something is an n} & \\
\text{Every n ds every n to every n} & \\
\text{No n ds any n to everything.} &
\end{array}$$

translates to formulas which clausify to $\mathscr{E}$, up to renaming of of predicates and Skolem functions. (At this point we use the assumption that the f_i are pairwise distinct.) The NEXPTIME-hardness of Cop+Rel+TV+DTV then follows. □

13.5 Anaphora

A more radical addition to the fragments considered above concerns pronouns and reflexives. Thus, for example, we might add to Cop+Rel+TV the following rules.

Syntax	Formal lexicon
NP $\rightarrow$ Reflexive	Reflexive $\rightarrow$ itself (him/herself)
NP $\rightarrow$ Pronoun	Pronoun $\rightarrow$ it (he/she/him/her)

For simplicity, we shall always take pronouns and reflexives to have antecedents in the sentences in which they occur. That is to say: all anaphora is intra-sentential. Of course, we also assume the selection of such antecedents to be subject to the usual rules of binding theory, which, again, we need not rehearse here. Providing a formal semantics for pronouns and anaphora is rather more complicated than for the fragments considered above; however, from a complexity-theoretic point of view, these details may be safely ignored.

It is easy to see that adding the above grammar rules results in anaphoric ambiguities. For example, in the sentence

Every philosopher who admires a cynic despises every stoic who castigates him,

the pronoun may take as antecedent either the NP headed by philosopher or the NP headed by cynic. (The NP headed by stoic is not available as a pronoun antecedent here.) We then have two options: either we resolve such ambiguities by fiat, or we decorate nouns and pronouns in these sentences with indices to record which pronouns take which NPs as antecedents.

Considering the former option, let the fragment Cop+Rel+TV+RA (RA for "restricted anaphora") be the fragment defined by the grammar rules for Cop+Rel+TV together with the above rules for reflexives and pronouns, subject to the restriction that *pronouns must take their closest allowed antecedents.* Here, *closest* means "closest measured along edges of the phrase-structure" and *allowed* means "allowed by the principles of binding theory". (We ignore case and gender agreement.) It turns out that this fragment corresponds closely to the two-variable fragment of first-order logic. Because of this correspondence, we have:

Theorem 6. *The problem of determining the satisfiability of a set of sentences in Cop+Rel+TV+RA is NEXPTIME-complete.*

Proof: See Pratt-Hartmann (2003), Corollaries 1 and 2. □

Turning attention now to the latter option for dealing with anaphoric ambiguity, let Cop+Rel+TV+GA (GA for "general anaphora") be the fragment defined by the

Fragment	Complexity of satisfiability
Cop	P
Cop+Rel	NP-complete
Cop+TV	P
Cop+Rel+TV	EXPTIME-complete
Cop+Rel+TV+DTV	NEXPTIME-complete
Cop+Rel+TV+RA	NEXPTIME-complete
Cop+Rel+TV+GA	undecidable

Table 13.1: Lattice of fragments of English and their complexity classes

grammar rules for Cop+Rel+TV together with the above rules for reflexives and pronouns, with anaphoric antecedents indicated by coindexing in the usual way, subject only to the usual rules of binding theory.

Theorem 7. *The problem of determining the satisfiability of a set of sentences in Cop+Rel+TV+GA is undecidable.*

Proof: See Pratt-Hartmann (2003), Theorem 5. □

13.6 Conclusions and relation to other work

This paper has extended results obtained in Pratt-Hartmann (2003, forthcoming), where the general programme of determining the semantic complexity of fragments of natural languages is outlined. The new results reported here concern the complexity of the fragments Cop+TV and Cop+Rel+TV+DTV. The complexity of satisfiability for all of the fragments considered above is shown in Table 13.1.

Several authors have proposed formalisms based on logical constructions inspired by the syntax of natural language, apparently in the belief that such formalisms increase inferential efficiency. These include Fitch (1973), Suppes (1979), Purdy (1991) and 'traditional' logicians such as Englebretsen (1981) and Sommers (1982). But none of these analyses yields any immediate complexity-theoretic consequences of the sorts reported here. On the other hand, McAllester and Givan (1992) present a natural-language-inspired formal language with NP-complete satisfiability (PTIME in certain cases). However, this formal language is not shown to be generated by any linguistically natural fragment such as those considered above. More recently, Purdy (1996, 1999, 2002) has analysed the *fluted fragment* of first-order logic, alleging some (not completely specified) affinity between this

fragment and quantification in natural language. We note in this regard that the fragment Cop+Rel+TV+ DTV does not translate into the fluted fragment.[1]

Bibliography

Börger, E., E. Grädel, and Y. Gurevich (1997). *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin.

Englebretsen, G. (1981). *Three Logicians*. Van Gorcum, Assen.

Fitch, F. B. (1973). Natural deduction rules for English. *Philosophical Studies*, **24**:89–104.

Leitsch, A. (1997). *The Resolution Calculus*. Springer, Berlin.

McAllester, D. A. and R. Givan (1992). Natural language syntax and first-order inference. *Artificial Intelligence*, **56**:1–20.

Pratt-Hartmann, I. (2003). A two-variable fragment of English. *Journal of Logic, Language and Information*, **12**:13–45.

Pratt-Hartmann, I. (forthcoming). Fragments of language. *Language and Computation*.

Purdy, W. C. (1991). A logic for natural language. *Notre Dame Journal of Formal Logic*, **32**(3):409–425.

Purdy, W. C. (1996). Fluted formulas and the limits of decidability. *Journal of Symbolic Logic*, **61**(2):608–620.

Purdy, W. C. (1999). Quine's 'limits of decision'. *Journal of Symbolic Logic*, **64**(4):1439–1466.

Purdy, W. C. (2002). The complexity and nicety of fluted logic. *Studia Logica*, **71**:177–198.

Sommers, F. (1982). *The Logic of Natural Language*. Clarendon Press, Oxford.

Suppes, P. (1979). Logical inference in English: a preliminary analysis. *Studia Logica*, **38**:375–391.

[1]The author wishes to thank Allan Third and Mark Steedman for their comments on the manuscript, and gratefully acknowledges the support of the EPSRC (grant reference GR/S22509) and the hospitality of the University of Edinburgh Institute for Communicating and Collaborative Systems.

Chapter 14

Word Vectors and Quantum Logic
Experiments with negation and disjunction

DOMINIC WIDDOWS*, STANLEY PETERS†

A calculus which combined the flexible geometric structure of vector models with the crisp efficiency of Boolean logic would be extremely beneficial for modelling natural language. With this goal in mind, we present a formulation for logical connectives in vector spaces based on standard linear algebra, giving examples of the use of vector negation to discriminate between different senses of ambiguous words. It turns out that the operators developed in this way are precisely the connectives of quantum logic (Birkhoff and von Neumann, 1936), which to our knowledge have not been exploited before in natural language processing. In quantum logic, arbitrary sets are replaced by linear subspaces of a vector space, and set unions, intersections and complements are replaced by vector sum, intersection and orthogonal complements of subspaces. We demonstrate that these logical connectives (particularly the orthogonal complement for negation) are powerful tools for exploring and analysing word meanings and show distinct advantages over Boolean operators in document retrieval experiments.

This paper is organised as follows. In Section 14.1 we describe some of the ways vectors have been used to represent the meanings of terms and documents in natural language processing, and describe the way the WORD-SPACE used in our later experiments is built automatically from text corpora. In Section 14.2 we define the logical connectives on vector spaces, focussing particularly on negation and disjunction. This introduces the basic material needed to understand the worked examples given in Section 14.3, and the document retrieval experiments described in Section 14.3.1. Section 14.4 gives a much fuller outline of the theory of quantum logic, the natural setting for the operators of Section 14.2. Finally,

*CSLI, Stanford University; dwiddows@csli.stanford.edu.
†Linguistics & CSLI, Stanford University; peters@csli.stanford.edu.

Proceedings of Mathematics of Language 8
R. T. Oehrle & J. Rogers (editors).

in Section 14.5, we examine the similarities between quantum logic and WORD-SPACE, asking whether quantum logic is an appropriate framework for modelling word-meanings or if the initial successes we have obtained are mainly coincidental.

To some extent, this paper may have been written backwards, in that the implementation and examples are at the beginning and most of the theory is at the end. This is for two reasons. Firstly, we hoped to make the paper as accessible as possible and were afraid that beginning with an introduction to the full machinery of quantum logic would defeat this goal before the reader has a chance to realise that the techniques and equations used in this work are really quite elementary. Secondly, the link with 'quantum logic' was itself only brought to our attention *after* the bulk of the results in this paper had been obtained, and since this research is very much ongoing, we deemed it appropriate to give an honest account of its history and current state.

14.1 Representing word-meaning in vector spaces

A vector space is a collection of points each of which can be specified by a list of co-ordinates (such as the familiar *x*-*y* co-ordinates in Cartesian geometry) (Jänich, 1994, Ch 2), where pairs of points can be added together by adding their co-ordinates, and an individual point can be multiplied by a 'scalar' or number (in this paper, these scalars are real numbers, so all of our vector spaces are 'real' vector spaces). The first linguistic examples of vector spaces were developed for information retrieval (Salton and McGill, 1983), where counting the number of times each word occurs in each document gives a *term-document matrix*, where the i, j^{th} matrix entry records the number of times the word w_i occurs in the document D_j. The rows of this matrix can then be thought of as *word-vectors*. The dimension of this vector space (the number of co-ordinates given to each word) is therefore equal to the number of documents in the collection. *Document vectors* are generated by computing a (weighted) sum of the word-vectors of the words appearing in a given document.

Such techniques are used in information retrieval to measure the similarity between words (or more general query statements) and documents, using a similarity measure such as the cosine of the angle between two vectors (Salton and McGill, 1983, p 121),

$$\mathrm{sim}(w,d) = \frac{\sum w_i d_i}{\sqrt{\sum w_i^2 \sum d_i^2}} = \frac{w \cdot d}{||w||\,||d||},$$

where w_i, d_i are the co-ordinates of the vectors w and d, $w \cdot d$ is the (Euclidean) scalar product of w and d, and $||w||$ is the norm of the vector w (Jänich, 1994, Ch

8). This calculation is simplified further by normalising all vectors to have unit length, so that the 'cosine similarity' is the same as the Euclidean scalar product. This is a standard technique which (for example) avoids giving too much semantic significance to frequent terms or long documents. Normalised vectors were used in all of the models and experiments described in this paper.

A natural advantage of this structure is that it can be used to define a similarity score between pairs of terms in exactly the same way — two terms will have a high similarity score if they often occur in the same documents, and only seldom occur without one another. In general, several terms are combined into a combined query-statement using commutative vector addition (though the fuzzy-set and *p*-norm operations of (Salton et al., 1983) give more sophisticated models for conjunction and disjunction which also combine some of the benefits of Boolean and vector approaches).

Typically, such term-document matrices are extremely sparse. The information can be concentrated in a smaller number of dimensions using (among other dimension reduction algorithms) singular value decomposition, projecting each word onto the *n*-dimensional subspace which gives the best least-squares approximation to the original data. This represents each word using the *n* most significant 'latent variables', and for this reason this process is called *latent semantic analysis* (Landauer and Dumais, 1997). A variant of latent semantic analysis was developed by Schütze (1998) specifically for the purpose of measuring semantic similarity between words. Instead of using the documents as column labels for the matrix, semantically significant *content-bearing words* are used, and other words in the vocabulary are given a score each time they occur within a context window of (*eg.*) 15 words of one of these content-bearing words. Thus the vector of the word *football* is determined by the fact that it frequently appears near the words *sport* and *play*, *etc*. This method has been found to be well-suited for semantic tasks such as word-sense clustering and disambiguation. Such a vector space where points are used to represent words and concepts is sometimes called a WORD-SPACE (Schütze, 1998). The examples and experiments described in this article use exactly this sort of WORD-SPACE, using the Euclidean scalar product on normalised vectors to compute similarity.

Traditional approaches to semantics using set theory and Boolean logic are well-adapted for arranging primitives into composite propositions but have little to say on the meaning of those primitives[1]. The vector models described in this section, by contrast, have plenty to say about the meaning of the primitive units,

[1]Typical analyses (eg. (Partee et al., 1993, Ch 13)) give lambda calculus such as $\lambda x \lambda y.loves(x,y)$ for the meaning of the predicate 'loves', but are content to say that the semantics of 'John' is given by `john` or *j* and the semantics of 'Mary' is given by `mary` or *m*.

but only limited means to infer the meaning of sentences from these units. We would ideally, of course, have the best of both worlds.

14.2 Logical Connectives in WORD-SPACE

In this section we introduce logical connectives which can be used to explore meanings of terms in WORD-SPACE. In particular, we define negation in terms of orthogonality and disjunction in terms of the vector sum of subspaces. A more thorough discussion of the logic behind these operations is given in Section 14.4.

Vector Negation

We want to model the meaning of a statement like 'rock NOT band' in such a way that the system realises we are interested in the geological, not the musical meaning of the word *rock*. This involves finding which aspects of the meaning of *rock* which are different from, and preferably unrelated to, those of *band*. Meanings are unrelated to one another if they have no features in common at all, just as a document is regarded as completely irrelevant to a user if its scalar product with the user's query is zero — precisely when the query vector and the document vector are orthogonal (Jänich, 1994, §8.2) [2]. Our definition of negation for vectors relies on precisely this correspondence between the notions of 'irrelevant' and 'orthogonal in WORD-SPACE'.

Definition 1. *Two words a and b are considered irrelevant to one another if their vectors are orthogonal, i.e. a and b are mutually irrelevant if* $a \cdot b = 0$.

The statement '*a* NOT *b*' is now interpreted as 'those features of *a* to which *b* is irrelevant'.

Definition 2. *Let V be a vector space equipped with a scalar product. For a vector subspace* $A \subseteq V$*, define the orthogonal subspace* $A^{\perp}$ *to be the subspace*

$$A^{\perp} \equiv \{v \in V : \forall a \in A, a \cdot v = 0\}.$$

Let A and B be subspaces of V. By NOT *B we mean* $B^{\perp}$ *and by A* NOT *B we mean the projection of A onto* $B^{\perp}$.

Let $a, b \in V$*. By a* NOT *b we mean the projection of a onto* $\langle b \rangle^{\perp}$*, where* $\langle b \rangle$ *is the subspace* $\{\lambda b : \lambda \in \mathbb{R}\}$.

[2] This idea of negation as 'otherness' is found in Plato's *Sophist* dialogue (Horn, 2001, p. 1).

We now show how to use these notions to perform simple calculations with individual vectors in WORD-SPACE, using a standard projection mapping technique (Jänich, 1994, §8.2).

Theorem 1. *Let $a, b \in V$. Then a* NOT *b is represented by the vector*

$$a \text{ NOT } b \equiv a - \frac{a \cdot b}{|b|^2} b.$$

where $|b|^2 = b \cdot b$ is the norm of b.

Proof. Taking scalar product with b, we have that

$$\begin{aligned} (a \text{ NOT } b) \cdot b &= (a - \frac{a \cdot b}{|b|^2} b) \cdot b \\ &= a \cdot b - \frac{(a \cdot b)\ (b \cdot b)}{b \cdot b} \\ &= 0. \end{aligned}$$

This shows that a NOT b and b are orthogonal, so the vector a NOT b is precisely the part of a which is irrelevant to b (in the sense of Definition 1) as desired. ∎

For normalised vectors, Theorem 1 takes the particularly simple form

$$a \text{ NOT } b = a - (a \cdot b) b.$$

In practice this vector is then renormalised for consistency. As well as being well-motivated theoretically, this expression for negation is computationally extremely efficient to implement in the 'search phase' of a retrieval system. In order to find terms or documents that are closely related to a NOT b, it is not necessary to compare each candidate with both a and b and then compute some difference. Theorem 1 gives a single vector for a NOT b, so finding the similarity between any other vector and a NOT b is just a single scalar product computation.

Vector Disjunction and Conjunction

Modelling disjunctive expressions (such as A OR B) works similarly. Disjunction in set theory is modelled as the union of sets, which corresponds in linear algebra to the vector sum of subspaces, since $A + B$ is the smallest subspace of V containing both A and B.

Definition 3. *Let $b_1 \ldots b_n \in V$. The expression b_1* OR *$\ldots$* OR *b_n is represented by the subspace*

$$B = \{\lambda_1 b_1 + \ldots + \lambda_n b_n : \lambda_i \in \mathbb{R}\}.$$

Finding the similarity between an individual term a and a general subspace B is more complicated than finding the similarity between individual terms. It makes sense to define

(14.2.1) $\text{sim}(a, B) = a \cdot P_B(a)$

that is, the scalar product of a with the *projection* of a onto the subspace B, since this measures the magnitude of the component of a which lies in the subspace B.

To find this similarity in practice, it is not correct simply to compute $\text{sim}(a, b_j)$ for each of the vectors b_j in turn unless the set $\{b_j\}$ is orthonormal *i.e.* the vectors are pairwise orthogonal and of unit length (Jänich, 1994, p 139) (this is very unlikely). Instead, an orthonormal basis for B must first be constructed, a process which can be accomplished in practice by first using the Gram-Schmidt process (Jänich, 1994, p 142) to obtain an orthonormal basis $\{\tilde{b}_j\}$ for the subspace B. Once this is accomplished, it follows that

$$P_B(a) = \sum_j (a \cdot \tilde{b}_j)\tilde{b}_j$$

so that $\text{sim}(a, B) = \sum_j (a \cdot \tilde{b}_j)$. To compute $\text{sim}(a, B)$ we need to take the scalar product of a with *each* of the vectors $\tilde{b}_j$, so this similarity is more expensive to compute than that given by Theorem 1. Thus the gain we get by comparing each document with the query a NOT b using only one scalar product operation is lost for disjunction, though we show later that this desirable property is recovered for *negated* disjunction.

Just as disjunction makes things more general, we would expect conjunction to make them more specific. Since our underlying WORD-SPACE is homogeneous (in the sense that any two non-zero points can be mapped to each other by a linear transformation), no one point is naturally any more or less general than any other. This is one of the noticeable drawbacks for the basic vector model generally: the terms *plant*, *fruit* and *apple* are all represented by single points without any notion of inclusion or inheritance. Ideally, this problem could be solved by having concepts represented not only by points but also by higher dimensional subspaces. Then *plant*, for example, could refer to a space with *fruit* as a subspace thereof, and with *apple* as an even smaller subspace or point in the *fruit* subspace. In theory it should be possible to build such a space using a taxonomy and corpus-data, though to our knowledge this has not been accomplished. Such a structure would present a natural model for conjunction: the conjunction of two subspaces would simply be their intersection. In the meantime, the intersection of distinct one-dimensional subspaces is always zero, so conjunction in this form is not a useful option.

suit		suit NOT lawsuit	
suit	1.000000	pants	0.810573
lawsuit	0.868791	shirt	0.807780
suits	0.807798	jacket	0.795674
plaintiff	0.717156	silk	0.781623
sued	0.706158	dress	0.778841
plaintiffs	0.697506	trousers	0.771312
suing	0.674661	sweater	0.765677
lawsuits	0.664649	wearing	0.764283
damages	0.660513	satin	0.761530
filed	0.655072	plaid	0.755880
behalf	0.650374	lace	0.755510
appeal	0.608732	worn	0.755260

Terms related to 'suit NOT lawsuit'

play		play NOT game	
play	1.000000	play	0.779183
playing	0.773676	playing	0.658680
plays	0.699858	role	0.594148
played	0.684860	plays	0.581623
game	0.626796	versatility	0.485053
offensively	0.597609	played	0.479669
defensively	0.546795	roles	0.470640
preseason	0.544166	solos	0.448625
midfield	0.540720	lalas	0.442326
role	0.535318	onstage	0.438302
tempo	0.504522	piano	0.438175
score	0.475698	tyrone	0.437917

Terms related to 'play NOT game'

Table 14.1: Examples of negation

14.3 Using negation to find word-senses

This section presents initial examples of our vector connectives which demonstrate the uses of vector negation, and of vector disjunction and negation together, to find vectors which represent different senses of ambiguous words. We briefly describe document retrieval experiments which show that vector negation has clear benefits over a traditional Boolean method, as shown in (Widdows, 2003b).

A WORD-SPACE model was built as described in Section 14.1 using the New York Times data from the LDC, a corpus consisting of *ca* 173 million words from news articles written between July 1994 and December 1996. As one might expect, news articles consistently prefer some meanings of ambiguous words over others: for example, the word *suit* is used far more often in a legal context than a clothing context. To test the effectiveness of our negation operator, we tried to find some of the less common meanings by removing words belonging to the more predominant meanings.

Table 14.1 shows that vector negation is very effective for removing the 'legal' meaning from the word *suit* and the 'sporting' meaning from the word *play*, leaving respectively the 'clothing' and 'performance' meanings. Note that removing a particular word also removes concepts related to the negated word. This gives credence to the claim that our mathematical model is removing the *meaning* of a word, rather than just a string of characters.

Vector negation and disjunction can be combined to remove several unwanted areas of meaning simultaneously. Suppose we negate not only one argument but several. If a user states that they want documents related to a but not $b_1, b_2, \ldots, b_n$, then (unless otherwise indicated) it is clear that they only want documents related to *none* of the unwanted terms b_i (rather than, say, the average of these terms). In

rock		rock NOT band		rock NOT band, arkansas	
rock	1.000000	rock	0.450473	rock	0.412383
band	0.892790	dubious	0.402324	stands	0.389242
band's	0.868856	arkansas	0.400669	celestial	0.387825
bands	0.867765	ark	0.392304	underground	0.381206
punk	0.861354	madison	0.378165	muck	0.376508
pop	0.848222	celestial	0.376519	touches	0.373402
guitar	0.840769	muck	0.367648	pure	0.373129
tunes	0.837099	sheds	0.363119	wind	0.373017
reggae	0.828602	whitewater	0.362743	echoes	0.360734
acoustic	0.820719	gore	0.360440	explosions	0.356637
blues	0.817073	wind	0.357299	beneath	0.355244
rockers	0.807684	majestic	0.355958	planet	0.354783

The word *rock* is most closely associated with pop music in the New York Times corpus. However, removing these meanings by negating the word *band* leaves a set of associations derived from the town *Little Rock, Arkansas*. (The word *little* is not indexed because it is regarded as too common and general to be a useful search term.) Removing *arkansas* as well gives meanings closely associated to rock as a geological material.

Table 14.2: Senses of *rock* in the New York Times

this way the expression

$$a \text{ AND } (\text{NOT } b_1) \text{ AND } (\text{NOT } b_2) \ldots \text{ AND } (\text{NOT } b_n)$$

becomes

(14.3.1) $a \text{ NOT } (b_1 \text{ OR } \ldots \text{ OR } b_n)$.

Using Definition 3 to model the disjunction b_1 OR $\ldots$ OR b_n as the vector subspace $B = \{\lambda_1 b_1 + \ldots + \lambda_n b_n : \lambda_i \in \mathbb{R}\}$, this expression can be assigned a unique vector which is orthogonal to *all* of the unwanted arguments $\{b_j\}$, this vector being $a - P_B(a)$, where P_B is the projection onto the subspace B just as in Equation 14.2.1. It follows that to compute the similarity between any vector and the expression a NOT $(b_1$ OR $\ldots$ OR $b_n)$ is again a *single* scalar product calculation, which gives the same computational efficiency as Theorem 1. This technique can be used to 'home in on' the desired meaning by systematically pruning away unwanted features (see Table 14.2).

14.3.1 Experiments with document retrieval

The effectiveness of vector negation and disjunction at removing unwanted concepts has been reliably demonstrated in document retrieval experiments, which are reported in much more detail in (Widdows, 2003b). In order to evaluate the effectiveness of different forms of negation, we used the hypothesis that a query for

term *a* NOT term *b*

should retrieve documents containing many occurences of term *a* and few occurences of term *b*. This can be accomplished trivially by first retrieving documents using the query 'term *a*' and then removing any documents that contain term *b*, in the traditional Boolean manner (Salton and McGill, 1983, p. 26). However, we also measured the occurence of synonyms and neighbours of the term *b*. Documents retrieved using vector negation contained far fewer of these than the Boolean method, which we believe to be strong evidence that vector negation removes not only unwanted words but unwanted *areas of meaning*.

14.4 Quantum Logic in Vector Spaces

A development that has recently come to our attention is that the logical operators on WORD-SPACE introduced in section 14.2 are precisely the connectives used in quantum logic. Quantum logic was formally introduced by Birkhoff and von Neumann (1936) as a framework in which to account for the observations and predictions of quantum mechanics, which exhibits some distinctly non-classical behaviour. A famous example is given by the two-slit experiment, in which patterns are observed which can not be accounted for by assuming that an electron must have passed through only one of the two slits (Putnam, 1976, p. 180). A much better approach is to model the emerging electron as a linear combination of states, assuming that the final description receives a contribution from each of the electron's possible routes. Classical logic has problems in this situation, because in set-theory if a is an element of the union $A \cup B$, it follows that at least one of the statements $a \in A$, $a \in B$ must hold — for example, where a represents the state of an electron which has passed through the two slits A and B then one of the statements "a passed through A" or "a passed through B" must hold.

Quantum logic solves this problem by describing the outcomes A and B not as arbitrary sets, but as subspaces of a vector space. Their disjunction is then their vector sum $A + B$, which is strictly larger than their set union $A \cup B$ unless $A \subseteq B$ or $B \subseteq A$. [3] Since there are many points $a \in A + B$ which are neither in A nor in B, the question "which slit did the electron a go through?" ceases to apply. Putnam (1976) contends that the differences between quantum logic and classical logic can account for all of the apparent 'difficulties' of quantum mechanics, and that we

[3] A simple way to envisage the difference between these two forms of disjunction is to consider the possible trajectories of a point which starts in the centre of a map with the instructions that it can travel in a North-South *or* an East-West direction. If this disjunction is interpreted classically, the particle can only travel to one of a 'cross-shape' of points which are either of the same latitude or of the same longitude as the starting point. In the disjunction is interpreted in the quantum framework, the point can travel anywhere that is a linear combination of north-south or east-west journeys, *i.e.* anywhere on the map.

should be prepared to change our view of 'logic' accordingly.

Philosophical issues aside, the structure of quantum logic itself is quite simple and is arrived at precisely by replacing the notions of sets and subsets with those of vector spaces and subspaces (Birkhoff and von Neumann, 1936, §6), (Cohen, 1989; Wilce, 2003; Putnam, 1976, p. 177). Events in quantum mechanics are represented by subspaces of a vector space V. [4] This leads us to consider the collection $L(V)$ of subspaces of vector space V, which is a partially ordered set under the inclusion relation, so that an event A implies an event B precisely when $A \leq B$.

The greatest lower bound or *meet* of $A, B \in L(V)$ is the greatest element $C \in L(V)$ such that $C \subseteq A$ and $C \subseteq B$, which is precisely the intersection $A \cap B$. The least upper bound or *join* of A and B is the smallest $D \in L(V)$ such that $A \subseteq D$ and $B \subseteq D$. However, the set union $A \cup B$ is not in general a member of $L(V)$, and the smallest member of $L(V)$ which contains this set is instead the linear span $A + B$. These two operations give the partially ordered set $L(V)$ the structure of a *lattice* (Birkhoff and von Neumann, 1936, §8), (Birkhoff, 1967), (Cohen, 1989, p. 35). Furthermore, because we are working in a space with an inner product, for each $A \in L(V)$ we can define its (unique) orthogonal complement $A^{\perp}$ just as in Definition 2. We now have three connectives on the lattice $L(V)$, defined as follows (Birkhoff and von Neumann, 1936, §1, §6) (Putnam, 1976, p. 178):

	Conjunction	A AND $B = A \cap B$
(14.4.1)	Disjunction	A OR $B = A + B$
	Negation	NOT $A = A^{\perp}$

It is simple to show that these connectives on $L(V)$ satisfy the necessary relations (such as, for example, $A + A^{\perp} = V$, $A \cap A^{\perp} = \{0 \in V\}$) to define a *logic* on $L(V)$. (Cohen, 1989, p. 36).

Another important equivalence is that each subspace $A \in L(V)$ can be identified (using the scalar product) with a unique projection map $P_A : V \to A$ (as in Theorem 1), and through this bijection the logic of subspaces $L(V)$ is equivalent to the logic of projection mappings on V. This logic plays a key role in quantum mechanics, where the image $P_A(V)$ of a point $v \in V$ under the projection P_A is used to measure the probability that a particle in the state represented by v will be found to have a physical property represented by the subspace A, using the scalar product $P_A(v) \cdot v$ as a probability measure (Wilce, 2003), just as in Equation 14.2.1.

[4] More precisely, quantum mechanics is usually modelled within a Hilbert space, which is a complete inner-product space (Cohen, 1989, 2.18). Every finite dimensional Euclidean space (and so every example of a WORD-SPACE with the Euclidean scalar product) is a Hilbert space, and so to avoid overly technical language we shall continue to talk about vector spaces rather than Hilbert spaces.

Quantum logic differs from classical Boolean logic in (at least) two well-known properties: quantum logic is neither distributive nor commutative. The distributive law

$$(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$$

is responsible for the question "which slit did the electron pass through", and so (as described above), quantum logic avoids this issue by avoiding the assumption that the electron *must* have passed entirely through either one of the slits. The commutative property fails because two projection mappings P_A and P_B do not in general commute. (An easy example is to consider the projections onto the x-axis and the line $y = x$ in the plane $\mathbb{R}^2$.) This is used to account for the fact that observations interfere with one another in quantum mechanics, which leads to Heisenberg's famous uncertainty principle (Birkhoff and von Neumann, 1936, §1). Measurements made by the projections P_A and P_B are said to be *compatible* if and only if P_A and P_B commute, which imposes particular conditions on the subspaces A and B (Cohen, 1989, p. 37).

14.5 Quantum Logic and WORD-SPACE — a fluke or a goldmine?

The reason for our interest in quantum logic is that we have already been using the quantum connectives on WORD-SPACE in Sections 14.2 and 14.3: the logical operations defined in Equations 2 and 3 are precisely the negation and disjunction connectives in Equation 14.4.1. This gives a much clearer account for some of the observations in Section 14.3. For example, the reformulation of the extended conjunction in Equation 14.3.1 follows immediately from knowing that the logic $L(V)$ satisfies the de Morgan laws (Cohen, 1989, p. 37), and it is precisely the non-commutativity of projection operators which forced us to first obtain an orthonormal basis for the subspace $(b_1$ OR $\ldots$ OR $b_n)$ in order to implement Equation 14.2.1.

This raises the question of whether quantum logic is a desirable framework for natural language semantics, or whether the links between quantum logic and concepts in WORD-SPACE are more accidental. The examples in Section 14.3, and in particular the retrieval experiments outlined in Section 14.3.1, demonstrate that the quantum connectives are at least very useful for manipulating word-meanings.

As models for composition of meaning, Boolean and quantum connectives seem to have different spheres of influence. One intuitive prediction, based on the mathematical models underlying the two frameworks, is that Boolean connectives

should be more appropriate for describing discrete entities, and quantum connectives should describe concepts which are more continuous. This prediction is borne out in at least some examples. If one's host for a dinner party said *"Please come at 7 or 7.30"*, one would expect 7.15 also to be a perfectly agreeable time to arrive, which would be false under a Boolean interpretation of *"7 or 7.30"* but true under a quantum interpretation. On the other hand, if upon arrival one was asked *"Would you like an apple or a plum?"* and responded positively, one would not really expect to be given a nectarine on the basis that nectarines are on a scale between apples and plums — here we are talking about discrete objects and it appears that a Boolean interpretation is appropriate. (In practice, there are many other factors to take into account — for example, in many day to day contexts (such as train timetables), a continuous variable becomes 'quantised' and interpretations change accordingly.) This discussion at the very least demonstrates that the differences between Boolean and vector connectives have linguistic significance beyond statistical word sense disambiguation and query generation for information retrieval.

One conceptual problem with the quantum disjunction operator is that in a WORD-SPACE of n-dimensions, n fairly similar concepts could be used to generate the whole space, provided they are linearly independent, leaving the possibility that the 'disjunctions' predicted by quantum logic may become far too general. Another problem with the 'linear span of the arguments' approach to disjunction is that it permits interpolation *and* extrapolation, where extrapolation may be inappropriate. For example, in the *"7 or 7.30"* example, we should not predict that 6 o'clock is also an acceptable arrival time. It follows that a better option might be to interpret a disjunction not as a linear subspace but as a *simplex* by adding the conditions $\lambda_i \geq 0$, $\sum_i \lambda_i = 1$ to Definition 3.

There are many apparent similarities between the historical debate over quantum and classical mechanics on the one hand, and the tension between 'symbolic' and 'statistical' approaches to natural language processing on the other. The vector model for information retrieval was first adopted largely because it allowed for a naturally continuous 'relevance score' rather than a simple dichotomy between relevance and irrelevance, in much the same way that quantum mechanics yields a probability that a particular event will be observed. The possible similarity between finding the 'state' of a particle through measurement and finding the 'sense' of an ambiguous word in context is raised in Widdows (2003a). More generally, quantum mechanics is possibly the single most successful scientific theory for making rigourous, testable predictions about systems where it is known that *exceptions are always a possibility*. That natural language bears the hallmarks of such a system is at the least plausible.

Demonstration

An interactive demonstration of word-similarity and negation in WORD-SPACE is publicly available at `http://infomap.stanford.edu/webdemo`.

Acknowledgements

The authors would particularly like to thank Larry Moss, who first suggested that the vector operators developed in this paper might be linked to quantum logic.

This research was supported in part by EC/NSF grant IST-1999-11438 for the MUCHMORE project and in part by the Research Collaboration between the NTT Communication Science Laboratories, Nippon Telegraph and Telephone Corporation and CSLI, Stanford University.

Bibliography

Birkhoff, G. (1967). *Lattice theory*. American Mathematical Society, 3rd edition. First edition 1940.

Birkhoff, G. and J. von Neumann (1936). The logic of quantum mechanics. *Annals of Mathematics*, **37**:823–843.

Cohen, D. (1989). *An introduction to Hilbert Spaces and Quantum Logic*. Sprige-Verlag.

Horn, L. (2001). *A Natural history of Negation*. CSLI publications, Stanford.

Jänich, K. (1994). *Linear algebra*. Undergraduate Texts in Mathematics. Springer-Verlag.

Landauer, T. and S. Dumais (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition. *Psychological Review*, **104**(2):211–240.

Partee, B. H., A. ter Meulen, and R. E. Wall (1993). *Mathematical Methods in Linguistics*. Kluwer.

Putnam, H. (1976). The logic of quantum maechanics. In *Mathematics, Matter and Method*, pp. 174–197. Cambridge University Press.

Salton, G., E. A. Fox, and H. Wu (1983). Extended boolean information retrieval. *Communications of the ACM*, **26**(11):1022–1036.

Salton, G. and M. McGill (1983). *Introduction to modern information retrieval*. McGraw-Hill, New York, NY.

Schütze, H. (1998). Automatic word sense discrimination. *Computational Linguistics,* **24**(1):97–124.

Widdows, D. (2003a). A mathematical model for context and word-meaning. In *(To appear in) Fourth International and Interdisciplinary Conference on Modeling and Using Context*. Stanford, California.

Widdows, D. (2003b). Orthogonal negation in vector spaces for modelling word-meanings and document retrieval. In *(To appear in) Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*. Sapporo, Japan.

Wilce, A. (2003). Quantum logic and probability theory. In E. N. Zalta, ed., *The Stanford Encyclopedia of Philosophy (Spring 2003 Edition)*.