•• IWPT'95 ••• IWPT'95 ••• IWPT'95 ••• IWPT'95 ••

# FOURTH INTERNATIONAL WORKSHOP ON PARSING TECHNOLOGIES

*Sponsored by ACL/SIGPARSE*

Prague and Karlovy Vary (Czech Republic)
September 20 – 24, 1995

•• IWPT'95 ••• IWPT'95 ••• IWPT'95 ••• IWPT'95 ••

# IWPT'95

• • •

# FOURTH INTERNATIONAL WORKSHOP ON PARSING TECHNOLOGIES

*Sponsored by ACL/SIGPARSE*

Prague and Karlovy Vary (Czech Republic)
September 20 – 24, 1995

• • •

## WORKSHOP PROCEEDINGS

The Proceedings are published by the Institute of Formal and Applied Linguistics, Charles University, Prague.
The workshop chair gratefully acknowledges the technical assistance of Barbora Hladká and Jan Hajič in the preparation of the volume.
Copies can be ordered at the address of the workshop secretariat.

# IWPT'95 Organization

| | |
|---|---|
| General Chairman: | Harry Bunt (Tilburg University) |
| Workshop Chair: | Eva Hajičová (Charles University, Prague) |
| Programme Committee: | Bernard Lang (chair)<br>Robert Berwick<br>Harry Bunt<br>Bob Carpenter<br>Ken Church<br>Eva Hajičová<br>Aravind Joshi<br>Ronald Kaplan<br>Martin Kay<br>Makoto Nagao<br>Anton Nijholt<br>Mark Steedman<br>Henry Thompson<br>Masaru Tomita<br>K. Vijay-Shanker<br>Yorick Wilks<br>Kent Wittenburg |
| Additional Referees: | Peter Asveld<br>Hugo ter Doest<br>Gerrit van der Hoeven<br>Sadao Kurohashi<br>Jan Rekers |
| Workshop Secretariat: | Libuše Brdičková<br>UFAL MFF UK<br>Charles University<br>Malostranské n. 25<br>11800 Prague 1<br>Czech Republic<br>phone: +42-2-2451 0286<br>fax: +42-2-532 742<br>email: {hajicova,brdickov}@ufal.mff.cuni.cz |

# Preface

In 1989, Masaru Tomita organized the *First International Workshop on Parsing Technologies (IWPT'89)* in Pittsburgh and Hidden Valley, Pennsylvania. This workshop was a great success. The contributed papers were of high quality and the unusual bilocation formula, with an emphasis on presentations during the first half of the workshop at the premises of Carnegie-Mellon University, and on discussions during the second half in a secluded conference resort, worked very well.

Inspired by this success, which lead to the book *Current Issues in Parsing Technologies* (Kluwer, Boston, 1991), the Second International Workshop on Parsing Technologies (IWPT'91) was organized by Masaru Tomita in Mexico. This workshop was successful too, although it suffered a little from travel restrictions related to the Gulf war. By then it seemed clear that IPWT'89 and IWPT'91 had established the beginning of a series of biannual workshops. Together, Masaru Tomita and I organized the third workshop in the series, IWPT'93, reusing the original '89 formula by having the first part of the workshop at Tilburg University (the Netherlands) and the second part in the small medieval town of Durbuy, in the Belgian Ardennes mountain range. This workshop has lead to the follow-up volume *Recent Advances in Parsing Technologies*, edited by Harry Bunt and Masaru Tomita, currently in production with by Kluwer Science Publishers.

By the time of IWPT'93 the Special Interest Group on Parsing *SIGPARSE* was set up within ACL, with Masaru Tomita and Harry Bunt as designated officers, and with the primary aim to give continuity to the IWPT series. During the preparation of IWPT'95, Masaru Tomita left the arena of natural language parsing and moved into biological engineering, applying and extending NL parsing techniques to DNA analysis. We are sorry that he has left the organization of the IWPT series, but we are happy to welcome him as invited speaker at IWPT'95 on the subject of DNA parsing.

On this occasion, I would like to thank the people primarily responsible for making IWPT'95 possible and, we hope, equally successful as its predecessors. Thanks go to the members of the Programme Committee, in particular to chairman Bernard Lang, for the careful work in reviewing submissions and deciding on the workshop programme, and to Eva Hajicova and her colleagues at Charles University in Prague, for making all the necessary bilocal (again!) arrangements in Prague and Karlovy Vary, sometimes with rather uncertain prospects, and for producing these proceedings.

Harry Bunt
IWPT'95 General Chairman

# Table of Contents

* *short paper (poster)*

# Author's Index

# Acyclic Context-sensitive Grammars

Erik Aarts*

Research Institute for
Language and Speech
Trans 10
3512 JK Utrecht
The Netherlands

Dept. of Mathematics
and Computer Science
Plantage Muidergracht 24
1018 TV Amsterdam
The Netherlands

**Abstract**

A grammar formalism is introduced that generates parse trees with crossing branches. The uniform recognition problem is NP-complete, but for any fixed grammar the recognition problem is polynomial.

## 1 Introduction

In this article we propose a new type of context-sensitive grammars, the *acyclic* context-sensitive grammars (ACSG's). Acyclic context-sensitive grammars are context-sensitive grammars with rules that have a "real rewrite part", which must be context-free and acyclic, and a "context part". The context is present on both sides of a rule. The motivation for introduction of ACSG's is that we want a formalism which

- allows parse trees with crossing branches
- is computationally tractable
- has a simple definition.

We enrich context-free rewrite rules with context and this leads to a simple definition of a formalism that generates parse trees with crossing branches. In order to gain efficiency, we add a restriction: the context-free rewrite part of the grammar must be acyclic. Without this restriction, the complexity would be the same as for unrestricted CSG's (PSPACE-complete). With the acyclicity restriction we get the same results as for *growing* CSG's (Dahlhaus and Warmuth 1986, Buntrock 1993), i.e., NP-completeness for uniform recognition and polynomial time for fixed grammars.

Possible applications are in the field of computational linguistics. In natural language one often finds sentences with so-called *discontinuous constituents* (constituents separated by other material). ACSG's can be used to describe such constructions. Most similar attempts (Pereira 1981, Johnson 1985, Bunt 1988, Abramson and Dahl 1989) allow an arbitrary distance between two parts of a discontinuous constituent. This is not allowed in ACSG's. For unbounded dependencies like *wh-movement* we either have to extend the formalism (allow arbitrary context) or introduce the *slash-feature*.

The acyclicity of the grammar does not seem to form a problem for the generative capacity necessary to describe natural language. Acyclicity is closely related to the *off-line parsability constraint* (Johnson 1988). Constituent structures satisfy the off-line parsability constraint iff

- it does not include a non-branching dominance chain in which the same category appears twice, and

*The author was sponsored by project NF 102/62-356 ('Structural and Semantic Parallels in Natural Languages and Programming Languages'), funded by the Netherlands Organization for the Advancement of Research (NWO).

- the empty string $\epsilon$ does not appear as the righthand side of any rule.

The off-line parsability constraint has been motivated both from the linguistic perspective and computationally. Kaplan and Bresnan (1982) say that "vacuously repetitive structures are without intuitive or empirical motivation" (Johnson 1988). ACSG's satisfy the off-line parsability constraint: they have no cycles and no $\epsilon$-rules.

The goal of designing a formalism that is computationally tractable is only achieved partially. We show that the uniform recognition problem is NP-complete. For any fixed grammar, however, the recognition problem is polynomial (in the sentence length).

The definition of ACSG is simple because it is a standard rewrite grammar. Derivability is defined by successive string replacement. This definition is, e.g., simpler than the definition of Discontinuous Phrase Structure Grammar (Bunt 1988). The main difference between DPSG and ACSG is that in ACSG constituents are "moved" when a context-sensitive rule is applied. In DPSG trees with crossing branches are described "static": the shape of a tree is described with node admissibility constraints.

The structure of this article is as follows. First we define acyclic CSG's, growing CSG's and quasi-growing CSG's formally. Then we present some results on the generative power of these classes of grammars and on their time complexity. We end with a discussion on the uniform recognition problem vs. the recognition problem for fixed grammars.

# 2 Definitions

## 2.1 Context-sensitive Grammars

**Definition 2.1** *A* grammar *is a quadruple* $\langle V, \Sigma, S, P \rangle$*, where* $V$ *is a finite set of symbols and* $\Sigma \subset V$ *is the set of terminal symbols.* $\Sigma$ *is also called the* alphabet. *The symbols in* $V \setminus \Sigma$ *are called the nonterminal symbols.* $S \in V \setminus \Sigma$ *is a start symbol and* $P$ *is a set of production rules of the form* $\alpha \rightarrow \beta$*, with* $\alpha, \beta \in V^*$*, where* $\alpha$ *contains at least one nonterminal symbol.*

**Definition 2.2** *A grammar is* context-free *if each rule is of the form* $Z \rightarrow \gamma$ *where* $Z \in V \setminus \Sigma$*;* $\gamma \in V^*$*. A* cycle *in a context-free grammar is a set of symbols* $a_1, \ldots, a_n$ *with* $a_i \rightarrow a_{i+1} \in P$ *for all* $1 \leq i < n$ *and* $a_1 = a_n$*.*

**Definition 2.3** *A grammar is* context-sensitive *if all rules are of the form* $\alpha \rightarrow \beta$*, with* $|\alpha| \leq |\beta|$*.*

**Definition 2.4** Derivability *(*$\Rightarrow$*) between strings is defined as follows:* $u\alpha v \Rightarrow u\beta v$ $(u, v, \alpha, \beta \in V^*)$ *iff* $(\alpha, \beta) \in P$*. The transitive closure of* $\Rightarrow$ *is denoted by* $\stackrel{+}{\Rightarrow}$*. The reflexive transitive closure of* $\Rightarrow$ *is denoted by* $\stackrel{*}{\Rightarrow}$*.*

**Definition 2.5** *The* language *generated by* $G$ *is defined as* $L(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} w\}$*.*

**Definition 2.6** *A derivation of a string* $\delta$ *is a sequence of strings* $x_1, x_2, \ldots, x_n$ *with* $x_1 = S$*, for all* $i$ $(1 \leq i < n)$ $x_i \Rightarrow x_{i+1}$ *and* $x_n = \delta$*.*

## 2.2 Labeled Context-sensitive Grammars

Context-sensitive grammars have just been described as grammars with rules of the form $\alpha \rightarrow \beta$ for which $|\alpha| \leq |\beta|$. There is an alternative definition where *context* is used. In this definition, rules are of the form $\alpha Z \beta \rightarrow \alpha\gamma\beta$ $(Z \in V, \alpha, \beta, \gamma \in V^*)$. In this format, $\alpha$ and $\beta$ are *context*, and $Z \rightarrow \gamma$ is called the *context-free backbone*. It is known that the two formats are weakly equivalent (Salomaa 1973, pp. 15,82). We introduce here a third form, which is a kind of a mix between the other two. Instead of having context on the outside of the rule, we can also have context inside. Suppose we have the rule "A rewrites to B C in the context D". In a standard context-sensitive grammar, this can only be expressed as D A $\rightarrow$ D B C or as A D $\rightarrow$ B C D. We are going to allow that the D is in between the B and the C. A possible rule is D A $\rightarrow$ B D C. In the rules of the form $\alpha Z \beta \rightarrow \alpha\gamma\beta$ it is not always clear which symbols are the context and which symbols form the context-free backbone. E.g., in the grammar rule

A A $\rightarrow$ A B A it is not clear what the context is. We are going to indicate the context with brackets. The rule A A $\rightarrow$ A B A can be written as [A] A $\rightarrow$ [A] B A or as A [A] $\rightarrow$ A B [A] (the context is between brackets). In the new form, where the context can be "scattered", brackets are not enough. Therefore we introduce labels for the context symbols. The labels left and right of the arrow must be the same of course. The rule [A] A $\rightarrow$ [A] B A is written now as $A_1$ A $\rightarrow$ $A_1$ B A and A [A] $\rightarrow$ A B [A] is written as A $A_1$ $\rightarrow$ A B $A_1$. The rule D A $\rightarrow$ B D C can be written as $D_1$ A $\rightarrow$ B $D_1$ C.

This can be formalized as follows.

**Definition 2.7** *An* acyclic context-sensitive grammar *$G$ is a quadruple $\langle V, \Sigma, S, P\rangle$, where $V$ and $\Sigma$ are, again, sets of symbols and terminal symbols. $V_l$ and $\Sigma_l$ (labeled symbols and terminals) denote $\{\langle a, k\rangle \mid a \in V, 1 \leq k \leq K\}$ and $\{\langle a, k\rangle \mid a \in \Sigma, 1 \leq k \leq K\}$ respectively, where $K$ depends on the particular grammar under consideration. $S \in V \setminus \Sigma$ is a start symbol and $P$ is a set of production rules of the form $\alpha \rightarrow \beta$, with $\alpha \in V_l^* V V_l^*, \beta \in (V \cup V_l)^*$. The left hand side contains exactly one unlabeled symbol, the right hand side at least one. For all production rules it holds that $|\alpha| \leq |\beta|$. The labeled symbols in a rule are called the context.*

*There are three conditions:*

- *If we leave out all members of $V_l$ (the context) from the production rules we obtain a context-free grammar. This is the context-free backbone.*
- *If we leave out all members of $V$ (the backbone) from the production rules we obtain a grammar that has permutations only. This is the context part. All context symbols in the rules should have different labels.*
- *If we remove all labels in the rules, i.e., we replace all symbols $\langle a, b\rangle$ by $a$, we get an ordinary context-sensitive grammar $G'$. We define that $L(G) = L(G')$.*

## 2.3 Acyclic Context-sensitive Grammars

Acyclic context-sensitive grammars, or ACSG's, are context-free grammars with an "acyclic contextfree backbone". This is formalized as follows.

**Definition 2.8** *An* acyclic context-sensitive grammar *is a labeled context-sensitive grammar that fullfills the following condition:*

- *If we leave out all members of $V_l$ from the production rules we must obtain a finitely ambiguous context-free grammar, i.e. a grammar for which $|\alpha| = 1$ and $|\beta| \geq 1$ and that contains no cycles.*

An example of a rule of an acyclic CSG is (pairs of $\langle N, L\rangle$ are written as $N_L$):

$$A_1\ B_2\ M\ C_3\ D_4 \rightarrow C_3\ K\ B_2\ A_1\ L\ M\ D_4$$

The context-free backbone is M $\rightarrow$ K L M. The context part is $A_1$ $B_2$ $C_3$ $D_4$ $\rightarrow$ $C_3$ $B_2$ $A_1$ $D_4$. The rule without labels is A B M C D $\rightarrow$ C K B A L M D. Another example is given after the definition of growing context-sensitive grammars.

## 2.4 Growing Context-sensitive Grammars

The definition of growing CSG's (GCSG's) is pretty simple: the lefthand side of a rule must be shorter than the righthand side. For the precise definition we follow Buntrock (1993):

**Definition 2.9** *A context-sensitive grammar $G = \langle V, \Sigma, S, P\rangle$ is growing if*

1. $\forall(\alpha \rightarrow \beta) \in P : \alpha \neq S \Rightarrow |\alpha| < |\beta|$, *and*
2. *$S$ does not appear on the right hand side of any rule.*

We also define grammars which are growing with respect to a weight function. These were introduced in (Buntrock 1993).

**Definition 2.10** *We call a function* $f : \Sigma^* \to N$ *a* weight function *if* $\forall a \in \Sigma : f(a) > 0$ *and* $\forall w, v \in \Sigma^* : f(w) + f(v) = f(wv)$.

Now we can define *quasi-growing* grammars:

**Definition 2.11** *Let* $G = \langle V, \Sigma, S, P \rangle$ *be a grammar and* $f : V^* \to N$ *a weight function. We call* $G$ *quasi-growing if* $f(\alpha) < f(\beta)$ *for all productions* $(\alpha \to \beta) \in P$.

Quasi-growing context-sensitive grammars (QGCSG's) are grammars that are both quasi-growing and context-sensitive.

# 3 An Example

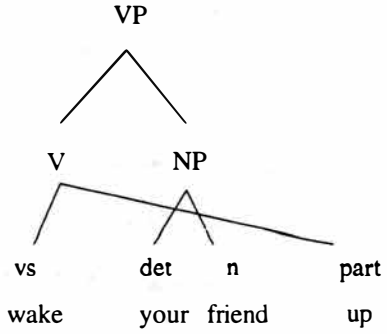
This section contains an example based on natural language of how ACSG's work. It is taken from Bunt (1988). Assume we have the following grammar. It has one rule that is not context-free.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| VP | → | V | NP | | vs | → | wake |
| $V\ NP_1$ | → | vs | $NP_1$ | part | det | → | your |
| NP | → | det | n | | n | → | friend |
| | | | | | part | → | up |

Table 1: An example grammar

Suppose we have the sentence: "Wake your friend up". This is a VP:
VP ⇒ V NP ⇒ vs NP part ⇒ vs det n part ⇒* wake your friend up

The corresponding parse tree is:




We see that we have a context-free backbone which allows us to draw parse trees (or structure trees) and that the scattered context causes branches in the trees to cross.

# 4 Properties of Acyclic Context-sensitive Grammars

## 4.1 Generative Power of Acyclic CSG's

In this section we discuss the relation ACSG's between and GCSG's and we sketch their position in the Chomsky hierarchy.

**Theorem 4.1** *$\epsilon$-free CFL $\subset$ ACSL*

*Proof:* if the cfl is generated by an acyclic cfg without empty productions we do not have to do anything. This cfg *is* an acyclic csg. If the cfg contains cycles we can remove them. A cycle can be removed by introduction of a new symbol. This symbol rewrites to any member of the cycle. Any cfg with empty productions can be changed into a cfg without empty productions that generates the same language. There's one exception here: languages containing the empty string can not be generated. Therefore acyclic context-sensitive grammars generate all cfl's that do not contain the empty word. □

**Theorem 4.2** *ACSL* $\neq$ *CFL*

*Proof:* ACSG's are able to generate languages that are not context-free. One example is the language $\{a^n b^{2^n} c^n \mid n \geq 1\}$. This language is generated by the grammar:

| | | | | | |
|---|---|---|---|---|---|
| S | $\rightarrow$ | A B B C | A | $\rightarrow$ | a |
| B $X_1$ | $\rightarrow$ | $X_1$ B B | B | $\rightarrow$ | b |
| B $C_1$ | $\rightarrow$ | X B B C $C_1$ | C | $\rightarrow$ | c |
| $A_1$ X $B_2$ | $\rightarrow$ | $A_1$ A $B_2$ | | | |

Table 2: Grammar for $\{a^n b^{2^n} c^n \mid n \geq 1\}$

A derivation of " a a b b b b c c " is:

S $\Rightarrow$ A B B C $\Rightarrow$ A B X B B C C $\Rightarrow$ A X B B B B C C $\Rightarrow$ A A B B B B C C
$\stackrel{*}{\Rightarrow}$ a a b b b b c c.

We see that together with an $A$ an $X$ is generated. This $X$ is sent through the sequence of $B$'s in the middle. When the $X$ meets the C's on the right-hand side it is changed into a $C$. While the $X$ travels through the $B$'s, the number of $B$'s is doubled. This is different from an ordinary CSG. In an ordinary CSG it is possible to have a travelling $X$ that does not double the material it passes (with a rule like X B $\rightarrow$ B X). □

**Lemma 4.3** *ACSL* $\subset$ *QGCSL*

Suppose we have an acyclic context-sensitive grammar $G = (\langle V, \Sigma, S, P \rangle)$. We construct a QGCSG $G' = (\langle V, \Sigma, S, P' \rangle)$ with weight function $g$ that generates the same language as the ACSG as follows. $P'$ is obtained by removing all labels from $P$. $G$ and $G'$ generate the same language by definition (the weight function is irrelevant for the generative capacity).

It remains to show that we can construct a function $g$ such that $\forall \alpha \rightarrow \beta \in P' : g(\alpha) < g(\beta)$. First we construct a graph as follows. For every unlabeled symbol in the ACSG there is a vertex in the graph. There is an arc from vertex $T$ to vertex $U$ iff the unary rule $T \rightarrow U$ is in the context-free backbone of the grammar. With $|V|$ we denote the cardinality of a set $V$. The maximal number of vertices in the graph is $|V|$.

We introduce a counter $i$ that is initialized to $|V| + 1$. Assign the weight $i$ to all vertices that are not connected to any other vertex and remove them. Now search the graph for a vertex without incoming edges. The weight of this symbol is $i$. Remove the vertex. We increment $i$ by 1 and search for the next vertex without incoming edges. We repeat this untill the graph is empty. The algorithm is guaranteed to stop with an empty graph because the graph is acyclic. Observe that $\forall x \in V : |V| < g(x) \leq 2|V|$.

We have to prove now that $\forall \alpha \rightarrow \beta \in P' : g(\alpha) < g(\beta)$. We consider two cases:

- $|\alpha| = |\beta|$. In this case the context-free backbone of the ACSG rule is unary. The weight of the context symbols is irrelevant because they occur both in $\alpha$ *and* in $\beta$. The context-free backbone is of the form $A \rightarrow B$. We know that $g(A) < g(B)$ because $A$ was removed earlier from the graph than $B$ and therefore has a lower weight.

- $|\alpha| < |\beta|$. Again, the weight of the context symbols is irrelevant because they are equal both left and right of the arrow. The context-free backbone is of the form $Z \rightarrow \gamma$, with $|\gamma| > 1$. We know that $g(Z) \leq 2|V|$ and that $g(\gamma) > 2|V|$, hence $g(Z) < g(\gamma)$. □

**Lemma 4.4** *QGCSL* $\subset$ *GCSL*

The proof is in Buntrock and Loryś (1992) and is repeated here. For any QGCSG $G$ we construct a GCSG $G'$ and a homomorphism $h$ such that $L(G') = h(L(G))$. Then we introduce a GCSG $G''$ with $L(G'') = L(G)$. We cite Buntrock and Loryś (1992):

> Let $L$ be generated by a quasi-growing grammar $G = \langle V, \Sigma, S, P\rangle$ with weight function $f$. Without loss of generality we can assume that $S$ does not appear on the right hand side of any production and that $f(S) = 1$. Let $c \notin V$ and let $h$ be a homomorphism such that $h(a) = ac^{f(a)-1}$ for each $a \in V$. Then $G' = \langle V \cup \{c\}, \Sigma, S, P'\rangle$, where $P' = \{h(\alpha) \to h(\beta) : (\alpha \to \beta) \in P\}$ is a growing context-sensitive grammar and $L(G') = h(L(G))$.

GCSL's are closed under inverse homomorphism (Buntrock and Loryś 1992). Therefore there exists a GCSG $G''$ that recognizes $L(G)$. □

**Theorem 4.5** *ACSL* $\subset$ *GCSL*

Follows immediately from Lemma's 4.3 and 4.4.

It is not known whether GCSL $\subset$ ACSL.

Buntrock and Loryś (1992) show that GCSL $\neq$ CSL. We get the following dependencies:

CFL $\subset$ ACSL $\subseteq$ GCSL $\subset$ CSL

## 4.2 Complexity of Acyclic CSG's

We formally introduce the following problems:

**UNIFORM RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR**
INSTANCE: An acyclic context-sensitive grammar $G = (V, \Sigma, S, P)$ and a string $w \in \Sigma^*$.
QUESTION: Is $w$ in the language generated by $G$ ?

**UNIFORM RECOGNITION FOR GROWING CONTEXT-SENSITIVE GRAMMAR**
INSTANCE: A growing context-sensitive grammar $G = (V, \Sigma, S, P)$ and a string $w \in \Sigma^*$.
QUESTION: Is $w$ in the language generated by $G$ ?

**RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR G**
INSTANCE: A string $w \in \Sigma^*$.
QUESTION: Is $w$ in the language generated by $G$ ?

**RECOGNITION FOR GROWING CONTEXT-SENSITIVE GRAMMAR G**
INSTANCE: A string $w \in \Sigma^*$.
QUESTION: Is $w$ in the language generated by $G$ ?

We can prove two theorems:

**Theorem 4.6** *There is a polynomial time reduction from UNIFORM RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR to UNIFORM RECOGNITION FOR GROWING CONTEXT-SENSITIVE GRAMMAR*

**Theorem 4.7** *for every G there is a G', such that RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR G is polynomial time reducible to RECOGNITION FOR GROWING CONTEXT-SENSITIVE GRAMMAR G'*

Proof of both theorems. Consider the proofs of Lemma's 4.3 and 4.4. These proofs show how we can find for every ACSG $G$ a GCSG $G'$ and a homomorphism $h$ such that $L(G') = h(L(G))$. We know that $w \in L(G)$ iff $h(w) \in L(G')$. The reduction is polynomial time. This can be seen as follows. Computing a weight function for an ACSG costs kwadratic time. The weights are linear in $|G|$. Computation of $h$ and $G'$ from the weight function is linear. The total reduction is quadratic. □

**Theorem 4.8** *The problem RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR G is in P for all G.*

We know from (Dahlhaus and Warmuth 1986) that RECOGNITION FOR GROWING CONTEXT-SENSITIVE GRAMMAR $G'$ is in P for all $G'$. The theorem follows from this fact and from Theorem 4.7. Aarts (1991) conjectured that the fixed grammar recognition problem was in P. An algorithm was given there without an estimate of the time complexity.

**Theorem 4.9** *The problem UNIFORM RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR is NP-complete.*

This is proved in Aarts (1992) (but ACSG is defined differently here) and in Aarts (1995).

**Theorem 4.10** *The problem UNIFORM RECOGNITION FOR GROWING CONTEXT-SENSITIVE GRAMMAR is NP-complete.*

From Theorems 4.6 and 4.9 it follows that the uniform recognition problem for GCSG is NP-hard. The problem is also in NP because we can guess derivations in GCSG and derivations in GCSG are very short (their length is smaller than the size of the input). □

This result is not new, however. The problem was put forward in an article from Dahlhaus and Warmuth (1986) and has been solved three times (Buntrock and Loryś 1992).

# 5 Discussion

In the introduction we said that we wanted to introduce a formalism that allows parse trees with crossing branches, which has a simple definition, and which was computationally tractable. The last goal has been achieved only partially. The uniform recognition problem is NP-complete. If the grammar is fixed, then the recognition problem is in P. An aside here: the complexity is $\mathcal{O}(n^k)$ where $k$ depends on the grammar. There is no fixed $k$.

An interesting question is which of the two problems is more relevant: the uniform recognition problem or the fixed grammar recognition problem. Barton Jr., Berwick and Ristad (1987) argue that the fixed grammar problem is not interesting because it is about languages, and not about grammars. The definitions they use are the following. The universal recognition problem is:

> Given a grammar $G$ (in some grammatical framework) and a string $x$, is $x$ in the language generated by $G$?

This problem is contrasted with the fixed language recognition problem:

> Given a string $x$, is $x$ in some independently specified set of strings $L$?

If we use the notation of Garey and Johnson (1979), we see that there are in fact not two but three different ways to specify recognition problems. Suppose we have a definition of context-sensitive grammars and the languages they generate. Then we can define the universal or uniform recognition problem as follows.

**UNIFORM RECOGNITION FOR CONTEXT-SENSITIVE GRAMMARS**
INSTANCE: A context-sensitive grammar $G = (V, \Sigma, S, P)$ and a string $w \in \Sigma^*$.
QUESTION: Is $w$ in the language generated by $G$ ?

There are two forms of the fixed language problem. Suppose we have a grammar ABCGRAM that generates the language $\{a^n b^n c^n | n \geq 1\}$. The following two problems can be defined.

**RECOGNITION FOR CONTEXT-SENSITIVE GRAMMAR ABCGRAM**
INSTANCE: A string $w \in \Sigma^*$.
QUESTION: Is $w$ in the language generated by $G$ ?

**MEMBERSHIP IN** $\{a^n b^n c^n | n \geq 1\}$
INSTANCE: A string $w \in \Sigma^*$.
QUESTION: Is $w$ in $\{a^n b^n c^n | n \geq 1\}$ ?

The complexity of RECOGNITION FOR CONTEXT-SENSITIVE GRAMMAR ABCGRAM is identical to the complexity of MEMBERSHIP IN $\{a^n b^n c^n | n \geq 1\}$ because any algorithm that solves the first problem solves the second too and vice versa. The difference lies in the way the problems are specified. Barton Jr. et al. (1987) only consider the type of problems where languages are specified in some other way than by giving a grammar that generates the language. They argue that this type of problems is not interesting because the grammar has disappeared from the problem, and that it is more interesting to talk about families of languages/grammars than about just one language. However, we see that the grammar does not necessarily disappear in the fixed language recognition problem. If the grammar is part of the specification, we can talk about the fixed language problem in the context of families of grammars. We do this by abstracting over the fixed grammar. E.g., we can try to prove that *for every G*, RECOGNITION FOR CONTEXT-SENSITIVE GRAMMAR G is PSPACE-complete. Abstracting from the grammar is different from putting the grammar as input on the tape of a Turing machine, as is done in the uniform recognition problem.

If we want to answer the question which of the two problems is more relevant for us we first have to answer the question: "Relevant for what?". Complexity analysis of grammatical formalisms serves at least two purposes. First, it helps us to design algorithms that can deal with natural language efficiently. Secondly, it can judge grammar formalisms on their psychological relevance.

It is hard to say whether the uniform problem or the fixed grammar problem is more relevant for efficient sentence processing. An argument in favor of the uniform recognition problem is the following. Usually, grammars are very big, bigger than the sentence length. They have a strong influence on the runtime of an algorithm in practice. Therefore we can not simply forget the grammar size as is done in the fixed language recognition problem. On the other hand, we can argue in favor of the fixed language problem as follows. In many practical applications the task an algorithm has to fullfill is the processing of sentences of some given language (e.g. a spelling checker for English). Practical algorithms have to decide over and over whether strings are in the language generated by some fixed grammar. Only in the development phase of the application the grammar changes. This argues for the relevance of the fixed grammar problem. It is not clear which of the two problems is more relevant in the design of efficient algorithms.

The psychological relevance is a very hard subject (shortly discussed in Barton Jr. et al. (1987, pp. 29,74)). Humans can process natural language utterances fast (linear in the length of the sentence). It seems that we have to compare this with the complexity of the fixed language problem. We follow the same line of reasoning more or less that was used in the previous paragraph. The problem that is solved by humans is the problem of understanding sentences of one particular language (or two, or three). People can not change their "built-in grammar" at will. Barton Jr. et al. (1987)) remark that natural languages must be acquired, and that in language acquisition the grammar changes over time. This change is a very slow change, however, compared to the time needed for sentence processing.

Our conclusion is that both the uniform and the fixed language problem are interesting. Which of the two is more important depends on what perspective one takes.

## References


Aarts, Erik: 1991, Recognition for Acyclic Context-Sensitive Grammars is probably Polynomial for Fixed Grammars, *ITK Research Memo no. 8*, Tilburg University.

Aarts, Erik: 1992, Uniform Recognition for Acyclic Context-Sensitive Grammars is NP-complete, *Proceedings of COLING '92*, Nantes, pp. 1157–1161.

Aarts, Erik: 1995, *Computational Complexity of Grammar Formalisms (?)*, PhD thesis, Research Institute for Language and Speech, University of Utrecht. to appear.

Abramson, H. and Dahl, V.: 1989, *Logic grammars*, Springer, New York–Heidelberg–Berlin.

Barton Jr., G. Edward, Berwick, Robert C. and Ristad, Eric Sven: 1987, *Computational Complexity and Natural Language*, MIT Press, Cambridge, MA.

Bunt, Harry: 1988, DPSG and its use in sentence generation from meaning representations, *in* M. Zock and G. Sabah (eds), *Advances in Natural Language Generation*, Pinter Publishers, London.

Buntrock, Gerhard: 1993, Growing Context-sensitive Languages and Automata, *Report no. 69*, University of Würzburg, Computer Science.

Buntrock, Gerhard and Loryś, Krzysztof: 1992, On growing context-sensitive languages, *Proceedings of $19^{th}$ International Colloquium on Automata, Languages and Programming*, Vol. 623 of *Lecture Notes in Computer Science*, Springer, pp. 77–88.

Dahlhaus, Elias and Warmuth, Manfred K.: 1986, Membership for Growing Context-Sensitive Grammars Is Polynomial, *Journal of Computer and System Sciences* **33**, 456–472.

Garey, Michael R. and Johnson, David S.: 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, CA.

Johnson, Mark: 1985, Parsing with discontinuous constituents, *Proceedings of the $23^{rd}$ Ann. Meeting of the ACL*, pp. 127–132.

Johnson, Mark: 1988, *Attribute-Value Logic and the Theory of Grammar*, Vol. 16 of *CSLI Lecture Notes*, CSLI, Stanford.

Kaplan, R. and Bresnan, J.: 1982, Lexical-Functional Grammar: a Formal System for Grammatical Representation, *in* J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, MA, pp. 173–281.

Pereira, Fernando: 1981, Extraposition Grammars, *Computational Linguistics* **7**(4), 243–256.

Salomaa, Arto: 1973, *Formal languages*, Academic Press.

# PARSING IN DIALOGUE SYSTEMS USING TYPED FEATURE STRUCTURES

Rieks op den Akker, Hugo ter Doest, Mark Moll and Anton Nijholt
Dept. of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede
e-mail: {infrieks,terdoest,moll,anijholt}@cs.utwente.nl

## 1 Abstract

The analysis of natural language in the context of keyboard-driven dialogue systems is the central issue addressed in this paper. A module that corrects typing errors, performs domain-specific morphological analysis is developed. A parser for typed unification grammars is designed and implemented in C++; for description of the lexicon and the grammer a specialised specification language is developed. It is argued that typed unification grammars and especially the newly developed specification language are convenient formalisms for describing natural language use in dialogue systems. Research on these issues is carried out in the context of the SCHISMA project, a research project in linguistic engineering; participants in SCHISMA are KPN Research and the University of Twente.

## 2 The Preprocessor MAF

As we postponed the development of a spoken interface to the SCHISMA system, we concentrate here on the analysis of keyboard input. Thus the input of the MAF module is the character string typed in by the client. The MAF module is best seen as the preprocessor of the SCHISMA system. It handles typing errors and detects certain types of phrases (proper names that occur in the database, date and time phrases, number names, etc.). The latter task of MAF is especially important, since it extracts information crucial for the continuation of the dialogue from the input string.

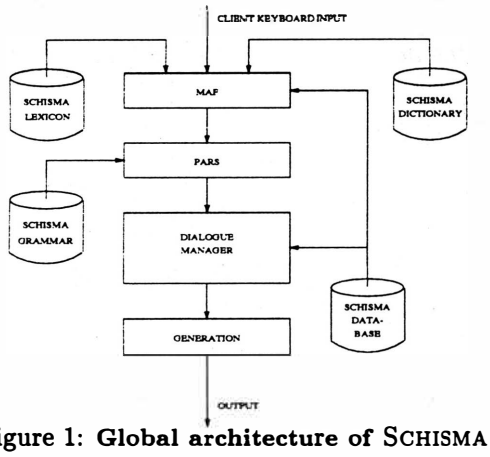



Figure 1: **Global architecture of SCHISMA**

Output of the MAF module is a *word graph*. We define a word graph here as a directed graph having as its nodes the positions in the input string identified as (possible) word boundaries. Nodes are numbered starting with 0 for the leftmost boundary; that is the position left to the first input character. A pair $(index_1, index_2)$ is an edge of the graph if $index_1$ and $index_2$ are word boundaries, $index_1 < index_2$ and the words enclosed between $index_1$ and $index_2$ are identified as one text unit; that is one or more words are identified by MAF as a lexical item to be provided to the parser as a whole. In addition, the MAF module labels the edges of the graph with a value $m$ that indicates the quality of the recognition (and maybe correction) performed.

On the implementation level this means that the MAF module has as output a collection of items $(rd, m)$ where $rd$ is a 3-tuple $(fstruct, index_1, index_2)$, $fstruct$ a typed feature structure, $index_1$ and $index_2$ indices on the word level as explained above, and $m$ is a value indicating the plausibility of $rd$ as a representation of (part of) the input string.

The architecture of the MAF module is quite simple: an error correcting module accepts the input string, processes it, sends its output to some tagging modules and these send their output to the module for morphological analysis and lexicon lookup.

The error correcting module ERROR outputs a word graph that is provided to the tagging modules PROPER, NUMBER, DATE and

TIME that scan the graph for phrases that have special meaning in the SCHISMA domain. In addition, the word graph is provided to the MORPH/LEX module. For performing the error correction ERROR has access to a large dictionary (typically 200,000 words). The tagging modules look for phrases in the input string that carry particularly important information for the dialogue; especially the detection of proper names referring to database items, phrases indicating date and time information and number names is aimed at here; for detecting proper names referring to the database the PROPER module needs access to the SCHISMA database. The output of the taggers then is provided to the MORPH/LEX module; MORPH/LEX creates items for the parser out of the tag information provided by the taggers and it searches the word graph for words that appear in the domain-specific lexicon and for which domain-dependent semantic information is recorded in it.

For details on the tagging modules and the phrases they recognise we refer to (Op den Akker et al. 1995). Also the ERROR and MORPH/LEX module are treated in depth there.

# 3 The Specification Language

To specify a language it is necessary to have a metalanguage. Almost always the usage of a specification language is limited to only one grammar formalism. This is not necessarily a drawback, as such a specification language can be better tailored towards the peculiarities of the formalism. For example, Carpenter's ALE is a very powerful (type) specification language for the domain of unification-based grammar formalisms. But apart from expressiveness of the specification language, the ease with which the intended information about a language can be encoded is also important. An example of a language that combines expressiveness with ease of use is Alshawi's Core Language Engine. Unfortunately the Core Language Engine (CLE) does not support typing. Within our project a type specification language has been developed that can be positioned somewhere between ALE and CLE. This specification language (called *TFS*) can be used to specify a type lattice, a lexicon and a unification grammar for a head-corner parser. The notation is loosely based on CLE, though far less extensive. For instance, the usage of lambda calculus is not supported.

The following example shows how a type lattice can be specified.

```
TYPE(performance;entity;<constr>;<QLF>)
TYPE(play;performance;<constr>;<QLF>)
TYPE(concert;performance;<constr>;<QLF>)
TYPE(musical;play,concert;<constr>;<QLF>)
TYPE(ballet;concert;<constr>;<QLF>)
```

A type specification consists of four parts: a type id for the type to be specified, a list of supertypes, a list of constraints and a formula expressing the semantics for the new type. For each type `<constr>` should be replaced with PATR-II-like path equations and `<QLF>` should be replaced with the semantics in a quasi-logical form. The idea is that the constraints are only necessary *during* parsing and the semantics are passed on to be used *after* parsing.

The next example shows how typing can make some grammar rules superfluous.

```
TYPE(perfphrase; nounphrase; ; )
RULE(nounphrase --> *perfphrase*;
  <nounphrase kind> = <perfphrase kind>,
  <nounphrase sem> = <perfphrase sem>)
```

The asterisks mark the head in the grammar rule. Both the type and rule specify that a performance phrase is a kind of noun phrase.

By path equations QLF expressions can be passed on to other constituents. In the following example a possible quasi-logical form for a phrase is given:

*the opera performance on the 4th of January*

```
EXISTS X (opera(X) AND date(X,4-1-95))
```

The `opera` predicate comes from the QLF part of the opera type and the `date` predicate is generated by parsing the time phrase. Another grammar rule combines these predicates and binds the variable `X`.

It also possible to bind unbound variables to a certain value. This can be done in the specification of a subtype, a word as well as a grammar rule.

# References

Op den Akker, R., Ter Doest, H., Moll, M., and Nijholt, A. (1995). Parsing in dialogue systems using typed feature structures. Memoranda Informatica 95-25, Department of Computer Science, University of Twente.

# Parallel Parsing:
# Different Distribution Schemata for Charts

Jan W. Amtrup, Univ. of Hamburg
e-mail: amtrup@informatik.uni-hamburg.de

### Introduction

We are going to present results from two experiments designed for parallel parsing within the chart paradigm. Parallel processing gains more relevance as applications become increasingly complex and nets of workstations as well as dedicated parallel computers are widely available. A chart-based parser is well suited for approaches to parallelism due to the identification of almost independent data objects that a chart is made of. A parallelization based on tasks of an agenda is only suitable for shared memory systems with a tight coupling, the choice of individual edges as autonomous agents may result in too many processes. But both nodes of a chart and rules of a grammar may provide sufficient possibilities for parallelization in a loosely coupled framework.

### Context-free parsing in a net of workstations

These latter two aspects — node-based and grammar-based approaches — of a data-driven parallelization of chart parsing have been pursued in two experiment sets (for other experiments, confirm e.g. Thompson (1994)). The first system was designed to parse written input on a workstation cluster. It operates with a context-free grammar and uses chart nodes as basis for the distribution of work. A set of chart nodes is assigned to each processor (workstation). It is stored combined with all edges emerging from the given vertices. The configuration and initial synchronization is carried out by an additional process that at the same time functions as a user interface. We use a complete graph as the connection scheme between the processing components. Communication is effected by extensions of the interpretative Scheme language and is based on UNIX sockets.

The chosen corpus consists of simple sentences from the blocks world (like "The green pyramid lies on the red case"), sentence length varying from 2 to 14 words producing 35 to 172 edges for a parse. We experimented with different configurations of up to three machines. Analysis times for the sample varied from 22 to 52 seconds, a speedup of 2.36 could be observed by using three processors compared to one.

### Unification-based parsing with transputers

The results of the first experiment set have to be critically reviewed for different reasons: First, due to the low execution speed of the Scheme dialect used and the resulting long processing time, the speedup values seem to be unrealistically high (extremely slow programs win a lot by

duplication of resources). Second, the results may not generally be valid because of the limited usefulness of a context-free formalism for adequate NLP applications.

Thus, we designed a second system (Amtrup, 1992) that uses an efficient, compiling programming language (C) and a unification-based formalism for grammar and lexicon. A transputer system became the hardware basis for the system which was configured to a ring topology. The program consists of computational processes, a dedicated user interface process that connects the transputer subsystem to the outer world, and routing facilities on each processor to support data flow within the communication network.

We examined two different styles of parallelization. Additionaly to the node based distribution schema described above, we introduced the possibility to assign subsets of the grammar rules to different processors. The system is able to parse sentences from a medical domain (German thorax radiology reports) which mostly consist of complex noun phrases (e.g. "Strahlentransparenz der linken Lunge ohne Hinweise auf frische Infiltrate"). The formalism used to describe grammar and lexicon is a slight modification of PATR II, implemented in C and using a linear unification algorithm. Feature structures are stored as compact arrays of feature node definitions and can be transferred from one process to another without the necessity of linearization and reconstruction.

The grammar used for the tests described here contains 30 context-free rules with some 70 constraint equations. The lexicon consists of 248 entries. The test sentences varied in length between 4 and 18 words, resulting in 152 up to 1382 edges when parsed on a monoprocessor. Again, we studied different configurations using up to seven transputers. The time span necessary for analyzing the sentences varied from 2 to 22 seconds using a large configuration. The speedup values ranged between 1.34 and 3.08 when using seven transputers instead of one. These ratios are naturally lower than that of the context-free parser, but nevertheless promising. It turned out that a parallelization based on a hand-partitioned grammar was more stable than a node-based parallelization which to a certain degree depends on the actual input sentence.

## Conclusion

We have shown that parallelization may increase the performance of chart parsers. While one experiment was conducted to get insights into the mechanisms in principle, the other one aimed at results that could be of practical relevance, too. The speedup ratio of 3.08 with sevenfold resources seems promising. Nevertheless, further investigation is necessary for evaluating the application of the described techniques to speech input. Speedup values grew with the sentence length in the second example, which would suggest that a parallel chart-parsing model can render a better performance when processing speech-data magnitudes larger than written input.

## References

Amtrup, Jan W. 1992. Parallele Strukturanalyse Natürlicher Sprache mit Transputern. ASL-TR 44-92/UHH, Univ. of Hamburg.

Thompson, Henry S. 1994. Parallel Parsers for Context-Free Grammars — Two Actual Implementations Compared. In Geert Adriaens and Udo Hahn, editors, *Parallel Natural Language Processing*. Ablex Publishing Corp., Norwood, NJ, chapter 3, pages 168–187.

# A FUZZY APPROACH TO ERRONEOUS INPUTS IN CONTEXT-FREE LANGUAGE RECOGNITION

Peter R.J. Asveld
Department of Computer Science, Twente University of Technology
P.O. Box 217, 7500 AE Enschede, The Netherlands
e-mail: infprja@cs.utwente.nl

**Abstract** – Using fuzzy context-free grammars one can easily describe a finite number of ways to derive incorrect strings together with their degree of correctness. However, in general there is an infinite number of ways to perform a certain task wrongly. In this paper we introduce a generalization of fuzzy context-free grammars, the so-called fuzzy context-free $K$-grammars, to model the situation of making a finite choice out of an infinity of possible grammatical errors during each context-free derivation step. Under minor assumptions on the parameter $K$ this model happens to be a very general framework to describe correctly as well as erroneously derived sentences by a single generating mechanism.

Our first result characterizes the generating capacity of these fuzzy context-free $K$-grammars. As consequences we obtain: (i) bounds on modeling grammatical errors within the framework of fuzzy context-free grammars, and (ii) the fact that the family of languages generated by fuzzy context-free $K$-grammars shares closure properties very similar to those of the family of ordinary context-free languages.

The second part of the paper is devoted to a few algorithms to recognize fuzzy context-free languages: viz. a variant of a functional version of Cocke–Younger–Kasami's algorithm and some recursive descent algorithms. These algorithms turn out to be robust in some very elementary sense and they can easily be extended to corresponding parsing algorithms.

## 1. Introduction

When we say that a parser is robust it is not quite clear what we mean, since the notion of robustness reflects in fact an informal collection of aspects related to the improper use or the exceptional behavior of the parser. One aspect that is mentioned frequently in this context, concerns the adequate behavior of the parser to small errors in its input. To this aspect and, particularly, the formal distinction between small and big errors, their arising in the derivational process due to a context-free grammar as well as their treatment in the corresponding recognition process, the present paper is devoted.

The first problem that we encounter, is the distinction between small errors ("tiny mistakes") and big errors ("capital blunders") in the input of a parser or recognizer for a context-free language. In traditional formal language theory there is no possibility for such a subtle distinction. Indeed, given a language $L_0$ over an alphabet $\Sigma$ and a string $x$ over $\Sigma$, then $x$ is either *in* or *out* the language $L_0$. This dichotomy of the set $\Sigma^*$ of strings over $\Sigma$ is also apparent when we look at the membership or characteristic function $\mu : \Sigma^* \to \{0, 1\}$ of the set $L_0$ which is defined by $\mu(x; L_0) = 1$ if and only if $x \in L_0$ and $\mu(x; L_0) = 0$ if and only if $x \notin L_0$. But now the notion of fuzzy set may solve this problem, since a fuzzy language over $\Sigma^*$ is defined in terms of a membership function $\mu : \Sigma^* \to [0, 1]$. Note that the two-element set $\{0, 1\}$ has been replaced by the continuous interval $[0, 1]$ and $\mu(x; L_0)$ expresses the degree of membership of the element $x$ with respect to the language $L_0$. Thus $x$ may fully belong to $L_0$ (when $\mu(x; L_0) = 1$), completely be out of $L_0$ (when $\mu(x; L_0) = 0$), or anything in between. In case we choose two appropriate constants $\delta$ and $\Delta$ with $0 < \delta, \Delta < \frac{1}{2}$ we are able to distinguish "tiny mistakes" (those strings $x$ over $\Sigma$ satisfying $1 - \delta \leq \mu(x; L_0) < 1$) from "capital blunders" (strings with $0 < \mu(x; L_0) \leq \Delta$).

The next matter we discuss is: how do errors show up, and which errors (small or big) do we consider. Henceforth, we assume that our language $L_0$ is generated by a context-free grammar $G = (V, \Sigma, P, S)$ consisting of an alphabet $V$, a terminal alphabet $\Sigma$ ($\Sigma \subseteq V$), an initial symbol $S$ ($S \in V$), and a finite set $P$ of rules ($P \subseteq N \times V^*$ where $N = V - \Sigma$). However, it turns out to be more convenient to view $P$ as a function from $V$ to finite subsets of $V^*$ (i.e., finite languages over $V$) rather than as a subset of $N \times V^*$. To be more specific, let $A$ be an arbitrary nonterminal with rules $A \rightarrow \omega_1 \mid \omega_2 \mid \cdots \mid \omega_k$, then we define the function $P$ for argument $A$ as $P(A) = \{A, \omega_1, \omega_2, \cdots, \omega_k\}$, while for each terminal $a$ in $\Sigma$ we have $P(a) = \{a\}$. Note that for each symbol $\alpha$ in $V$, the value of $P(\alpha)$ is a finite language over the alphabet $V$ that contains $\alpha$. The containment of $\alpha$ in this value allows us to interpret $P$ as a nested finite substitution; a concept introduced in [10] and to be recalled in §2.

Let us return to errors and their description. Wrongly applying a rule $A \rightarrow \omega$, will mean in this paper that an occurrence of $A$ is replaced by an incorrect string $\omega'$ instead of the correct string $\omega$. This can be modeled by changing the set $P(A)$ into a fuzzy subset of $V^*$, and adding a finite number of strings $\omega'$ to $P(A)$ with $\mu(\omega'; P(A)) < 1$ for each $\omega'$. This process results in the notion of fuzzy context-free grammar $G = (V, \Sigma, P, S)$ where for each $A$ in $N$, the set $P(A)$ is now a finite fuzzy subset of $V^*$ rather than an ordinary, or so-called crisp subset. Fuzzy context-free grammars have been introduced in a slightly different, but equivalent way in [14]. So using fuzzy context-free grammars, now we are able to model the situation in which the use of a single correct rule can be replaced by the application of any out of a finite number of incorrect rules.

However, in general there is an infinite number of ways to perform a certain task in an erroneous way and performing a grammatical derivation step is no exception to this rule. But simply replacing the finite fuzzy sets $P(\alpha)$ (for each $\alpha$ in $V$) by infinite ones will not work, since in that case the language $L(G)$ generated by the resulting grammar $G$ might not even be recursively enumerable [9]. Thus we have to restrain the languages $P(A)$ in some, preferably natural way. The method we use here, originates from [16]; viz. we assume that a family $K$ of fuzzy languages is given in advance, from which we are allowed to take whatever languages we think to be appropriate. Then replacing the finite languages $P(A)$ over $V$ by members from the family $K$, yields the concept of fuzzy context-free $K$-grammar. The family $K$ plays the role of parameter in our discussion, and when we take $K$ equal to the constant value $\mathrm{FIN}_f$, the family of finite fuzzy languages, we reobtain the ordinary fuzzy context-free grammars.

Remember that fuzzy sets, fuzzy logic and fuzzy grammars have been applied frequently in linguistics and natural language processing. From the many references we only mention the papers in [18] by the inventor of "fuzziness". The present paper is more a sequel to [14] than to any of the more linguistically oriented papers in [18].

The remaining part of this paper is organized as follows. §2 contains some elementary definitions related to fuzzy languages and in §3 we define fuzzy context-free $K$-grammars and the fuzzy languages they generate. Properties of these grammars and languages are discussed in §4. Then §5 is devoted to recognizing algorithms for fuzzy context-free languages: we give appropriate modifications of Cock-Younger-Kasami's algorithm and of some recursive descent algorithms. Finally, §6 contains a comparison with an alternative way of describing grammatical errors using fuzzy grammars too [15, 13], and a straightforward generalization of our results from previous sections.

In the next sections emphasis is on the main ideas and on concrete examples; for detailed formal proofs we refer to [6, 7, 8].

## 2. Preliminaries on Fuzzy Languages

We assume the reader to be familiar with the rudiments of formal language and parsing theory. So for the definitions of context-free grammar, Chomsky Normal Form, and Greibach Normal Form we refer to standard texts like [1, 11, 12].

As mentioned in §1 a *fuzzy language* $L$ over an alphabet $\Sigma$ is a fuzzy subset of $\Sigma^*$, i.e. $L$ is defined by a degree of membership function $\mu_L : \Sigma^* \to [0,1]$. Actually the function $\mu_L$ is the primary notion and $L$ a derived concept, since $L = \{x \in \Sigma^* \mid \mu_L(x) > 0\}$. Henceforth, we write $\mu(x;L)$ rather than $\mu_L(x)$. The *crisp* part of a fuzzy language $L$ is the set $c(L) = \{x \in \Sigma^* \mid \mu(x;L) = 1\}$. A *crisp* (or ordinary) language $L$ is a fuzzy language that satisfies $c(L) = L$.

Next we need a few operations on fuzzy languages: viz. union, intersection, concatenation, and applying a fuzzy function on a fuzzy language. For union and intersection of fuzzy languages $L_i$ ($L_i \subseteq \Sigma_i^*$, $i = 1,2$) we have

$$\begin{aligned} \mu(x;L_1 \cup L_2) &= \max\{\mu(x;L_1), \mu(x;L_2)\}, \text{ and} \\ \mu(x;L_1 \cap L_2) &= \min\{\mu(x;L_1), \mu(x;L_2)\}, \end{aligned} \tag{1}$$

for all $x$ in $(\Sigma_1 \cup \Sigma_2)^*$, respectively. The *concatenation* [14] of fuzzy languages $L_1$ and $L_2$, denoted by $L_1 L_2$, is the fuzzy language satisfying: for all $x$ in $(\Sigma_1 \cup \Sigma_2)^*$,

$$\mu(x;L_1L_2) = \max\{\min\{\mu(y;L_1), \mu(z;L_2)\} \mid x = yz\}. \tag{2}$$

A *fuzzy relation* $R$ between (ordinary) sets $X$ and $Y$ is a fuzzy subset of $X \times Y$. For fuzzy relations $R \subseteq X \times Y$ and $S \subseteq Y \times Z$, their composition $R \circ S$ is defined by

$$\mu((x,z);R \circ S) = \max\{\min\{\mu((x,y);R), \mu((y,z);S)\} \mid y \in Y\}. \tag{3}$$

A *fuzzy function* $f : X \to Y$ is a fuzzy relation $f \subseteq X \times Y$, satisfying for all $x$ in $X$: if $\mu((x,y);f) > 0$ and $\mu((x,z);f) > 0$, then $y = z$ and, consequently, we have $\mu((x,y);f) = \mu((x,z);f)$. Note that (3) applies to fuzzy functions as well. But the composition of two functions $f : X \to Y$ and $g : Y \to Z$ is usually written as $g \circ f : X \to Z$ rather than $f \circ g$.

In the sequel we need a function of type $f : X \to \mathcal{P}(X)$ — where $\mathcal{P}(X)$ denotes the power set of the set $X$ — that will be extended to the function $f : \mathcal{P}(X) \to \mathcal{P}(X)$ by $f(S) = \bigcup\{f(x) \mid x \in S\}$ and for each subset $S$ of $X$,

$$\mu(y;f(S)) = \max\{\min\{\mu(x;S), \mu((x,y);f)\} \mid x \in X\}; \tag{4}$$

cf. e.g. Definition 2.2 below where $X = V^*$ for some alphabet $V$, and $S$ is a language over $V$. Fuzzy functions like $f \circ f$, $f \circ f \circ f$, and so on, are now meaningful by (3) and (4).

Next we turn to the notion of family of fuzzy languages.

**Definition 2.1.** Let $\Sigma_\omega$ be a countably infinite set of symbols. A *family of fuzzy languages* is a set of pairs $(L, \Sigma_L)$ where $L$ is a fuzzy subset of $\Sigma_L^*$ and $\Sigma_L$ is a finite subset of $\Sigma_\omega$. We assume that the alphabet $\Sigma_L$ is minimal with respect to $L$, i.e., a symbol $a$ is a member of $\Sigma_L$ if and only if $a$ occurs in a word $x$ with $\mu(x;L) > 0$. A family of fuzzy languages $K$ is called *nontrivial* if $K$ contains a language $L$ such that $\mu(x;L) > 0$ for some $x \in \Sigma_\omega^+$. A family is called *crisp* if all its members are crisp languages.

Frequently, we write $L$ instead of $(L, \Sigma_L)$ for members of a family of (fuzzy) languages, especially when $\Sigma_L$ is clear from the context. □

Examples of simple, nontrivial families of fuzzy languages, which we will use, are the family $\mathrm{FIN}_f$ of finite fuzzy languages, the family $\mathrm{ALPHA}_f$ of finite fuzzy languages of which the members have unit length (i.e., these languages are alphabets), and the family $\mathrm{SYMBOL}_f$ of singleton languages of unit length. Formally,

$$\begin{aligned} \mathrm{FIN}_f &= \{\{w_1, w_2, \cdots, w_n\} \mid w_i \in \Sigma_\omega^*,\ 1 \le i \le n,\ n \ge 0\}, \\ \mathrm{ALPHA}_f &= \{\Sigma \mid \Sigma \subset \Sigma_\omega, \Sigma \text{ is finite}\}, \text{ and} \\ \mathrm{SYMBOL}_f &= \{\{\alpha\} \mid \alpha \in \Sigma_\omega\}. \end{aligned}$$

The corresponding families of crisp languages are denoted by FIN, ALPHA, and SYMBOL, respectively. Note that the family $\mathrm{FIN}_f$ is closed under the operations union, intersection and concatenation, the family $\mathrm{ALPHA}_f$ is closed under union and intersection but not under concatenation, whereas $\mathrm{SYMBOL}_f$ is not closed under any of these three operations. A similar statement holds for the corresponding crisp families.

Finally, we will consider a more complicated operation on languages; it slightly generalizes a concept from [10].

**Definition 2.2.** Let $K$ be a family of fuzzy languages. A *nested fuzzy $K$-substitution* over an alphabet $V$ is a mapping $P : V \to K$ satisfying:

(i) for each $\alpha$ in $V$, $P(\alpha)$ is a fuzzy $K$-language over $V$, and

(ii) $P$ is nested, i.e., $\mu(\alpha; P(\alpha)) = 1$ for each $\alpha$ in $V$.

The mapping $P$ is extended to words over $V$ by $P(\lambda) = \{\lambda\}$ with $\mu(\lambda; P(\lambda)) = 1$ — where $\lambda$ denotes the empty word — and $P(\alpha_1\alpha_2 \cdots \alpha_n) = P(\alpha_1)P(\alpha_2) \cdots P(\alpha_n)$ with $\alpha_i \in V$ for all $i$ $(1 \le i \le n,\ n \ge 0)$. Finally, $P$ is extended to languages over $V$ by $P(L) = \cup\{P(x) \mid x \in L\}$ and, according to (4), for each language $L$ over $V$,

$$\mu(y; P(L)) = \max\{\min\{\mu(x; L), \mu((x,y); P)\} \mid x \in L\}. \tag{5}$$

A nested fuzzy $K$-substitution $P$ over $V$ can be iterated, giving rise to a *iterated nested fuzzy $K$-substitution* over $V$, i.e., a mapping $P^*$ from languages over $V$ to languages over $V$, defined by $P^*(L) = \cup\{P^n(L) \mid n \ge 0\}$ with $P^{i+1}(L) = P(P^i(L))$ for each $i \ge 0$, and $P^0(L) = L$.

A family $K$ of fuzzy languages is closed under [iterated] nested fuzzy substitution if for each fuzzy language $L$ over some alphabet $V$, and for each [iterated] nested fuzzy $K$-substitution over $V$, we have $P(L) \in K$ [$P^*(L) \in K$, respectively]. □

Note that in Definition 2.2 we used the operations union, concatenation, function application and function composition; cf. (3), (4) and (5) in particular.

**Example 2.3.** The families $\mathrm{SYMBOL}_f$ and $\mathrm{ALPHA}_f$ are closed under (iterated) nested fuzzy substitution. On the other hand, although the family $\mathrm{FIN}_f$ is closed under nested fuzzy substitution, it is not closed under iterated nested fuzzy substitution. Viz. consider $P$ over $V = \{a, b\}$ with $\mu(a; P(a)) = 1$, $\mu(b; P(b)) = 1$, and $\mu(aba; P(b)) = 0.4$, whereas for all other arguments $\mu$ takes the value 0. Now for each fuzzy language $L$ over $V$ that contains at least one word in which a symbol $b$ occurs, the fuzzy language $P^*(L)$ is infinite. Let $L$ be $\{a^n ba^n \mid 0 \le n \le 3\}$ with $\mu(a^n ba^n; L) = 1$ for $n = 0, 1$ and $\mu(a^n ba^n; L) = 0.2$ for $n = 2, 3$. Then $P^*(L) = \{a^n ba^n \mid n \ge 0\}$ where $\mu(a^n ba^n; P^*(L))$ equals 1 for $n = 0, 1$ and 0.4 for all $n \ge 2$. □

## 3. Fuzzy Context-Free $K$-grammars

In this section we first discuss grammatical errors by means of a few examples of (fuzzy) context-free grammars. Then we formally define fuzzy context-free $K$-grammars and the languages they generate.

**Example 3.1.** Consider the (ordinary) context-free grammar $G = (V, \Sigma, P, S)$ with $V = \Sigma \cup \{S\}$, $\Sigma = \{[, ], \langle, \rangle\}$ and $P$ consists of the rules $S \to [S]S \mid \langle S\rangle S \mid \lambda$, where $\lambda$ denotes the empty word. The language $L(G)$ is called the *Dyck language over two types of parentheses* and it consists of all well-matched sequences over $\Sigma$. So $[[]\langle\rangle]$ and $\langle\langle\rangle[]\langle\rangle\rangle$ are in $L(G)$, but $[\langle\rangle[]$ and $[\langle]\rangle$ are not. $L(G)$ plays an important role in the theory of context-free languages, since any context-free language $L_0$ can be obtained from $L(G)$ by the application of an appropriate non-deterministic finite-state transducer $T$, i.e. $L_0 = T(L(G))$, where $T$ depends on $L_0$. As a nested FIN-substitution $P$ looks like $P(S) = \{S, [S]S, \langle S\rangle S, \lambda\}$ and $P(\sigma) = \{\sigma\}$ for each $\sigma$ in $\Sigma$. □

**Example 3.2.** Let $G_0 = (V, \Sigma, P_0, S)$ be the fuzzy context-free grammar that is equal to $G$ of Example 3.1 except that $P_0(S) = P(S) \cup \{[S\rangle S, [SS\}$, $P_0(\sigma) = \{\sigma\}$ for each $\sigma$ in $\Sigma$, $\mu([S\rangle S; P_0(S)) = 0.9$, $\mu([SS; P_0(S)) = 0.1$, and $\mu$ equals 1.0 in all other cases. The string $[S\rangle S$ gives rise to e.g. $\mu([\langle\rangle[\rangle]; L(G_0)) = 0.9$ which is a "tiny mistake" from which it is easy to recover. However, the string $[SS$ causes much more serious problems: we have $\mu([[][]; L(G_0)) = 0.1$, but what is the corresponding correct string? There are three possibilities: $[][][]$, $[[]][]$ and $[[][]]$. So $[[][]$ is considered to be a "capital blunder", when we choose, for instance, $\delta$ and $\Delta$ equal to 0.2 (cf. §1). □

The next step is that we will allow for an infinite number of ways to make grammatical errors, for which we need grammars with an infinite number of rules.

**Example 3.3.** Consider the fuzzy context-free grammar $G_1 = (V, \Sigma, P_1, S)$ that is equal to $G_0$ of Example 3.2 except that $P_1(S) = P(S) \cup \{[^n S\rangle^n S \mid n \geq 1\} \cup \{[SS\}$ with $\mu([^n S\rangle^n S; P_1(S)) = 0.9$ for all $n \geq 1$. It is straightforward to show that $L(G_1) = L(G_0)$, i.e. $\mu(x; L(G_0)) = \mu(x : L(G_1))$ for all $x$ in $\Sigma^*$. □

Crisp grammars with an infinite number of rules have been considered previously; e.g. grammars in extended BNF and the grammatical devices in [16, 2, 3]. In the next definition we generalize the fuzzy context-free grammars from [14].

**Definition 3.4.** Let $K$ be a family of fuzzy languages. A *fuzzy context-free $K$-grammar* $G = (V, \Sigma, P, S)$ consists of

- an alphabet $V$ (the *alphabet* of $G$);
- a subset $\Sigma$ of $V$ (the *terminal alphabet* of $G$);
- a special nonterminal symbol $S$ (the *initial* or *start symbol* of $G$);
- a nested fuzzy $K$-substitution $P$ over $V$, i.e., a mapping $P : V \to K$ satisfying: for each symbol $\alpha$ in $V$, $P(\alpha)$ is a fuzzy language over the alphabet $V$ from the family $K$ with $\mu(\alpha; P(\alpha)) = 1$.

The fuzzy *language* generated by $G$ is the fuzzy set $L(G)$ defined by $L(G) = P^*(S) \cap \Sigma^*$. The family of fuzzy languages generated by fuzzy context-free $K$-grammars is denoted by $A_f(K)$. The corresponding family of crisp languages is denoted by $c(A_f(K))$, i.e., $c(A_f(K)) = \{c(L) \mid L \in A_f(K)\}$. □

Comparing Definition 3.4 with the concept of fuzzy context-free grammar from [14] yields the following differences:

(1) Following 3.4 it is allowed to rewrite terminal symbols.
(2) In 3.4 $L(G)$ is defined in terms of the operations union, intersection, concatenation and (iterated) function application rather than in terms of derivations.
(3) In 3.4 there is an infinite number of rules in $P$.

Now (1) happens to be a minor point (cf. §4), (2) is just a reformulation, but (3) is the main point. With respect to (2), the language $L(G)$ can also be defined using derivations; cf. [14]. A string $x$ over $\Sigma$ belongs to $L(G)$ if and only if there exist strings $\omega_0, \omega_1, \cdots, \omega_n$ over $V$ such that $S = \omega_0 \Rightarrow \omega_1 \Rightarrow \omega_2 \cdots \Rightarrow \omega_n = x$. If $A_i \to \psi_i$ $(0 \leq i < n)$ are the respective rules used in this derivation, then

$$\mu(x; L(G)) = \max\{\min\{\mu(\psi_i; P(A_i)) \mid 0 \leq i < n\} \mid S = \omega_0 \Rightarrow^* \omega_n = x\}, \qquad (6)$$

i.e., the maximum is taken over all possible derivations of $x$ from $S$. If such a derivation is viewed as a chain link of rule applications, its total "strength" equals the strength of its weakest link; hence the min-operation. And $\mu(x; L(G))$ is the strength of the strongest derivation chain from $S$ to $x$; cf. [14].

**Example 3.5.** According to the grammar $G_1$ of Example 3.3 we have the derivation

$$S \Rightarrow [S]S \Rightarrow [[[S\rangle\rangle S]S \Rightarrow [[[\rangle\rangle S]S \Rightarrow [[[\rangle\rangle]S \Rightarrow [[[\rangle\rangle]]$$

and $\mu([[[\rangle\rangle]]; L(G_1)) = 0.9$ since in the second step we used $\mu([^2 S\rangle^2 S; P_1(S)) = 0.9$ while

in all other steps $\mu$ has the value 1.0. □

**Example 3.6.** $A_f(\mathrm{FIN}_f) = \mathrm{CF}_f$, i.e., the family of fuzzy context-free languages from [14]. Hence, $c(A_f(\mathrm{FIN}_f)) = \mathrm{CF}$, the family of (ordinary or crisp) context-free languages. Note that $A_f(\mathrm{SYMBOL}_f) = c(A_f(\mathrm{SYMBOL})) = \{\varnothing\}$, since $S \Rightarrow S$ is the only possible derivation for the corresponding grammars. But $A_f(\mathrm{ALPHA}_f) = \mathrm{ALPHA}_f$, and similarly $c(A_f(\mathrm{ALPHA}_f)) = A_f(\mathrm{ALPHA}) = \mathrm{ALPHA}$ as only sentential forms of length 1 occur in each derivation of the corresponding grammars. □

## 4. Properties of Fuzzy Context-Free $K$-languages

Throughout this section we restrict our attention to those families $K$ satisfying some minor conditions, collected in

**Assumption 4.1.** Henceforth, $K$ is a family of fuzzy languages that satisfies:

(1) $K$ contains all crisp SYMBOL-languages ($K \supseteq \mathrm{SYMBOL}$);

(2) $K$ is closed under union with SYMBOL-languages, i.e. for each $L$ from $K$ and each crisp $\{\alpha\}$ from SYMBOL, the fuzzy language $L \cup \{\alpha\}$ also belongs to $K$;

(3) $K$ is closed under isomorphism ("renaming of symbols"), i.e. for each $L$ over $\Sigma$ from $K$ and for each bijective mapping $i : \Sigma \to \Sigma_1$ — extended to words and to languages in the usual way — we have that the language $i(L)$ belongs to $K$. □

Note that the family $\mathrm{ALPHA}_f$ is the smallest family satisfying these properties.

Our first result deals with the generating power of fuzzy context-free $K$-grammars.

**Theorem 4.2.** Under assumption 4.1, we have $A_f(A_f(K)) = A_f(K)$.

*Proof (sketch):* The inclusion $A_f(K) \subseteq A_f(A_f(K))$ is easy to establish. Viz. for each $L_0$ in $A_f(K)$ with fuzzy context-free $K$-grammar $G = (V, \Sigma, P, S)$ and $L(G) = L_0$, there is a fuzzy context-free $A_f(K)$-grammar $G_0 = (V_0, \Sigma, P_0, S_0)$ with $V_0 = \Sigma \cup \{S_0\}$, $P_0(S_0) = \{S_0\} \cup L(G)$, and $P_0(\sigma) = \{\sigma\}$ for all $\sigma$ in $\Sigma$. Then for each $x$ in $\Sigma^*$, we have $\mu(x; L(G_0)) = \mu(x; L(G)) = \mu(x; L_0)$.

To show the converse inclusion, let $G = (V, \Sigma, P, S)$ be a fuzzy context-free $A_f(K)$-grammar. So $P$ is a nested fuzzy $A_f(K)$-substitution over the alphabet $V$. For each $\alpha$ in $V$, let $G_\alpha = (V_\alpha, V, P_\alpha, S_\alpha)$ be a fuzzy context-free $K$-grammar — i.e. each $P_\alpha$ is a nested fuzzy $K$-substitution over $V_\alpha$ — such that $L(G_\alpha) = P(\alpha)$. We assume that all nonterminal alphabets $V_\alpha - V$ are mutually disjoint. The proof that $L(G) \in A_f(K)$ consists of the following three parts:

(a) Using 4.1.(3) we modify each grammar $G_\alpha$ ($\alpha \in V$) in such a way that $P_\alpha(\beta) = \{\beta\}$ holds for each terminal symbol $\beta$ in $V$.

(b) For each nested fuzzy $K$-substitution $P_\alpha$ over $V_\alpha$, we define a corresponding nested fuzzy $K$-substitution $Q_\alpha$ by

$$\begin{aligned} Q_\alpha(\beta) &= P_\alpha(\beta) && \text{iff} \quad \beta \in V_\alpha - V \\ Q_\alpha(\beta) &= \{\beta, S_\beta\} && \text{iff} \quad \beta \in V \\ Q_\alpha(\beta) &= \{\beta\} && \text{iff} \quad \beta \in V_1 - V_\alpha \end{aligned}$$

with $V_1 = \bigcup\{V_\alpha \mid \alpha \in V\}$. Now we have that $L(G) = \{Q_\alpha \mid \alpha \in V\}^* \cap \Sigma^*$.

(c) Finally, using 4.1.(1)-(3) we reduce the finite set $\{Q_\alpha \mid \alpha \in V\}$ of nested fuzzy $K$-substitutions over $V_1$ to an equivalent, single nested fuzzy $K$-substitution $P_0$ over an extension $V_0$ of $V_1$.

Then for $G_0 = (V_0, \Sigma, P_0, S_S)$, we have $\mu(x; L(G_0)) = \mu(x; L(G))$. Thus, $L(G_0) = L(G)$, and $L(G) \in A_f(K)$, i.e., $A_f(A_f(K)) \subseteq A_f(K)$. □

For a complete constructive proof of 4.2 we refer to [6, 7]. Though from a mathematical point of view 4.2 is quite appealing, we turn to the special case $K = \mathrm{FIN}_f$ in order to resume our discussion on errors and fuzzy context-free grammars.

**Corollary 4.3.** $A_f(A_f(\mathrm{FIN}_f)) = A_f(\mathrm{CF}_f) = A_f(\mathrm{FIN}_f) = \mathrm{CF}_f$. □

Corollary 4.3 provides us the limit of deriving grammatical errors within the framework of fuzzy context-free grammars. Viz. we may extend the sets $P(\alpha)$ (for each $\alpha$ in the alphabet $V$) to infinite sets, as long as the resulting sets $P(\alpha)$ still constitute fuzzy context-free languages over $V$. Only in this way we are able to model the case of an infinite number of possible grammatical errors. Of course, during each derivation only a finite choice out of this infinity of possible errors will be made.

Though the construction in (second part of) the proof of 4.2 is applicable to each fuzzy context-free $A_f(K)$, sometimes an ad hoc construction may result in a simpler grammar.

**Example 4.4.** Consider the fuzzy context-free $A_f(\mathrm{FIN}_f)$-grammar or fuzzy context-free $\mathrm{CF}_f$-grammar $G_1$ of Example 3.3. We will construct an equivalent fuzzy context-free $\mathrm{FIN}_f$-grammar $G_2 = (V_2, \Sigma, P_2, S)$. Let $V_2 = \{A\} \cup V_1$, $P_2(S) = P(S) \cup \{[SS, AS\}$, $P_2(A) = \{A, [A\rangle, [S\rangle\}$, $\mu(AS; P_2(S)) = \mu([A\rangle; P_2(A)) = \mu([S\rangle; P_2(A)) = 0.9$ and everything else is as in Example 3.3. Then $\mu(x; L(G_2)) = \mu(x; L(G_1))$ for all $x$ in $\Sigma^*$, i.e., $L(G_2) = L(G_1)$. □

We conclude this section with some mathematical consequences of 4.2 and 4.3 for which we need the following fuzzy analogue to the notion of full super-AFL [10].

**Definition 4.5.** A nontrivial family $K$ of fuzzy languages is called a *full super-AFFL* (i.e., *full super-Abstract Family of Fuzzy Languages*) if $K$ is closed under

- finite fuzzy substitution (i.e. $\mathrm{FIN}_f$-substitution);
- intersection with fuzzy regular languages;
- iterated nested fuzzy substitution. □

From closure under these three operations, closure under many other operations well known in formal language theory follows: closure under union, concatenation, Kleene $*$, homomorphism, inverse homomorphism, substitution, nondeterministic finite-state transductions, and so on; cf. [10, 2, 4] and also [17].

**Theorem 4.6.** [7]

(1) Let $K$ be a nontrivial family of fuzzy languages closed under finite fuzzy substitution and under intersection with fuzzy regular languages. Then $A_f(K)$ is a full super-AFFL, and, in particular, it is the smallest full super-AFFL that includes the family $K$.

(2) Each full super-AFFL includes the family $\mathrm{CF}_f$ of fuzzy context-free languages.

(3) The family $\mathrm{CF}_f$ is the smallest full super-AFFL. □

The proof of 4.6 heavily relies on Theorem 4.2 and Corollary 4.3; cf. [7]. Comparing 4.6 with results in [10, 2, 4] yields that the family of fuzzy context-free languages possesses closure properties very similar to those of the (ordinary or crisp) context-free languages.

## 5. Recognizing Fuzzy Context-Free Languages

In this section we give a few algorithms for recognizing fuzzy context-free languages. When a fuzzy context-free language has been specified by a fuzzy context-free $\mathrm{CF}_f$-grammar (Corollary 4.3), we must first transform that grammar into an equivalent fuzzy context-free $\mathrm{FIN}_f$-grammar by means of the construction in the proof of Theorem 4.2. Next we must transform the resulting grammar into Chomsky or Greibach Normal Form, using results from [14], before we can apply the algorithms from this section.

The first algorithm is a modification of Cocke–Younger–Kasami's algorithm (or CYK-algorithm for short); cf. Algorithm 5.2 below. In e.g. [1, 11, 12] the CYK-algorithm is given in terms of nested **for**-loops that fill an upper-triangular matrix. Here we start

from an alternative functional version from [5] which has some interesting features: it omits implementation details like the data structure, reference to the indices of matrix entries and to the length of the input string; cf., e.g., Algorithm 12.4.1 in [11] and Algorithm 5.1 below.

**Algorithm 5.1.** Let $G = (V, \Sigma, P, S)$ be a $\lambda$-free context-free grammar in Chomsky Normal Form and let $w$ be in $\Sigma^+$. Define functions $f : \Sigma^+ \to \mathcal{P}(N^+)$ and $g : \mathcal{P}(N^+) \to \mathcal{P}(N)$ by:

- For each $w$ in $\Sigma^+$ the function $f$ is defined as the finite substitution generated by

$$f(a) = \{A \mid a \in P(A)\} \tag{7}$$

and extended to words over $\Sigma$ by

$$f(w) = f(a_1)f(a_2)\cdots f(a_n) \quad \text{if} \quad w = a_1a_2\cdots a_n \quad (a_k \in \Sigma,\ 1 \le k \le n). \tag{8}$$

- For each $A$ in $N$ we define $g(A) = \{A\}$ and for each $\omega$ in $N^+$ with $|\omega| \ge 2$ we have

$$g(\omega) = \bigcup \{g(\chi) \otimes g(\eta) \mid \chi, \eta \in N^+,\ \omega = \chi\eta\} \tag{9}$$

where for each $X$ and $Y$ in $\mathcal{P}(N)$ the binary operation $\otimes$ is defined by

$$X \otimes Y = \{A \mid BC \in P(A), \text{ with } B \in X \text{ and } C \in Y\}. \tag{10}$$

- For each (finite) language $M$ over $N$, $g(M)$ is defined by

$$g(M) = \bigcup \{g(\omega) \mid \omega \in M\}. \tag{11}$$

Finally, compute $g(f(w))$ and determine whether $S$ belongs to $g(f(w))$.

Clearly, we have $w \in L(G)$ if and only if $S \in g(f(w))$. □

From this functional version of the CYK-algorithm it is easy to derive an algorithm for recognizing fuzzy context-free languages.

**Algorithm 5.2.** Let $G = (V, \Sigma, P, S)$ be a $\lambda$-free fuzzy context-free grammar in Chomsky Normal Form and let $w$ be in $\Sigma^+$. Extend (7)–(11) in Algorithm 5.1 with

$$\mu(A; f(a)) = \mu(a; P(A)), \tag{7'}$$

$$\mu(A; X \otimes Y) = \min\{\mu(BC; P(A)), \mu(B; X), \mu(C; Y)\}, \tag{9'}$$

$$\mu(A; g(\omega)) = \max\{\mu(A; g(\chi) \otimes g(\eta)) \mid \chi, \eta \in N^+,\ \omega = \chi\eta\}, \tag{10'}$$

whereas corresponding equalities for (8) and (11) follow from the definitions of concatenation and finite union, respectively; cf. §2. Finally, compute $\mu(S; g(f(w)))$.

Then, we have $\mu(w; L(G)) = \mu(S; g(f(w)))$. □

**Example 5.3.** Consider the fuzzy context-free grammar $G_3 = (V_3, \Sigma, P_3, S)$ with $V_3 = \Sigma \cup \{S, A, B, C, D, E, F\}$, $P_3$ consists of the rules

$$\begin{array}{llll} S \to SS \mid AC \mid BC \mid DF \mid EF \mid AF \mid BF \mid BS \mid [, & & & \\ A \to BS, & B \to [, & C \to ], & \\ D \to ES, & E \to \langle, & F \to \rangle, & \end{array}$$

where $\mu(AF; P_3(S)) = \mu(BF; P_3(S)) = 0.9$, $\mu(BS; P_3(S)) = \mu([; P_3(S)) = 0.1$ and $\mu$ has value 1.0 in all other cases. The grammar $G_3$ is $\lambda$-free, in Chomsky Normal Form, and equivalent (modulo the empty word $\lambda$) to $G_0$ from Example 3.2.

Applying Algorithm 5.2 with, e.g., input equal to $[]\langle\rangle$, yields

$$\mu([]\langle\rangle; L(G_3)) = \mu(S; g(f([]\langle\rangle))) = \mu(S; g(\{B, S\}CEF)) = \mu(S; g(\{B, S\}) \otimes g(CEF) \cup$$
$$\cup g(\{B, S\}C) \otimes g(EF) \cup g(\{B, S\}CE) \otimes g(F)) = \cdots = \mu(S; S) = 1.0$$

where we write, as usual, $X$ for a singleton set $\{X\}$. Similarly, for input equal to $[[]\rangle$, we get

$$\mu([[]\rangle; L(G_3)) = \mu(S; g(f([[]\rangle))) = \mu(S; g(\{B, S\}\{B, S\}CF)) =$$
$$= \mu(S; g(\{B, S\}) \otimes g(\{B, S\}CF) \cup g(\{B, S\}\{B, S\}) \otimes g(CF) \cup$$
$$\cup g(\{B, S\}\{B, S\}C) \otimes g(F)) = \cdots = 0.9$$ □

Algorithms 5.1 and 5.2 are bottom-up algorithms for recognizing λ-free (fuzzy) context-free languages in Chomsky Normal Form. Functional versions of top-down ("recursive descent") algorithms for crisp context-free languages have been introduced in [5], from which we recall Definition 5.4 and Algorithm 5.5. In Algorithm 5.6 we give a modification of 5.5 which results in a recursive descent recognizer for fuzzy context-free languages in Chomsky Normal Form.

**Definition 5.4.** For each context-free grammar $G=(V,\Sigma,P,S)$ with $N=V-\Sigma$, the set $T(\Sigma,N)$ of *terms* over $(\Sigma,N)$ is the smallest set defined by

(a) $\lambda$ is a term in $T(\Sigma,N)$ and each $a$ $(a\in\Sigma)$ is a term in $T(\Sigma,N)$.

(b) For each $A$ in $N$ and each term $t$ in $T(\Sigma,N)$, $A(t)$ is a term in $T(\Sigma,N)$.

(c) If $t_1$ and $t_2$ are in $T(\Sigma,N)$, then their concatenation $t_1t_2$ is a term in $T(\Sigma,N)$ too. □

Note that for any two sets of terms $S_1$ and $S_2$ $(S_1,S_2\subseteq T(\Sigma,N))$ the set $S_1S_2$, defined by $S_1S_2=\{t_1t_2 \mid t_1\in S_1,\ t_2\in S_2\}$, is also a set of terms over $(\Sigma,N)$.

**Algorithm 5.5.** Let $G=(V,\Sigma,P,S)$ be a λ-free context-free grammar in Chomsky Normal Form and let $w$ be a string in $\Sigma^+$. Each nonterminal symbol $A$ in $N$ is considered as a function from $\Sigma^*\cup\{\perp\}$ to $\mathcal{P}(T(\Sigma,N))$ defined as follows. (The symbol $\perp$ will be used to denote "undefined".) First, $A(\perp)=\varnothing$ and $A(\lambda)=\{\lambda\}$ for each $A$ in $N$. If the argument $x$ of $A$ is a word of length 1 (i.e. $x$ is in $\Sigma$) then

$$A(x) = \{\lambda \mid x\in P(A)\} \qquad (x\in\Sigma) \tag{12}$$

and in case the length $|x|$ of the word $x$ is 2 or more, then

$$A(x) = \bigcup\{B(y)C(z) \mid BC\in P(A),\ y,z\in\Sigma^+,\ x=yz\}. \tag{13}$$

Finally, we compute $S(w)$ and determine whether $\lambda$ belongs to $S(w)$.

It is straightforward to show that $w\in L(G)$ if and only if $\lambda\in S(w)$. □

**Algorithm 5.6.** Let $G=(V,\Sigma,P,S)$ be a λ-free fuzzy context-free grammar in Chomsky Normal Form and let $w$ be a string in $\Sigma^+$. For all $A$ in $N$, $\mu(\lambda;A(\lambda))=1$ and $\mu(t;A(\perp))=0$ for each $t$ in $\mathcal{P}(T(\Sigma,N))$. Extend (12)–(13) in Algorithm 5.5 with

$$\mu(\lambda;A(x))=\mu(x;P(A)) \qquad (x\in\Sigma), \tag{12'}$$

$$\mu(\lambda;A(x)) = \max\{\min\{\mu(BC;P(A)),\mu(\lambda;B(y)),\mu(\lambda;C(z))\} \mid BC\in P(A),\ y,z\in\Sigma^+,\ x=yz\}. \tag{13'}$$

Finally, we compute $\mu(\lambda;S(w))$. Then we have $\mu(w;L(G))=\mu(\lambda;S(w))$. □

**Example 5.7.** Applying Algorithm 5.6 to the fuzzy context-free grammar $G_3$ of Example 5.3 results in

$$\begin{aligned}\mu(\langle\rangle[];L(G_3)) &= \mu(\lambda;S(\langle\rangle[])) = \\ &= \mu(\lambda;S(\langle\rangle[)S(])\cup S(\langle\rangle)S([])\cup S(\langle)S(\rangle[])\cup \\ &\quad A(\langle\rangle[)C(])\cup A(\langle\rangle)C([])\cup A(\langle)C(\rangle[])\cup \\ &\quad B(\langle\rangle[)C(])\cup B(\langle\rangle)C([])\cup B(\langle)C(\rangle[])\cup \\ &\quad D(\langle\rangle[)F(])\cup D(\langle\rangle)F([])\cup D(\langle)F(\rangle[]))\cup \\ &\quad E(\langle\rangle[)F(])\cup E(\langle\rangle)F([])\cup E(\langle)F(\rangle[])) = \cdots = 1\end{aligned}$$

$$\mu([\langle\rangle;L(G_3)) = \mu(\lambda;S([\langle\rangle)) = \cdots = 0.1$$

$$\mu(\langle]\rangle;L(G_3)) = \mu(\lambda;S(\langle]\rangle)) = \cdots = 0$$ □

Finally, we give analogues of Algorithms 5.5 and 5.6 based on Greibach 2-form (viz. Algorithms 5.8 and 5.9) which are slightly more efficient then 5.5 and 5.6, respectively. Recall that a λ-free context-free grammar is in *Greibach 2-form* if its rules possess one of the forms: $A\to aBC$, $A\to aB$ and $A\to a$ $(a\in\Sigma,\ A,B,C\in N)$.

**Algorithm 5.8.** Let $G=(V,\Sigma,P,S)$ be a λ-free context-free grammar in Greibach 2-form and let $w$ be a string in $\Sigma^+$. The algorithm is as Algorithm 5.5 except that (13) is

replaced by

$$A(x) = \bigcup \{B(y)C(z) \mid aBC \in P(A),\ y,z \in \Sigma^+,\ x = ayz\} \cup \bigcup \{B(y) \mid aB \in P(A),\ y \in \Sigma^+,\ x = ay\}. \tag{14}$$

Still we have that $w \in L(G)$ if and only if $\lambda \in S(w)$. □

**Algorithm 5.9.** Let $G = (V, \Sigma, P, S)$ be a $\lambda$-free fuzzy context-free grammar in Greibach 2-form and let $w$ be a string in $\Sigma^+$. For all $A$ in $N$, $\mu(\lambda; A(\lambda)) = 1$ and $\mu(t; A(\perp)) = 0$ for each $t$ in $\mathcal{P}(T(\Sigma, N))$. Extend (14) in Algorithm 7.5 with

$$\mu(\lambda; A(x)) = \mu(\lambda; A'(x) \cup A''(x)) \qquad \text{with} \tag{14'}$$

$$\mu(\lambda; A'(x)) = \max\{\min\{\mu(aBC; P(A)), \mu(\lambda; B(y)), \mu(\lambda; C(z))\} \mid aBC \in P(A),\ y,z \in \Sigma^+,\ x = ayz\}, \tag{14'}$$

$$\mu(\lambda; A''(x)) = \max\{\min\{\mu(aB; P(A)), \mu(\lambda; B(y))\} \mid aB \in P(A),\ y \in \Sigma^+,\ x = ay\}. \tag{14'}$$

Finally, compute $\mu(\lambda; S(w))$. Then $\mu(w; L(G)) = \mu(\lambda; S(w))$. □

**Example 5.10.** Let $G_4 = (V_4, \Sigma, P_4, S)$ be the fuzzy context-free grammar with $V = \Sigma \cup \{S, C, F\}$, and $P_4$ consists of rules which are displayed with their degree of membership in the following table.

| set | elements | degree |
|---|---|---|
| $P_4(S)$ | $S, [SCS, \langle SFS, [CS, \langle FS, [SC, \langle SF, [C, \langle F$<br>$[SFS, [FS, [SF, [F$<br>$[SS, [S, [$ | 1.0<br>0.9<br>0.1 |
| $P_4(C)$ | $C, ]$ | 1.0 |
| $P_4(F)$ | $F, \rangle$ | 1.0 |

Applying Algorithm 5.9 yields $\mu(\lambda; C(])) = \mu(\lambda; F(\rangle)) = 1.0$, $\mu(\lambda; S([)) = 0.1$, and

$$\begin{aligned} S(w) = {} & \bigcup \{S(x)C(y)S(z) \mid w = [xyz\} \cup \bigcup \{C(x)S(y) \mid w = [xy\} \cup \\ & \bigcup \{S(x)F(y)S(z) \mid w = \langle xyz\} \cup \bigcup \{F(x)S(y) \mid w = \langle xy\} \cup \\ & \bigcup \{S(x)C(y) \mid w = [xy\} \cup \bigcup \{S(x)F(y) \mid w = \langle xy\} \cup \\ & \bigcup \{S(x)F(y)S(z) \mid w = [xyz\} \cup \bigcup \{F(x)S(y) \mid w = [xy\} \cup \\ & \bigcup \{S(x)F(y) \mid w = [xy\} \cup \bigcup \{S(x)S(y) \mid w = [xy\} \cup \\ & \cup C([ \setminus w) \cup F(\langle \setminus w) \cup F([ \setminus w) \cup S([ \setminus w) \end{aligned}$$

where $x, y$ and $z$ are nonempty strings over $\Sigma$, and $u \setminus v = w$ if $v = uw$, and $\perp$ otherwise ($u, v, w \in \Sigma^*$). Then we have

$$\begin{aligned} \mu(\lambda; S([\langle\rangle])) &= \mu(\lambda; \cdots \cup S(\langle\rangle)C(]) \cup \cdots) = \\ &= \mu(\lambda; \cdots \cup S(\langle\rangle) \cup \cdots) = \mu(\lambda; \cdots \cup F(\rangle) \cup \cdots) = 1.0 \end{aligned}$$

where numerous non-productive terms have been omitted. Similarly,

$$\mu(\lambda; S([[])) = \mu(\lambda; \cdots \cup S([)C(]) \cup S([ \setminus [[]) \cup \cdots) = 0.1$$ □

Of course, the Greibach 2-form is by no means essential; the transformation to this normal form gives rise to numerous additional rules and less transparent algorithms. For instance, a $\lambda$-free version of $G_0$ from Example 3.2 will result in an algorithm that is simpler than 5.10.

## 6. Concluding Remarks

We showed that using fuzzy context-free $K$-grammars we are able to model the case in which at each derivation step a choice from an infinity of possible grammatical errors is made. From Theorem 4.2 and Corollary 4.3 it followed that in order to stay within the

framework of fuzzy context-free language generation this choice should be limited to a fuzzy context-free language. However, to apply the recognition algorithms from §5, these fuzzy context-free $CF_f$-grammars should be transformed into equivalent fuzzy context-free $FIN_f$-grammars. These recognition algorithms, which are straightforward modifications of existing ones, are robust in a very primitive sense; viz. since they compute the membership function, they can distinguish between "tiny mistakes" and "capital blunders".

Our approach in describing grammatical errors has a global character: a right-hand side $\omega$ of a grammar rule may be replaced erroneously by a completely different string $\omega'$. In [15] an alternative way of describing errors — using fuzzy context-free grammars too — is given. Here $\omega'$ is restricted to those strings that are obtainable by simple edit operations (deletion, insertion, and substitution of symbols) from $\omega$, and these operations are performed on nonterminal symbols only. In a companion paper [13] this approach is extended to context-sensitive grammars as well, but both papers are restricted to the discussion of simple examples rather than proving general results.

The definition of fuzzy context-free $K$-grammar can be slightly generalized: viz. instead of a single nested fuzzy $K$-substitutions we may allow a finite number of such substitutions [7]. Under assumption 4.1 it is possible to reduce this finite number to an equivalent single nested fuzzy $K$-substitution; cf. the last part of the proof of 4.2.

Of much more practical interest is another modification / generalization. From a certain point of view the model discussed in this paper is rather trivial: to each grammar rule we associate a real number in between 0 and 1, and these numbers are propagated by means of the min-operation (of course, without any alternation) to a string derived by the fuzzy context-free grammar. So making the very same error twice is as bad as making it a single time. Intuitively, one would prefer that the degree of membership in the first case is strictly lower than the one due to the single error. This can be achieved by deviating from the original definition of fuzzy grammar from [14]. When we replace the min-operation in (2)–(5) and, consequently, in (6), (9′), (13′) and (14′) — but not in (1) — by the multiplication operation, we are able to model this accumulation process of errors.

**Example 6.1.** When we replace the min-operation by multiplication at the appropriate places, Algorithm 5.2 applied to the grammar $G_3$ of Example 5.3 yields, for instance, $\mu([[\rangle[]\rangle;L(G_3)) = \cdots = 0.81$, $\mu([[\rangle[\rangle\rangle;L(G_3)) = \cdots = 0.729$, $\mu([[\rangle[];L(G_3)) = \cdots = 0.08$ and $\mu([[\rangle[;L(G_3)) = \cdots = 0.008$. □

In this way the occurrence of many "tiny mistakes" may result in the end in something that resembles a "capital blunder".

*Acknowledgements.* I am indebted to Rieks op den Akker and Anton Nijholt for their critical remarks.

## References

1. A.V. Aho & J.D. Ullman: *The Theory of Parsing, Translation and Compiling – Volume I: Parsing* (1972), Prentice-Hall, Englewood Cliffs, NJ.
2. P.R.J. Asveld: *Iterated Context-Independent Rewriting – An Algebraic Approach to Families of Languages*, (1978), Ph.D. Thesis, Dept. of Appl. Math., Twente University of Technology, Enschede, The Netherlands.
3. P.R.J. Asveld: Abstract grammars based on transductions, *Theoret. Comput. Sci.* **81** (1991) 269–288.
4. P.R.J. Asveld: An algebraic approach to incomparable families of formal languages, pp. 455-475 in G. Rozenberg & A. Salomaa (eds.): *Lindermayer Systems – Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology* (1992), Springer-Verlag, Berlin, etc.

5. P.R.J. Asveld: An alternative formulation of Cocke–Younger–Kasami's algorithm, *Bull. Europ. Assoc. for Theoret. Comp. Sci.* (1994) No. 53, 213–216.
6. P.R.J. Asveld: Towards robustness in parsing — Fuzzifying context-free language recognition, Memoranda Informatica 95-08, Dept. of Comp. Sci., Twente University of Technology, Enschede, The Netherlands. To appear in *Proc. 2nd Internat. Conf. on Developments in Language Theory* (1995).
7 P.R.J. Asveld: Fuzzy context-free languages — Part I: Generalized fuzzy context-free grammars, Memoranda Informatica 95-??, Dept. of Comp. Sci., Twente University of Technology, Enschede, The Netherlands (In preparation).
8. P.R.J. Asveld: Fuzzy context-free languages — Part II: Recognition Algorithms, Memoranda Informatica 95-??, Dept. of Comp. Sci., Twente University of Technology, Enschede, The Netherlands (In preparation).
9. G. Gerla: Fuzzy grammars and recursively enumerable fuzzy languages, *Inform. Sci.* **60** (1992) 137-143.
10. S.A. Greibach: Full AFL's and nested iterated substitution, *Inform. Contr.* **16** (1970) 7–35.
11. M.A. Harrison: *Introduction to Formal Language Theory* (1978), Addison-Wesley, Reading, Mass.
12. J.E. Hopcroft & J.D. Ullman: *Introduction to Automata Theory, Languages, and Computation* (1979), Addison-Wesley, Reading, Mass.
13. M. Inui, W. Shoaff, L. Fausett & M. Schneider: The recognition of imperfect strings generated by fuzzy context-sensitive grammars, *Fuzzy Sets and Systems* **62** (1994) 21-29.
14. E.T. Lee & L.A. Zadeh: Note on fuzzy languages, *Inform. Sci.* **1** (1969) 421-434.
15. M. Schneider, H. Lim & W. Shoaff: The utilization of fuzzy sets in the recognition of imperfect strings, *Fuzzy Sets and Systems* **49** (1992) 331-337.
16. J. van Leeuwen: A generalization of Parikh's theorem in formal language theory, pp. 17-26 in: J. Loeckx (ed.): *2nd ICALP*, Lect. Notes in Comp. Sci. **14** (1974), Springer-Verlag, Berlin, etc.
17. W. Wechler: *The Concept of Fuzziness in Automata and Language Theory* (1978), Akademie-Verlag, Berlin.
18. R.R. Yager, R.M. Tong, S. Ovchinnikov & H.T. Nguyen (eds.): *Fuzzy Sets and Applications – Selected Papers by L.A. Zadeh* (1987), J. Wiley, New York.

# PARSING NON-IMMEDIATE DOMINANCE RELATIONS*

Tilman Becker
DFKI GmbH
D-66123 Saarbruecken
becker@dfki.uni-sb.de

Owen Rambow
CoGenTex, Inc.
Ithaca, NY 14850
owen@cogentex.com

## Abstract

We present a new technique for parsing grammar formalisms that express non-immediate dominance relations by 'dominance-links'. Dominance links have been introduced in various formalisms such as extensions to CFG and TAG in order to capture long-distance dependencies in free-word order languages (Becker et al., 1991; Rambow, 1994).

We show how the addition of 'link counters' to standard parsing algorithms such as CKY- and Earley-based methods for TAG results in a polynomial time complexity algorithm for parsing lexicalized V-TAG, a multi-component version of TAGs defined in (Rambow, 1994). A variant of this method has previously been applied to context-free grammar based formalisms such as UVG-DL.

## 1 Linguistic Data

Scrambling is the permutation of arguments of a verb in languages such as Finnish, German, Korean, Japanese, Hindi, and Russian. If there are embedded clauses, certain matrix verbs allow scrambling of elements out of the embedded clauses ("long-distance" scrambling). In German, scrambling is quite free.[1] There is no bound on the number of clause boundaries over which an element can scramble. Furthermore, more than one constituent can scramble in the same sentence ("iterability of scrambling"), and an element scrambled (long-distance or not) from one clause does not preclude an element from another clause from being scrambled.

(1) ...daß [dem Kunden]$_i$ [den Kühlschrank]$_j$ bisher noch niemand $t_i$
...that [the client]$_{DAT}$ [the refrigerator]$_{ACC}$ so far as yet no-one$_{NOM}$
[[$t_j$ zu reparieren] zu versuchen] versprochen hat
to repair to try promised has
...that so far, no-one yet has promised the client to repair the refrigerator

We conclude that scrambling in German (and other languages) is "doubly unbounded" in the sense that there is no bound on the distance over which each element can scramble (unboundedness of scrambling), and there is no bound on the number of elements that can scramble in one sentence (iterability of scrambling). This generalization should not be taken to mean that all sentences in which "doubly unbounded" scrambling has occurred will be judged equally acceptable. Clearly, scrambling is constrained by pragmatic and processing factors, and perhaps also by semantic factors. For example, processing load appears to increase with an increasing number of scrambled elements. However, we do not have any reason to define a particular "cut-off point" beyond which all orders are ungrammatical. (This argument is similar to the argument in favor of allowing unlimited recursion in the competence grammar.)

Finally, we observe that scrambling does not preclude long-distance topicalization (a separate linguistic phenomenon also found in English, in which a single element moves into sentence-initial position):

*We would like to thank Aravind Joshi, Ron Kaplan, Martin Kay, John Maxwell, Giorgio Satta, K. Vijay-Shanker, and David Weir for helpful comments and discussions. This work was partially performed while the authors were at the University of Pennsylvania under grants AR● DAAL 03-89-C-0031; DARPA N00014-90-J-1863; NSF IRI 90-16592; and Ben Franklin 91S.3078C-1.

[1] In German, scrambling can never proceed out of tensed clauses. It is widely assumed that embedded infinitival clauses undergo "clause union". If clause union takes place, then in fact there is no long-distance scrambling in German because no clause boundary is crossed. However, throughout this paper we will use the term "long-distance scrambling" in a more descriptive way. We will take the term to mean that an argument of a verb appears to the left of an argument of a less deeply embedded verb.

(2) [Dieses Buch]$_i$ hat [den Kindern]$_j$ bisher noch niemand [PRO $t_j$ $t_i$ zu geben]
[this book]$_{ACC}$ has [the children]$_{DAT}$ so far yet [no-one]$_{NOM}$ to give
versucht.
tried

So far, no-one has tried to give this book to the children.

We conclude that doubly unbounded non-local dependencies occur in natural language, that they co-occur with other unbounded dependency constructions (such as topicalization) and that there is a need to account for such constructions, both formally and computationally.

# 2 V-TAG

Tree Adjoining Grammar (TAG, see (Joshi, 1987) for an introduction) is a tree rewriting system. A TAG consists of a set of *elementary trees* which are combined by the *adjoining* operation, which may insert one tree into the center of another. TAGs are more powerful than context-free grammars but they are only mildly context-sensitive since they generate only semi-linear languages and are polynomially parsable. Their extended domain of locality allows the factoring of recursion and the statement of linguistic dependencies within the elementary structures of the grammar.

However, in (Becker et al., 1991), we argue that scrambling is beyond the formal power of TAG by assuming that elementary trees express a complete predicate-argument structure. (Becker et al., 1991) proposes two variants of the TAG formalism which can derive scrambling while still preserving most of the desirable properties of TAGs (in particular, an extended domain of locality and the factoring of recursion). One of them, FO-TAG, is based on relaxing LP constraints, and we do not discuss it here. The other approach is based on relaxing immediate dominance. This has the effect of creating several "chunks" of the tree which are related by (not necessarily immediate) dominance edges or *dominance links*. Dominance links are essential for encoding structural relations (c-command) between related linguistic elements, such as a head and its arguments.[2] The resulting formalism is called Multi-Component TAG with Dominance Links (MC-TAG-DL).

In defining MC-TAG-DL, several options are available for the definition of the derivation relation. In V-TAG (Rambow, 1994), there are no restrictions on adjunction sites. Trees from one tree set can be adjoined anywhere in the derived tree, and they need not be adjoined simultaneously or in a fixed order. Furthermore, trees in the tree sets are equipped with dominance links. A dominance link can relate the foot node of a tree to any node in any tree of the same set. The dominance links provide a constraint on possible derivations: after a derivation is completed, each dominance link must hold in the derived tree.

We give a V-TAG derivation for sentence (2). Scrambling will be modeled by multi-component adjunction, while Topicalization is derived by standard adjunction. The grammar is a set of tree sets. Each tree set contains a head (e.g., a verb) and its projections, and slots for its arguments. Two examples are shown in Figure 1. We use IP (= S) and CP (= S′) as projections of the verb. In the set for the *geben* 'to give' embedded clause, one nominal argument is in a separate auxiliary tree, reflecting the fact that it may be scrambled, and the other nominal argument is included in the verbal projection tree, reflecting the fact that it is (long-distance) topicalized. The dotted line represents the dominance link. In the set for the *versuchen* 'to try' matrix clause, the only nominal argument is in a separate auxiliary tree. Its clausal subcategorization requirement is indicated by the fact that the verb is in an auxiliary tree (rooted in C′), forcing adjunction into an embedded clause.

The derivation now proceeds by first adjoining the matrix clause into the embedded clause at the C′ node, yielding the structure on the left in Figure 2. This adjunction implements the long-distance topicalization of the embedded direct argument. We are left with two auxiliary trees that still need to be adjoined, representing the scrambled arguments. We first adjoin the

[2] Without being formally defined, dominance links have been used previously in linguistic work, for example (Kroch, 1989).

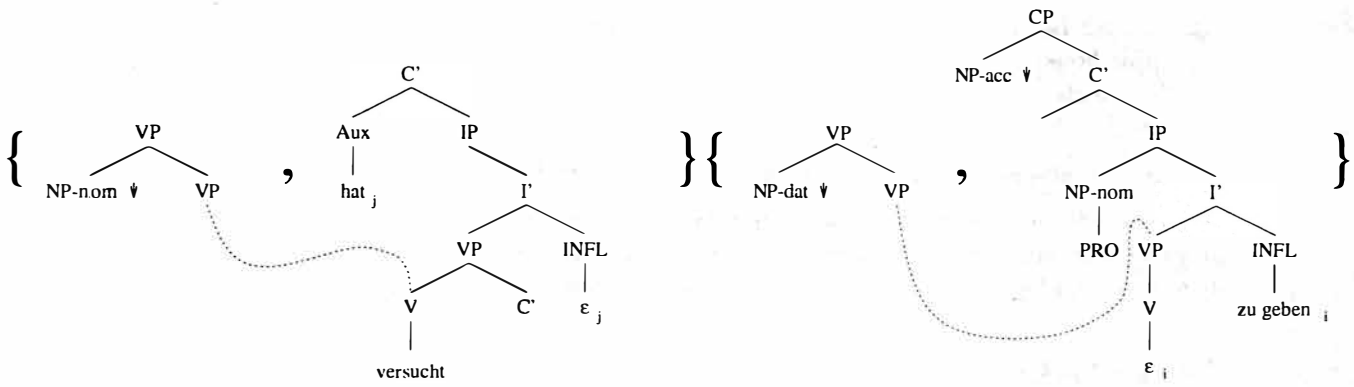



Figure 1: Initial tree set for *versuchen* matrix clause and *geben* embedded clause

matrix subject into its own clause, and then adjoin the embedded indirect object just above the matrix subject. The result is shown in Figure 2 on the right.

Observe that the tree sets given in Figure 1 have the property that they each represent a verb. In linguistic applications of TAG and related formalisms such as V-TAG, it is useful to associate each elementary structure (tree set in the case of V-TAG) with at least one lexical item (i.e., terminal symbol). Such a grammar is called "lexicalized". This has an important consequence, namely that derivations in a lexicalized grammar are always bounded in length by a linear function of the length of the derived sentence. In the following discussion of a parser for V-TAG, we will make crucial use of this property.

# 3 Parsing V-TAG

Both parsing algorithms presented in this paper deal only with adjunction, the core operation in TAG and in V-TAG. We ignore the substitution operation as well as constraints on adjunction which can easily be added and do not contribute to the complexity of the parsing algorithms.

In this section, we use an extension of the CYK-type parser for TAG defined by Vijay-Shanker (1987, p.110) to give a polynomial time parser for a large subset of the V-TAG languages. We first describe Vijay-Shanker's parser for simple TAG, and then describe the extensions necessary for V-TAG.

The main idea of Vijay-Shanker's parser is the introduction of a 4-dimensional matrix $T$, in which an entry of a node $\eta$ from an elementary tree $\tau$ at $T[i,j,k,l]$ represents the fact that either

(i) there is some derived tree $\tau'$ such that $\eta$ is its root node and $\eta$ dominates the substring $a_{i+1} \cdots a_j \eta_1 a_k \cdots a_l$ where $\eta_1$ is the (label of the) foot node of $\tau$ or

(ii) there is some derived tree $\tau'$ such that $\eta$ is its root node and $\eta$ dominates the substring $a_{i+1} \cdots a_l$ and $j = k$.

The parser fills the matrix $T$ bottom-up, starting from entries for the leaves. (We assume that the grammar is in extended two form, i.e., in every tree every node has at most two children.) In the presentation here, we split every node into a top and a bottom version, similar to the definition of "top" and "bottom" features in a feature-based TAG (Vijay-Shanker, 1987). If $\eta$ is a node in some tree of some set of a V-TAG, then $\eta^{\mathrm{T}}$ denotes the top version of that node, and $\eta^{\mathrm{B}}$ the bottom version. There are six cases which fall into two basic categories:

**(i)** Cases 1 to 4 and correspond to context-free expansions within one elementary tree. In Cases 1 to 3, two top versions of sibling nodes $\eta_1$ and $\eta_2$ are combined and the bottom version of their parent $\eta$ is added to T if we know that $\eta_1$ and $\eta_2$ cover a contiguous string (possibly interrupted only by the foot node). In Case 1 the footnote is dominated by $\eta_1$, in Case 2 it is dominated by $\eta_2$ and in Case 3 it is not dominated by either. Case 4 deals with nodes without siblings. Figure 3 shows Case 1.

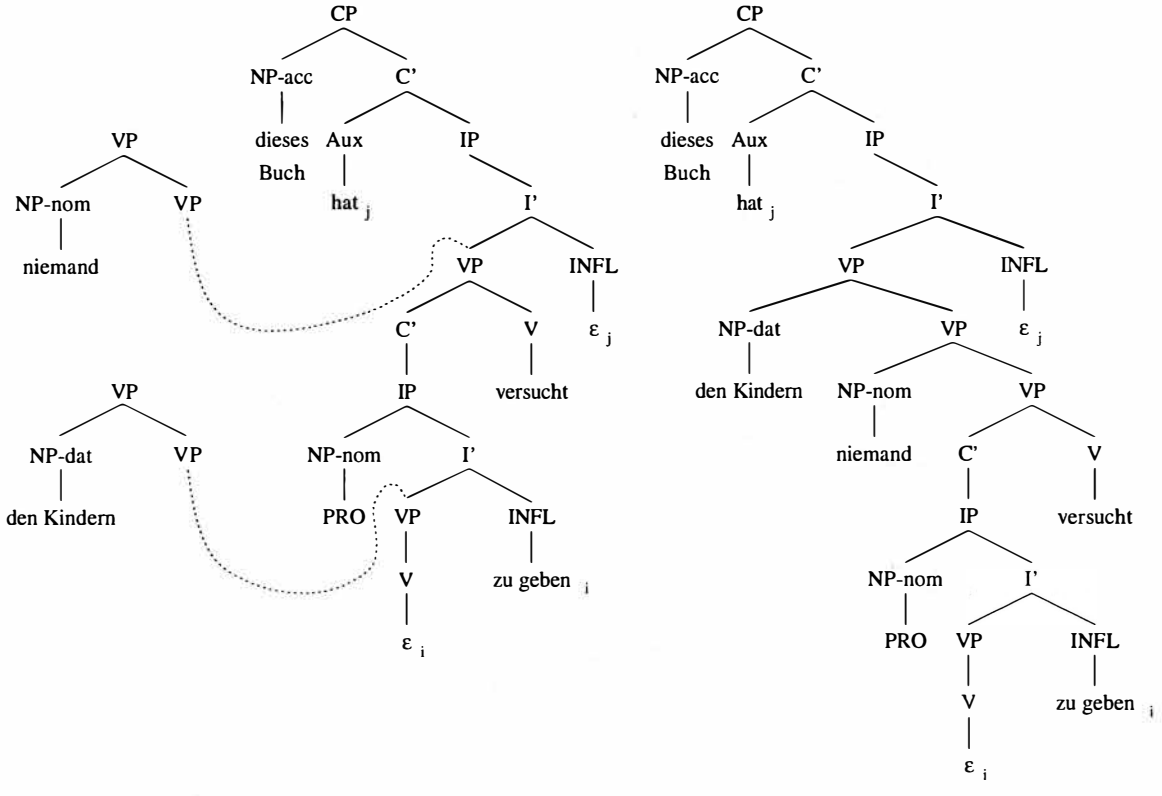



Figure 2: After adjoining matrix clause into subordinate clause (left) and final derived tree (right)

**(ii)** Cases 5a, 5b, and 6 deal with adjunction. Cases 5a and 5b correspond to adjunction (either at a node which dominates the foot node (5a) or not (5b)). The top version of the node is added to the matrix to reflect the string covered after adjunction at that node has taken place, as illustrated in Figure 3 for Case 5a. Case 6 corresponds to no adjunction: the top version of a node is added if the bottom version is already present in the same cell of the matrix.

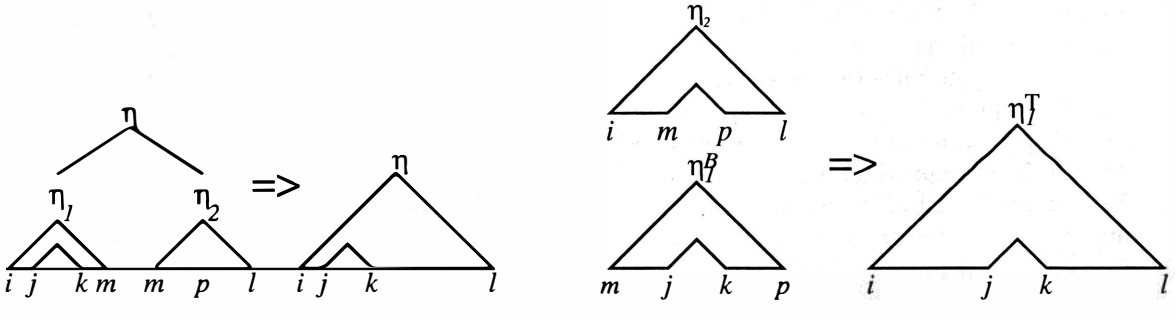



Figure 3: Cases 1 and 5a.

We now turn to the extensions necessary to handle V-TAG. We first introduce some additional terminology. If two nodes $\eta_1$ and $\eta_2$ are linked by a dominance link such that $\eta_1$ dominates $\eta_2$, then we will say that $\eta_1$ has a *required bottom* and that $\eta_2$ has a *required top*. If the tree of which $\eta_1$ ($\eta_2$) is a node has been adjoined during a derivation, but the tree of which $\eta_2$ ($\eta_1$) is a node has not, the requirement (top or bottom) will be called *unfulfilled*. The multiset of unfulfilled required tops of a node $\eta$ will be denoted by $\top(\eta)$, and the multiset of all required bottoms will be denoted by $\bot(\eta)$. We extend this notation to derived trees in the obvious way. Observe that a (partial) derived initial tree (i.e., a tree without a foot node on its frontier)

cannot have any unfulfilled required bottoms if it is to be part of a successful derivation.

Clearly, in a lexicalized V-TAG, in every derivation $|\top(\tau)|$ and $|\bot(\tau)|$ are always linear with respect to the length of the input string. Note that a V-TAG such that $|\top(\tau)|$ and $|\bot(\tau)|$ are always bounded by a constant $c$ in every derivation is equivalent to a TAG.

In order to keep track of unfulfilled dominance requirements, we add to each entry in the matrix two link-counters which record the number and type of required tops and bottoms, respectively, which still need to be fulfilled. A link-counter $\gamma$ is an array whose elements are indexed on the dominance links of $G$, and whose values are integers.[3] The sum of two counters is defined component-wise, the norm $|\gamma|$ is defined as the sum of the values of all components. We will denote by $\gamma^\top$ the required tops counter, by $\gamma^\bot$ the required bottoms counter, and by 0 the counter all of whose values are 0.

The need for maintaining two link counters – one for required tops and one for required bottoms – arises from the fact that entries in the four-dimensional parse matrix do not span a contiguous substring of the input string. Thus a (partial) derivation of a (derived) auxiliary tree may have required bottoms as well as required tops which will only be fulfilled once this structure is adjoined into another structure.

We now spell out what happens to the link counters in the cases 1 and 5a. The other cases are analogous. In the following, $a \dot{-} b$ is defined to be $a - b$ if $a \geq b$, and 0 otherwise.

**Case 1:** $\eta_1$ dominates the foot node (see Figure 3). If there is $(\eta_1^\top, \gamma_1^\bot, \gamma_1^\top) \in T[i,j,k,m]$ and $(\eta_2^\top, 0, \gamma_2^\top) \in T[m,p,p,l]$, $i \leq j \leq k \leq m \leq p \leq l$, then add $(\eta^\mathrm{B}, \gamma_1^\bot, \gamma_1^\top + \gamma_2^\top + \top(\eta))$ to $T[i,j,k,l]$.

**Case 5a:** $\eta_1$ dominates the foot node. If there is $(\eta_1^\mathrm{B}, \gamma_1^\bot, \gamma_1^\top) \in T[m,j,k,p]$ and $(\eta_2^\top, \gamma_2^\bot, \gamma_2^\top) \in T[i,m,p,l]$ where $\eta_2$ is the root node of an auxiliary tree with the same symbol as $\eta_1$, then add $(\eta_1^\top, (\gamma_2^\bot \dot{-} \gamma_1^\top) + \gamma_1^\bot, (\gamma_1^\top \dot{-} \gamma_2^\bot) + \gamma_2^\top)$ to $T[i,j,k,l]$.

In all six cases, after calculating the new $\gamma^\bot$ and $\gamma^\top$, the entry is discarded if $|\gamma^\bot + \gamma^\top| \geq c \cdot n$, where $c$ is the maximal number of links in a tree set of the grammar. The recognition of a string $a_1 \cdots a_n$ is successful if for some $j$, $0 \leq j \leq n$, and some $\eta$, a root node of an initial tree, we have $(\eta^\top, 0, 0) \in T[0,j,j,n]$.

In case 1 (see figure 3) $\eta_1^\top$ represents a partial derivation of a (derived) auxiliary tree where $\eta_1$ dominates the foot node. The required bottoms link-counter $\gamma_1^\bot$ represents links that go down from some nodes in the partially derived tree. These nodes must be on the spine of the tree and they can only be fulfilled after the tree is adjoined into another tree. Therefore, these links must 'go through' the foot node. For the node $\eta$ dominating $\eta_1$ the required bottom counter is simply copied. (Recall that only foot nodes can be at the top end of a link.) $\eta_2$ cannot contribute any required bottoms since the substring underneath $\eta_2$ is already completely derived and no nodes will added underneath $\eta_2$. The required tops however are simply added from $\gamma_1^\top$, $\gamma_2^\top$ and $\top(\eta)$ since they can be fulfilled by nodes which are added later above $\eta$.

In case 5a $\eta_1^\mathrm{B}$ represents a partial derivation of a (derived) auxiliary tree where $\eta_1$ dominates the foot node. All of its required bottoms ($\gamma_1^\bot$) must be added to the new required bottoms link-counter. However, some of the required bottoms of the adjoined auxiliary tree ($\gamma_2^\bot$) might have been fulfilled by required tops at node $\eta_1$, so only $\gamma_2^\bot \dot{-} \gamma_1^\top$ many are added to the new required bottoms link-counter. For the same reason, not all required tops of $\eta_1$ are added to the new required tops link-counter, but only $\gamma_1^\top \dot{-} \gamma_2^\bot$ many. However, all required tops of $\eta_2$, i.e. $\gamma_2^\top$ are added to the new required tops link-counter.

The core of the algorithm is a loop through indices $i,j,k,l \in \{0..n\}$, $i \leq j \leq k \leq l$, applying Cases 1 through 6 until the matrix is unchanged.

Using back pointers (e.g., for every $(\eta, \gamma^\bot, \gamma^\top)$ which is added to $T$, pointers to the contributing nodes $\eta_1$ (and $\eta_2$) in their respective positions are added), the matrix $T$ can be augmented to

[3] Link-counters are used for an extension to CFG (called UVG-DL) in (Rambow, 1994), and for a different CFG-based system with dominance links (called D-Tree Grammar) in (Rambow et al., 1995b). The present paper is similar to (Rambow et al., 1995b) in that it it is concerned with a formal system that includes dominance links. In both papers, these are parsed using multisets, and the parsers are polynomial for the same reason. However, the fact that V-TAG is a tree rewriting system which includes adjunction provides much of the conceptual complexity of the algorithms presented here.

represent a parse forest from which all derivations of an accepted string can be constructed.
**Theorem:** A lexicalized V-TAG is parsable in deterministic polynomial time.

The correctness of the recognition algorithm for TAG is proven by Vijay-Shanker (1987). It can easily be seen by induction on the number of dominance links that the link-counters correctly impose the dominance constraints.

The time complexity of the algorithm is that of Vijay-Shanker's algorithm, $O(n^6)$, multiplied by a factor representing the maximal number of elements of each cell of matrix $T$. Since $|\gamma^{\blacktriangle}|, |\gamma^{\top}|$ are in $O(n)$, we have that the number of possible pairs of link-counters is bounded by $O(n^{2|L|})$ (where $|L|$ is the total number of links in $G$), and the the time complexity of the algorithm is in $O(|G|n^{4|L|+6})$.

While for a linguistic grammar with significant coverage $|L|$ may be quite large, in practice the time complexity of the parser will be much lower. This is because in general, the number of different link-counters associated with the same node in any square of the parse matrix will be low. Several factors contribute to this fact.

First, recall that we are using multicomponent adjunction only for scrambling (and perhaps certain forms of "long" topicalization out of picture NPs), and not for raising and standard topicalization. These syntactic phenomena are still derived by simple adjunction, which does not contribute to the link counters.

Second, since CKY is a pure bottom-up parser, we need not actually distinguish between required top links which have the same tree at their top end. In fact, in scrambling languages such as German, the set of trees at the top end of dominance links is quite limited: it consists of substitution structures for nominal and clausal arguments, perhaps one for each case or verbal form. Thus $|L|$ in fact is a small number (say, four to six) rather than the large number one obtains if one counts the dominance links in the elementary structures for every single verb in the language.

Finally, observe that if we choose a constant $c$ and fix the number of open dominance links at any point in the derivation to be less than or equal to $c$, then we obtain a parser that simply runs in time $O(n^6)$. This is because the numbers of possible entries in the square of the parse matrix is again bounded by a grammar constant, part of which is $c$. In this case, we can view the formalism and the parser as a dynamic implementation of "slash" categories. Since in fact multiple scrambling is quite rare in real text, we can choose such a constant – say, $c = 3$ – and obtain a $O(n^6)$ parser for all but a tiny percentage of the sentences of the language in question. Determining the proper number may require empirical investigations.

The data structures which are built up in our parsing algorithms also yield themselves to an iterative algorithm if we gradually increase the maximum link constant $c$. The steps of such an iterative algorithm all have time complexity $O(n^6)$. Careful bookkeeping minimizes overhead with respect to a non-iterative approach. An iterative strategy will find "unscrambled" parses earlier than "scrambled" parses. In particular, we can continue the iteration process only if the "unscrambled" parses do not meet semantic or pragmatic conditions. This approach avoids many false ambiguities that arise when the parser postulates multiple scrambling, when in fact none has occurred.

# 4 An Earley-Type Parsing Algorithm

We now briefly describe an extension of the Earley-based TAG parsing algorithm of (Schabes, 1990) which results in a practical parsing algorithm for V-TAG. Again, we use link counters to keep track of unfulfilled dominance requirements. We will only give an informal review of the original parsing algorithm. For full details, we refer the reader to (Schabes, 1990). However, some detail is necessary to explain our extensions.

The basic data structure is a 'dotted tree'. A dotted tree is an elementary tree, usually an auxiliary tree, with a dot marking a node in this tree, together with two intervals (i.e. four indices) which represent the recognized strings underneath the marked node; one to the left and one to the right of the foot node. For a dot on a node, there are four possible positions: it

is either to the left (prediction phase) or the right (completion phase) and independent of this it is either above or below the node.

The parser proceeds in a mixed fashion with top-down prediction steps and a bottom-up completion steps. Beginning with an initial tree and the dot in the left-above position on the root node, the dot is propagated in a depth-first left-to-right fashion through the entire tree until it returns to the root node (to the right-above position). Dotted trees are collected in states sets $S_i$ which contain all dotted trees that represent a partial derivation that covers the input string up to position $i$.

In order to extend the algorithm to V-TAG, we again use 'link counters' to store the information about unfulfilled dominance constraints. Every dotted tree is extended by a link counter that for each link in the grammar counts the number of required bottoms in the partial derivation which is represented by the dotted tree. Since every state in the parsing algorithm is reached through a sequence of top-down and bottom-up steps which begins with the root node of the initial tree, there is never a need to keep track of required tops. As long as the dot is on the left, it moves downward, collecting required bottoms. If it hits required tops on a node, it either subtracts from the list of required bottoms or, if there are none, it immediately signals failure, since all nodes above a given node in a partial derivation have been visited already. When the dot is on the right, the bottom-up steps always refer to already visited structures (see especially the tests in the 'move dot up' and 'right-prediction' steps).

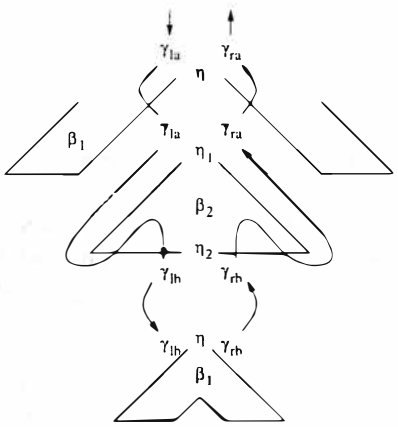


Figure 4: Percolation of the link counter.

Figure 4 shows the movements of the dot and the corresponding link counters on one node $\eta$ for all relevant steps of the parsing algorithm. The node $\eta$ is shown twice, with the dots in the above positions on the top of the figure, an auxiliary tree that might adjoin is shown in the middle, and on the bottom again the node $\eta$ is shown with the dots in the below positions.

In the first step with the dot in the left-above position of node $\eta$ of a dotted tree $\beta_1$, the corresponding link counter is $\gamma_{1-la}$. It contains all required bottoms in this partial derivation. Note that for this partial derivation, the dot has been moved through all positions above and to the left of $\eta$. The left-prediction step creates an entry for every auxiliary tree $\beta'$ that can be adjoined, with the dot in the left-above position on the root node. The link counter $\gamma_{1-la}$ is copied and if the root node of $\beta'$ fulfills some required bottoms, the link counter is decremented accordingly. When moving the dot has proceeded through $\beta'$ to the left-below position of the foot node, a left-completion step for $\beta_1$ can take place. The link-counter $\gamma'_{lb}$ is copied as $\gamma_{lb}$. An alternative left-completion step assumes that no adjunction has taken place on node $\eta$ and $\gamma_{la}$ is copied as $\gamma_{lb}$. If node $\eta$ of $\beta_1$ is itself a foot node and introduces a required bottom, then the link-counter $\gamma_{lb}$ is incremented accordingly.

The next move of the dot is the 'move dot down' to the left-above position of the leftmost daughter of node $\eta$. If this daughter node fulfills some required bottom, the link-counter is decremented accordingly.

Now the dot moves until it reaches the right-below position of $\eta$. If the node $\eta$ itself is a foot node and has a required bottom, the relevant component of the link-counter $\gamma_{lb}$ is decremented. If this is impossible (because the value is zero), then the derivation fails. Then, the new link-counter $\gamma_{rb}$ must be compared with the old link-counter $\gamma_{lb}$. If any component of the link counter $\gamma_{rb}$ is greater than $\gamma_{lb}$, something has gone wrong and the dot is not moved. This can only happen if new required bottoms are added, but not fulfilled below $\eta$. Since for this partial derivation, the dot has moved through the entire derived tree below $\eta$, these required bottoms can never be fulfilled. The old link-counter $\gamma_{lb}$ must be retrieved from the appropriate set $S_i$.

The next step is the (right-)prediction of an adjunction at node $\eta$. If there is a dotted (auxiliary) tree $\beta'$ covering an immediate preceding substring with the dot in the left-below position

of the foot node, this dot is moved to the right-below position. The counter $\gamma_{rb}$ is copied as $\gamma_{2-rb}$.

Again, there are two alternative right-completion steps. Assuming that no adjunction has taken place, the dot is moved up to the right-above position. The new link-counter $\gamma_{ra}$ is copied from $\gamma_{rb}$.

If there is a dotted (auxiliary) tree $\beta'$ covering an immediate surrounding substring in which the dot has moved all the way up to the right-above position of the root node, the adjunction of this tree is assumed and the new link-counter $\gamma_{ra}$ is copied from $\gamma_{2-ra}$.

From there, the dot is moved up, either to the left-above position of the right sister node, or, if there is no right sister, to the right-below position of the mother node. In the first case the link-counter is copied without changes, in the second case the above mentioned test is performed,

Also, in the scanner steps, the link counter is copied without changes.

Note that the only point in which the comparison of new and old link-counters is necessary is the arrival of the dot in a right-below position. In all moves, after calculating the new link-counter $\gamma$, the entry is discarded if $|\gamma| \geq c \cdot n$, where $c$ is the maximal number of links in a tree set of the grammar. The recognition of a string $a_1 \cdots a_n$ is successful if $[\alpha, 0, ra, 0, -, -, n]$ is in $S_n$ with $\alpha$ an initial tree.

The core of the algorithm is a loop from $i := 0$ to $n$ through the sets $S_i$ where the modified steps of the Earley-style parser are applied until no more states can be added.

The correctness of the recognition algorithm for TAG is proven by Schabes (1990). It can easily be seen by induction on the number of dominance links that the link-counters correctly impose the dominance constraints.

The time complexity of the algorithm is that of Schabes' algorithm, $O(n^6)$, multiplied by a factor representing the maximal number of entries in the states sets $S_i$ which differ only in the link counters $\gamma$. Since $|\gamma| \leq cn$, we have that the number of possible link-counters is bounded by $O(n^{2\times|L|})$ (where $|L|$ is the total number of links in $G$), and the the time complexity of the algorithm is in $O(|G|n^{2\times|L|}n^6)$.

Again, using back pointers, the entries in the states sets can be augmented to represent a parse forest from which all derivations of an accepted string can be constructed. The discussion about the relevance of the exponents and an iterative algorithm from section 3 also applies to this Earley-style parsing algorithm.

# Bibliography


Becker, Tilman; Joshi, Aravind; and Rambow, Owen (1991). Long distance scrambling and tree adjoining grammars. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL'91)*, pages 21–26. ACL.

Joshi, Aravind K. (1987). An introduction to Tree Adjoining Grammars. In Manaster-Ramer, A., editor, *Mathematics of Language*, pages 87–115. John Benjamins, Amsterdam.

Kroch, Anthony (1989). Asymmetries in long distance extraction in a Tree Adjoining Grammar. In Baltin, Mark and Kroch, Anthony, editors, *Alternative Conceptions of Phrase Structure*, pages 66–98. University of Chicago Press.

Rambow, Owen (1994). *Formal and Computational Aspects of Natural Language Syntax.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia. Available as Technical Report 94-08 from the Institute for Research in Cognitive Science (IRCS).

Rambow, Owen; Vijay-Shanker, K.; and Weir, David (1995a). D-tree grammars. In *33rd Meeting of the Association for Computational Linguistics (ACL'95).* ACL.

Rambow, Owen; Vijay-Shanker, K.; and Weir, David (1995b). Parsing D-Tree Grammars. Fourth International Workshop on Parsing Technologies.

Schabes, Yves (1990). *Mathematical and Computational Aspects of Lexicalized Grammars.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania.

Vijay-Shanker, K. (1987). *A study of Tree Adjoining Grammars.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA.

# YET ANOTHER $\mathcal{O}(n^6)$ RECOGNITION ALGORITHM FOR MILDLY CONTEXT-SENSITIVE LANGUAGES

Pierre BOULLIER
INRIA-Rocquencourt, BP 105, 78153 Le Chesnay Cedex, France
E-mail: Pierre.Boullier@inria.fr

**Abstract**

Vijay-Shanker and Weir have shown in [17] that Tree Adjoining Grammars and Combinatory Categorial Grammars can be transformed into equivalent Linear Indexed Grammars (LIGs) which can be recognized in $\mathcal{O}(n^6)$ time using a Cocke-Kasami-Younger style algorithm. This paper exhibits another recognition algorithm for LIGs, with the same upper-bound complexity, but whose average case behaves much better. This algorithm works in two steps: first a general context-free parsing algorithm (using the underlying context-free grammar) builds a shared parse forest, and second, the LIG properties are checked on this forest. This check is based upon the composition of simple relations and does not require any computation of symbol stacks.

*Keywords:* context-sensitive parsing, ambiguity, parse tree, shared parse forest.

## 1 Introduction

It is well known that natural language processing cannot be described by purely context-free grammars (CFGs). On the other hand, general context-sensitive formalisms are powerful enough but cannot be parsed in reasonable time. Therefore, various intermediate frameworks have been investigated, the trade-off being between expressiveness power and computational tractability. One of these formalism classes is the so-called mildly context-sensitive languages which can be described by several equivalent grammar types. Among these types, Tree Adjoining Grammars (TAGs) are attractive because they can express some natural language phenomena (see Abeillé and Schabes [1]) and many systems are based upon this framework (see for example [10] and [6]). Formal properties of TAGs have been studied (see Vijay-Shanker and Joshi [16], and Vijay-Shanker [15]) and a recognizer for TAGs (see [17]), based upon a Cocke-Kasami-Younger method ([4] and [18]), works in $\mathcal{O}(n^6)$ worst time. Unfortunately, with this algorithm, this complexity is always reached. More practical methods, which are usually based upon the Earley parsing algorithm [3], have also been investigated (see for example Schabes [13] and Poller [11]). Though the $\mathcal{O}(n^6)$ worst-case time is not improved, for some inputs, the actual complexity may be much better. However, the design of a better worst-case recognizer remains an open problem.

In [17], Vijay-Shanker and Weir have shown that mildly context-sensitive grammars have the same formal power and that TAGs and Combinatory Categorial Grammars (CCGs) can be transformed into equivalent Linear Indexed Grammars (LIGs).

An Indexed Grammar [2] is a CFG in which each object is a non-terminal associated with a stack of symbols. The productions of this grammar class define on the one hand a derived relation in the usual sense and, on the other hand, the way symbols are pushed or popped on top of the stacks which are associated with each non-terminal. A restricted form of Indexed Grammar called LIG allows only for the stack associated with the left-hand side (LHS) non-terminal of a given production to be associated with at most one non-terminal in the right-hand side (RHS).

This paper presents a new recognition algorithm for LIGs. It works in two main steps:

1. a CF-parsing algorithm, working on the underlying CF-grammar, builds a shared parse forest;
2. the LIG conditions (see Section 3) are checked on that forest.

Since that second step does not depend on the way the shared parse forest is built, any general CF-parsing algorithm can be used in the first step. As previously mentioned, CKY or Earley parsing

algorithms are candidates but others too. In particular, generalized LR parsing methods (see Lang [7], Tomita [14], and Rekers [12]) are good challengers. In fact, the CF-parsing algorithm of our prototype system upon which this paper ideas have been tested, implements a non-deterministic LR (or LC at will) parsing algorithm using a graph-structured stack. Under certain conditions, for a given input string of length $n$, the CF-parsing takes a time $\mathcal{O}(n^3)$ in the worst case and moreover the output shared parse forest of size $\mathcal{O}(n^3)$ (in the worst case) is such that each elementary tree can be seen as an occurrence (after some non-terminal renaming) of a production in the underlying CFG. Following Lang [8], in Section 2, we stress this analogy by defining a shared parse forest as a CFG.

Obviously, the checking of the LIG properties can be performed on a forest by computing the stacks of symbols along paths, and in associating with each (shared) node a set of such stacks. As in [17], in devising an elaborate sub-stack sharing mechanism, this check could be performed in $\mathcal{O}(n^6)$ time. In Section 4, we take a different approach: this check is performed without the computation of any stack of symbols (and hence without having to design any sub-stack sharing structure). Given a single tree of the (unfolded) shared parse forest, we identify *spines* as being paths along which individual stack of symbols are evaluated. The origin of such a spine corresponds to the birth of a stack which evolves according the LIG stack schemas and which finally vanishes at the end of the spine. The checking of LIG conditions relies on the simple observation that, for a given spine, the stack actions must be bracketed. Each time a push or pop occurs at a node, there is a twin node where the opposite action, acting on the same symbol at the same stack level, should take place. In a shared parse forest, different spines may share nodes. In particular, a given couple of twin nodes may be shared among several spines, with the corresponding check being done only once. In Section 5 we show that this check sharing, expressed as relations between twin nodes, results in a worst case $\mathcal{O}(n^6)$-time LIG recognition. Our algorithm is illustrated by an example in Section 6. Since TAGs or CCGs can be transformed into equivalent LIGs [17], this complexity extends over mildly context-sensitive languages.

## 2 Parse Tree and Shared Parse Forest

The goal of this section is to set up the vocabulary and to define our vision of shared parse forests.

Let $G = (V_N, V_T, P, S)$ be a CFG where:

- $V_N$ is a non-empty finite set of *non-terminal* symbols.
- $V_T$ is a finite set of *terminal* symbols; $V_N$ and $V_T$ are disjoint; $V = V_N \cup V_T$ is the *vocabulary*.
- $S$ is an element of $V_N$ called the *start symbol*.
- $P \subseteq V_N \times V^*$ is a finite set of productions. Each production is denoted by $A \rightarrow \sigma$ or by $r_p, 1 \leq p \leq |P|$; such a production is called an $A$-production.

We adopt the convention that $A, B, C$ denote non-terminals, $a, b, c$ denote terminals, $w, x$ denote elements of $V_T^*$, $X$ denotes elements of $V$, and $\beta, \sigma$ denote elements of $V^*$.

On $V^*$ we define $|P|$ disjoint binary relations named *right derive by* $B \rightarrow \beta$[1] and denoted by $\underset{G}{\overset{B\rightarrow\beta}{\Rightarrow}}$ (or simply $\overset{B\rightarrow\beta}{\Rightarrow}$ when $G$ is understood) as the set $\{(\sigma B x, \sigma\beta x) \mid B \rightarrow \beta \in P\}$.

The relation *derive* denoted by $\Rightarrow$ is defined by:

$$\Rightarrow \;=\; \bigcup_{B\rightarrow\beta\in P} \overset{B\rightarrow\beta}{\Rightarrow}$$

Let $\sigma_1, \ldots, \sigma_i, \sigma_{i+1}, \ldots, \sigma_l$ be strings in $V^*$ such that $\forall i, 1 \leq i < l, \exists r_p \in P, \sigma_i \overset{r_p}{\Rightarrow} \sigma_{i+1}$ then the sequence of strings $(\sigma_1, \ldots, \sigma_i, \sigma_{i+1}, \ldots, \sigma_l)$ is called a *derivation*. Conversely, since $\overset{r_p}{\Rightarrow}$ and $\overset{r_q}{\Rightarrow}$ are disjoint when $p$ and $q$ are different, between any two consecutive strings $\sigma_i$ and $\sigma_{i+1}$ in a derivation, the relation $\overset{r_p}{\Rightarrow}$ whose $(\sigma_i, \sigma_{i+1})$ is an element is uniquely known.

[1] In the sequel the qualifier *right* will disappear since only right derivations, right sentential forms, etc ... are introduced.

A *$\sigma$-derivation* is a derivation starting with $\sigma$. A $\sigma$-derivation whose last element in the sequence is $\beta$ is called a *$\sigma/\beta$-derivation*. The elements of a $\sigma$-derivation are called a *$\sigma$-phrase*. A $\sigma$-phrase in $V_T^*$ is a *$\sigma$-sentence*. On the other hand an $S$-phrase is a *sentential form* and an $S$-sentence is a *sentence*.

The language defined by $G$ is the set of its sentences:

$$\mathcal{L}(G) = \{x \mid S \underset{G}{\overset{+}{\Rightarrow}} x \wedge x \in V_T^*\}$$

In an $S/x$-derivation, to accurately define the contribution of any symbol $X$ (its $X$-sentence) to the sentence $x$, we will define the notion of split of $x$ by $X$.

A triple $(x_1, x_2, x_3)$ is called *3-split* (or more simply *split*) of $x$ when $x = x_1x_2x_3$. If $n$ is the length of $x$, such a triple can also be denoted by $x_{i..j}$ with $0 < i \leq j \leq n+1$, $|x_1| = i-1$, $|x_2| = j-i$, and $|x_3| = n-j+1$. Two splits of $x$, say $(x'_1, x'_2, x'_3)$ and $(x''_1, x''_2, x''_3)$ can be composed into one split iff we have $x'_1x'_2 = x''_1$. In such a case the resulting split of $x$ is $(x'_1, x'_2x''_2, x''_3)$. Assume that $x_{i..j}$ denotes $(x'_1, x'_2, x'_3)$ and that $x_{k..l}$ denotes $(x''_1, x''_2, x''_3)$, the previous condition simply means that $j = k$ and that the composed split is denoted by $x_{i..l}$. We allow $x$ itself to designate the split $(\varepsilon, x, \varepsilon) = x_{1..|x|+1}$.

We call *split of $x$ by $X$* the couple $(X, (x_1, x_2, x_3))$ if there is an $S/x$-derivation $S \overset{*}{\Rightarrow} \sigma X x_3 \overset{*}{\Rightarrow} \sigma x_2 x_3 \overset{*}{\Rightarrow} (x_1x_2x_3 = x)$. This couple, when $x$ is understood, could be denoted by $[X]_i^j$ if $(x_1, x_2, x_3) = x_{i..j}$ or even by $[X]$ when the split of $x$ is not necessary. This definition and notations extends from symbols to strings.

Now we can define our vision of parse trees.

**Definition 1** *Let $G = (V_N, V_T, P, S)$ be a CFG, $x$ a sentence in $\mathcal{L}(G)$, and $d^x$ an $S/x$-derivation. We call parse tree (w.r.t. $G$ and $d^x$) the CFG $G^{d^x} = (V_N^x, V_T^x, P^{d^x}, S^x)$ where:*

- $V_N^x = \{(B, (x_1, x_2, x_3)) \mid B \in V_N \wedge x = x_1x_2x_3\}$.
- $V_T^x = \{(a, (x_1, a, x_3)) \mid a \in V_T \wedge x = x_1ax_3\}$.
- $S^x = [S] = (S, (\varepsilon, x, \varepsilon))$.
- $P^{d^x} = \{[B] \rightarrow [X_1]\ldots[X_k]\ldots[X_p] \mid B \rightarrow X_1\ldots X_k\ldots X_p \in P\}$ *and* $d^x = S \overset{*}{\Rightarrow} \sigma B x_3 \Rightarrow \sigma X_1\ldots X_k\ldots X_p x_3 \overset{*}{\Rightarrow} \sigma X_1\ldots X_{k-1}x_2^k\ldots x_2^p x_3 \overset{*}{\Rightarrow} \sigma x_2^1\ldots x_2^k\ldots x_2^p x_3 \overset{*}{\Rightarrow} x$ *where:* $x = x_1x_2x_3$, $x_2 = x_2^1\ldots x_2^k\ldots x_2^p$, $[X_k] = (X_k, (x_1x_2^1\ldots x_2^{k-1}, x_2^k, x_2^{k+1}\ldots x_2^p x_3))$, *and* $[B] = (B, (x_1, x_2, x_3))$.

Parse trees are trees in which the start symbol $S^x$ is the root, non-terminal symbols are the internal nodes while terminal symbols are the leaves. Obviously we have $\mathcal{L}(G^{d^x}) = \{[x]\}$. In fact, for any two consecutive strings $\sigma B x_3$ and $\sigma\beta x_3$ in $d^x$, we have $[\sigma][B][x_3] \underset{G^{d^x}}{\overset{[B]\rightarrow[\beta]}{\Rightarrow}} [\sigma][\beta][x_3]$. It is easy to see that this definition of a parse tree is a tree in the usual sense only when the derivation $d^x$ does not involve any cycle (i.e. $\nexists A, A \overset{+}{\Rightarrow} A$). If there is a cycle, our definition denotes, by a single parse tree, the ambiguities denoted by the unbounded number of (usual) trees when this cycle is taken 0, 1, 2, ... times. The whole notion of ambiguity will be captured by the following definition of shared parse forest.

**Definition 2** *Let $G = (V_N, V_T, P, S)$ be a CFG, and $x$ a sentence in $\mathcal{L}(G)$. The shared parse forest for $x$ (w.r.t. $G$) is the CFG, $G^x = (V_N^x, V_T^x, P^x, S^x)$ where:*

- $V_N^x = \{(B, (x_1, x_2, x_3)) \mid B \in V_N \wedge x = x_1x_2x_3\}$.
- $V_T^x = \{(a, (x_1, a, x_3)) \mid B \in V_T \wedge x = x_1ax_3\}$.
- $S^x = (S, (\varepsilon, x, \varepsilon))$.
- $P^x = \bigcup_{d^x \in D^x} P^{d^x}$ *where $D^x$ is the set of all $S/x$-derivations, and $P^{d^x}$ is the production set of the parse tree $G^{d^x} = (V_N^x, V_T^x, P^{d^x}, S^x)$ associated with any derivation $d^x$ in $D^x$.*

Any production $r_p = [B] \rightarrow [X_1] \ldots [X_q]$ in $P^x$ is mapped by the unary operator $^-$ to its associated production $\overline{r_p} = B \rightarrow X_1 \ldots X_q$ in $P$.

This vision of a set of parse trees as a CFG has several formal and practical advantages (thanks to [9]). It exhibits a particular case of a general result: the intersection of CF-languages (defined by $G$) and regular languages (the input string $x$) are CF-languages (the resulting shared parse forest $G^x$). $G^x$ can also be seen as a specialization of $G$ (productions in $G^x$ are productions in $G$, up to some renaming), which only defines (in all the same possible ways as $G$) the string $x$. This CFG allows to define an unbounded number of derivations (when $G$ is cyclic) in a finite way. A *context sharing* occurs when there are several occurrences of the same non-terminal in RHSs, while a *sub-tree sharing* occurs when there are several occurrences of the same non-terminal in LHSs. This sharing may even be considered as optimal if we impose (as done here) that productions (elementary trees) in a parse tree, have the same structure as their corresponding production in $G$.

Without any restriction on $G$, the size of $P^x$ is $\mathcal{O}(n^{l+1})$ where $l$ is the length of the longest RHS in $P$. If $G$ is unambiguous, (or if the parsing of $x$ does not exhibit any ambiguity,) this size is linear in $n$.

## 3 Linear Indexed Grammars (LIGs)

An indexed grammar is a CFG in which stack of symbols are associated with non-terminals. The derive relation, in addition to its usual meaning, handles these stacks of symbols. LIGs are a restricted form of indexed grammars in which the stack associated with the non-terminal in the LHS of any production is associated with at most one non-terminal in the RHS. Other non-terminals are associated with stacks of bounded size.

In fact, in a production, it is not a stack which is associated with a non-terminal, but rather a stack schema expressing a way to compute a stack. Let $V_l$ denotes a finite set of (stack) symbols, a stack is an element of $V_l^*$. A stack schema is an element of $V_b \times V_l^*$ where $V_b = \{\varepsilon, ..\}$. The stack schema $(..\alpha)$ where $\alpha \in V_l^*$ matches all the stacks whose prefix (bottom) part is left unspecified and whose suffix (top) part is $\alpha$. A stack may be considered as a stack schema whose first component (the element of $V_b$) is $\varepsilon$.

Following [17], we formally defined a LIG as follows:

**Definition 3** *A LIG, $L$ is denoted by $(V_N, V_T, V_l, P_L, S)$ where:*

- *$V_N$ is a non-empty finite set of non-terminal symbols.*
- *$V_T$ is a finite set of terminal symbols, $V_N$ and $V_T$ are disjoint, and $V = V_N \cup V_T$ is the vocabulary.*
- *$V_l$ is a finite set of stack symbols.*
- *$P_L$, the production set, is a finite subset of $(V_N \times V_b \times V_l^*) \times ((V_N \times V_b \times V_l^*) \cup V_T)^*$.*
- *$S \in V_N$ is the start symbol.*

We adopt the convention that $\alpha$ will denote members of $V_l^*$, $\pi$ elements of $V_b$, and $\gamma$ elements of $V_l$.

A triple $(A, \varepsilon, \alpha)$ in $V_N \times V_b \times V_l^*$ is called a *secondary object* and is denoted by $A(\alpha)$ while a triple $(A, .., \alpha)$ is called a *primary object* and is denoted by $A(..\alpha)$. The disjoint sets of primary and secondary objects are respectively denoted by $V_O^P$ and $V_O^S$. The set of objects denoted $V_O$ is $V_O^P \cup V_O^S$. The object $A(\pi\alpha)$, whose non-terminal component part is $A$, is called an $A$-object. We use $\Gamma$ to denote strings in $(V_O \cup V_T)^*$. $A(..\alpha)$ denotes an object whose stack suffix (stack top) is $\alpha$ and with an arbitrary prefix (stack bottom). $A()$ denotes that an empty stack schema is associated with the non-terminal $A$. $A(\alpha)$ denotes that the stack $\alpha$ is associated with the non-terminal $A$. Each production in $P_L$ is denoted by $A(\pi\alpha) \rightarrow \Gamma$ or $r_p()$[2] where $1 \leq p \leq |P_L|$.

The general form of a production in a LIG is:

$$r_p() = A(\pi\alpha) \rightarrow w_1 A_1(\alpha_1) \ldots w_{i-1} A_{i-1}(\alpha_{i-1}) w_i A_i(\pi\alpha_i) w_{i+1} A_{i+1}(\alpha_{i+1}) \ldots w_p A_p(\alpha_p) w_{p+1}$$

[2]The parentheses reminds us that we are in a LIG!

If the LHS object $A(\pi\alpha)$ is secondary (i.e. $\pi = \varepsilon$), we observe that all the objects (if any) in the RHS should also be secondary, while if $A(\pi\alpha)$ is primary (i.e. $\pi = ..$), there must be exactly one primary object in the RHS.

The above production is called an $A$-production. If this production is used, for any $\Gamma_1, \Gamma_2 \in (V_O \cup V_T)^*$ and $\alpha' \in V_l^*$, we define the binary relation *derive by* $r_p()$ on LIGs by:

$$\Gamma_1 A(\alpha'\alpha)\Gamma_2 \overset{r_p()}{\Rightarrow} \Gamma_1 w_1 A_1(\alpha_1) \ldots w_{i-1}A_{i-1}(\alpha_{i-1})w_i A_i(\alpha'\alpha_i)w_{i+1}A_{i+1}(\alpha_{i+1}) \ldots w_p A_p(\alpha_p)w_{p+1}\Gamma_2$$

when $\pi = .. \vee \alpha' = \varepsilon$.

We observe that the stack $\alpha'\alpha$ associated with the non-terminal $A$ in the LHS and the stack $\alpha'\alpha_i$ associated with the non-terminal $A_i$ in the RHS have the same prefix $\alpha'$.

**Definition 4** *We define the CF-backbone of a LIG as being its underlying CFG. Formally, if* $L = (V_N, V_T, V_l, P_L, S)$ *is a LIG, its CF-backbone is the CFG,* $G_L = (V_N, V_T, P_G, S)$, *or simply* $G$ *when* $L$ *is understood, where:*

$$P_G = \{A \rightarrow w_1 A_1 \ldots w_{i-1}A_{i-1}w_i A_i w_{i+1}A_{i+1} \ldots w_p A_p w_{p+1} \mid A(\pi\alpha) \rightarrow w_1 A_1(\alpha_1) \ldots w_{i-1}A_{i-1}(\alpha_{i-1})w_i A_i(\pi\alpha_i)w_{i+1}A_{i+1}(\alpha_{i+1}) \ldots w_p A_p(\alpha_p)w_{p+1} \in P_L\}.$$

If there is a one to one mapping between $P_L$ and $P_G$ the LIG is said to be *fair*. It is not very difficult to find an algorithm which transforms any LIG into an equivalent fair LIG. In the sequel we will only consider fair LIGs.

Due to the one to one mapping between (fair) LIGs and their CF-backbones we assume that iff $r_p()$ is a production in $P_L$, then $r_p$, with the same index $p$, denotes the corresponding production in its CF-backbone $P_G$.

Let $L = (V_N, V_T, V_l, P_L, S)$ be a LIG, $G = (V_N, V_T, P_G, S)$ its CF-backbone, $x$ a string in $\mathcal{L}(G)$, and $G^x = (V_N^x, V_T^x, P_G^x, S^x)$ its shared parse forest for $x$. Consider the LIG $L^x = (V_N^x, V_T^x, V_l, P_L^x, S^x)$ s.t. $G^x$ is its CF-backbone and each stack schema $(\pi_k\alpha_k)$ associated with the non-terminal $[A_k]$, occurring at position $k$ in production $r_p() \in P_L^x$ is the stack schema of the object at position $k$ in $\overline{r_p}() \in P_L$. More formally we have:

$$P_L^x = \{r_p() = [A_0](\pi_0\alpha_0) \rightarrow [w_1][A_1](\pi_1\alpha_1) \ldots [w_k][A_k](\pi_k\alpha_k) \ldots [w_{m+1}] \mid r_p = [A_0] \rightarrow [w_1][A_1] \ldots [w_k][A_k] \ldots [w_{m+1}] \in P_G^x \wedge \overline{r_p}() = A_0(\pi_0\alpha_0) \rightarrow w_1 A_1(\pi_1\alpha_1) \ldots w_k A_k(\pi_k\alpha_k) \ldots w_{m+1} \in P_L\}$$

$L^x$ is called the *LIGed forest* for $x$.

By construction, any LIGed forest is fair and between a LIG $L$ and its LIGed forest $L^x$ for $x$, we have $x \in \mathcal{L}(L) \iff x \in \mathcal{L}(L^x)$. (Recall that $x$ can designate the split $(\varepsilon, x, \varepsilon)$).

An object is said *initial* (resp. *final*) if it is secondary and occurs in the RHS (resp. LHS) of a production. $V_O^I$ (resp. $V_O^F$) denotes the set of initial (resp. final) objects.

**Definition 5** *For a given LIGed forest for* $x$, *we call spine, any sequence of* $2p$ $(1 \leq p)$ *objects* $(o_1, o_2, \ldots, o_{2i-1}, o_{2i}, o_{2i+1}, \ldots, o_{2p})$ *such that:*

- $o_1$ *(resp.* $o_{2p}$*) is an initial (resp. final) object.*
- *Inside objects* $o_j$ *(if any)* $(\forall j, 1 < j < 2p)$ *are primary.*
- $\forall i, 1 \leq i \leq p$, *two consecutive objects* $o_{2i-1} = X_1(\pi_1\alpha_1)$, *and* $o_{2i} = X(\pi\alpha)$ *are such that* $X_1 = X$, *and* $o_{2i-1}$ *(resp.* $o_{2i}$*) occurs in the RHS (resp. LHS) of a* $P_L^x$ *production.*

This notion of spine is fundamental in LIG theory since it represents a path upon which stacks of symbols are evaluated. For example, followed in the direct way (top-down), the spine $(o_1 = X_1(\alpha_1), o_2 = X_1(..\alpha_1'), \ldots, o_{2i-1} = X_i(..\alpha_i), o_{2i} = X_i(..\alpha_i'), \ldots, o_{2p} = X_p(\alpha_p'))$ indicates that:

- a stack $s$ is created and initialized with $\alpha_1$ on the initial object $o_1$;

- if $\alpha'_1$ is a suffix of $s$, then $\alpha'_1$ is popped from $s$ on object $o_2$;

⋮

- the string of symbols $\alpha_i$ is pushed on $s$ on object $o_{2i-1}$;
- if $\alpha'_i$ is a suffix of $s$, then $\alpha'_i$ is popped from $s$ on object $o_{2i}$;

⋮

- on the final object $o_{2p}$, if $\alpha'_p$ is a suffix of $s$, then $\alpha'_p$ is popped from $s$ and the stack $s$ is checked for emptiness.

A spine is said to be *valid* if each check sketched above succeeds[3].

## 4 Our LIG Recognition Algorithm

In this paper we restrict our attention to LIGs with the following characteristics:

1. the RHS of a production contains at most two symbols;
2. the stack schema ($\pi\alpha$) of any object (primary or secondary) is such that $0 \leq |\alpha| \leq 1$.

Recall that our recognition algorithm works on shared parse forests. Therefore, it is assumed that such a forest has been built by any general CF-parsing algorithm, working on the associated CF-backbone grammar, with a string $x$ as input.

The reason why we allow at most two symbols in the RHS of the CF-backbone is to build the forest in time $\mathcal{O}(n^3)$. Moreover, in such a case, the parameters of the shared parse forest are kept within some suitable upper bounds: in particular the number of productions is $\mathcal{O}(n^3)$, the number of non-terminal symbols is $\mathcal{O}(n^2)$, the number of $X$-productions for any given $X = (A, x_{i..j})$ is $\mathcal{O}(n)$ and the number of occurrences of such a non-terminal symbol $X$ in the RHSs is also $\mathcal{O}(n)$.

The restriction on stack schemas, have been chosen only for pedagogic facilities. This restriction does not change neither our algorithm principle nor its upper bound complexity. Moreover, it is easy to see that this form of LIG constitutes a normal form.

We will restrict our attention to non-cyclic CF-backbones. This restriction will guarantee that in any parse (sub-)tree, internal nodes are different from the root node. Nevertheless, this restriction is not mandatory and slight modifications of our algorithms allow to also handle cyclic grammars without changing their complexities.

Contrary to the previous section where we saw that a stack of symbols can be evaluated along spines, we choose not to compute stacks explicitly. The idea of our algorithm is based upon the remark that each time a symbol $\gamma$ is pushed on a stack at a given place, this very symbol should be popped at some other place. The converse should also be true. The following will exhibit a mean by which this property could be checked without explicitly computing neither stacks nor spines.

We could remark that we are not interested in finding all the valid spines between any pair of objects $(o_1, o_2)$, but only if there is at least one such valid spine. As a first consequence we will only consider *abridged spines* (a-*spine* for short) $(o_1, o_2, o_4, \ldots, o_{2i}, o_{2i+2}, \ldots, o_{2p})$ which summarize all the spines $(o_1, o_2, o_3, o_4, \ldots, o_{2i}, o_{2i+1}, o_{2i+2}, \ldots, o_{2p})$ where the RHSs (odd) objects (except the initial one) have been erased. If the length of a spine is $2p$, we see that the length of its a-spine is $p + 1$.

The first purpose of our algorithm is to compute the relation *valid spine* denoted by $\bowtie$ and which is the set of all couples $(o_1, o_2)$ s.t. $o_1$ is an initial object, $o_2$ is a final object, and there is at least one valid spine between $o_1$ and $o_2$.

In order to reach this goal, for a given LIGed forest for $x$, we define on its objects $V_O$, $2|V_l| + 1$ binary relations noted (for some $\gamma$ in $V_l$) $\overset{\gamma}{\prec}$, $\overset{\gamma}{\succ}$, and $\multimap\!\!\diamond\!\!\multimap$. These relations between objects indicate the evolution of an imaginary stack between the first and the second object.

[3]Of course it is possible to adopt the dual vision and to evaluate stacks along spines in the opposite (bottom-up) way. A stack is created and initialized with $\alpha'_p$ on the final object $o_{2p}$. Elements are pushed on LHS objects while they are checked and popped on RHS objects, and finally $\alpha_1$ is popped on $o_1$ and the stack is checked for emptiness.

The element $(o_1, o_2)$ of $\overset{\gamma}{\prec}$ (resp. $\overset{\gamma}{\succ}$) means that the stack associated with $o_2$ is built by pushing $\gamma$ (resp. popping $\gamma$ if possible) on top of the stack associated with $o_1$. The element $(o_1, o_2)$ of $\multimap$ means that the stacks associated with $o_1$ and $o_2$ are identical.

Let $[X_1](\pi_1\alpha_1) \rightarrow \ldots [X_2'](\pi_2'\alpha_2') \ldots$ and $[X_2](\pi_2\alpha_2) \rightarrow \Gamma$ be two productions in $P_L^x$ with $[X_2'] = [X_2]$. Moreover, assume that $o_1$, $o_2'$, and $o_2$ respectively denotes the objects $[X_1](\pi_1\alpha_1)$, $[X_2'](\pi_2'\alpha_2')$, and $[X_2](\pi_2\alpha_2)$. The Table 1 indicates precisely the way these relations are defined. All other couples of objects are non comparable.

| $\pi_1$ | $\pi_2'$ | $\pi_2$ | Conditions | Relations |
|---|---|---|---|---|
| any | $\varepsilon$ | $\varepsilon$ | $\alpha_2' = \alpha_2$ | $o_2' \bowtie o_2$ |
| any | $\varepsilon$ | .. | $\alpha_2' = \alpha_2$ | $o_2' \multimap o_2$ |
| any | $\varepsilon$ | .. | $\alpha_2' = \gamma \wedge \alpha_2 = \varepsilon$ | $o_2' \overset{\gamma}{\prec} o_2$ |
| .. | .. | any | $\alpha_2' = \alpha_2$ | $o_1 \multimap o_2$ |
| .. | .. | any | $\alpha_2' = \gamma \wedge \alpha_2 = \varepsilon$ | $o_1 \overset{\gamma}{\prec} o_2$ |
| .. | .. | any | $\alpha_2' = \varepsilon \wedge \alpha_2 = \gamma$ | $o_1 \overset{\gamma}{\succ} o_2$ |

**Table 1: $\overset{\gamma}{\prec}$, $\overset{\gamma}{\succ}$, and $\multimap$ definitions.**

Our algorithm will simply compose the previous relations in order to relate an object where a symbol is pushed to the object(s) where this very symbol is popped in order to finally answer the question: is there at least one valid spine between $o_1$ and $o_2$ where $o_1$ is initial and $o_2$ is final?

Formally the valid spine relation is defined by $\bowtie = \{(o_1, o_2) \mid o_1 \in V_O^I \wedge o_2 \in V_O^F \wedge o_1 \overset{+}{\approx} o_2\}$ where the $\approx$ relation is the smallest solution of the set of recursive equations $\approx = \multimap$ and $\approx = \overset{\gamma}{\prec} \overset{*}{\approx} \overset{\gamma}{\succ}$.

We will implement this computation as a limit of the composition of the $\overset{\gamma}{\prec}$, $\multimap$, and $\overset{\gamma}{\succ}$ relations and we will show that our algorithm has an $\mathcal{O}(n^6)$-time upper bound complexity.

The laws governing this composition are shown in Table 2 where $o_1$ and $o_3$ are any sorts of objects and $o_2$ always designates a primary object[4].

These composition rules are applied until no more new element can be added to any of these relations.

| | | | | | | |
|---|---|---|---|---|---|---|
| $o_1 \multimap o_2$ | and | $o_2 \multimap o_3$ | and | $o_1 \in V_O^I \wedge o_3 \in V_O^F$ | $\Longrightarrow$ | $o_1 \bowtie o_3$ |
| $o_1 \overset{\gamma}{\prec} o_2$ | and | $o_2 \overset{\gamma}{\succ} o_3$ | and | $o_1 \in V_O^I \wedge o_3 \in V_O^F$ | $\Longrightarrow$ | $o_1 \bowtie o_3$ |
| $o_1 \multimap o_2$ | and | $o_2 \multimap o_3$ | and | $o_1 \notin V_O^I \vee o_3 \notin V_O^F$ | $\Longrightarrow$ | $o_1 \multimap o_3$ |
| $o_1 \overset{\gamma}{\prec} o_2$ | and | $o_2 \overset{\gamma}{\succ} o_3$ | and | $o_1 \notin V_O^I \vee o_3 \notin V_O^F$ | $\Longrightarrow$ | $o_1 \multimap o_3$ |
| $o_1 \overset{\gamma}{\prec} o_2$ | and | $o_2 \multimap o_3$ | | | $\Longrightarrow$ | $o_1 \overset{\gamma}{\prec} o_3$ |
| $o_1 \multimap o_2$ | and | $o_2 \overset{\gamma}{\succ} o_3$ | | | $\Longrightarrow$ | $o_1 \overset{\gamma}{\succ} o_3$ |

**Table 2: Valid Composition of relations.**

[4] If unrestricted stack schemas have been used, for example, the composition of $\overset{\alpha_1}{\prec}$ and $\overset{\alpha_2}{\succ}$ would have led to three possibilities, depending upon the stack suffixes $\alpha_1$ and $\alpha_2$, namely $\multimap$ if $\alpha_1 = \alpha_2$, $\overset{\alpha_2'}{\prec}$ if $\alpha_1 = \alpha_2'\alpha_2$, and $\overset{\alpha_1'}{\succ}$ if $\alpha_1'\alpha_1 = \alpha_2$.

If an initial object $o_1$ and a final object $o_2$ are such that $o_1 \bowtie o_2$, this means that there is (at least) one valid spine between these objects. Conversely if there are initial objects with no corresponding final object (in $\bowtie$), or final objects with no initial object, this means that there is no valid spine starting (or ending) at that object and that the productions where these objects occur are invalid w.r.t. the LIG conditions and therefore should be erased. This erasing of productions in the LIGed forest $L^x$ for $x$, creates a new LIG say $\underline{L^x}$.

The string $x$ is an element of the initial LIG $L$ iff the language of the CF-backbone for $\underline{L^x}$ is non empty.

In order to facilitate the evaluation of our algorithm complexity we will add a parameter $k$ to the previous relations $\diamond$, $\overset{\gamma}{\prec}$, and $\overset{\gamma}{\succ}$. The value $k = 1$ is assigned to the initial relations defined in Table 1, and the $k$-relations are achieved by composing 1-relations and $(k-1)$-relations. In fact this value $k$ expresses the existence of sub-strings of length $k+1$ in a-spines.

The procedure in Table 3 implements the definition of the level 1 relations given in Table 1.

```
(1)   procedure 1-relations ()
(2)     V_O^LHS = {o | o → Γ ∈ P_L^x}
(3)     for each o' = [X](π'α') in V_O^LHS do
(4)       for each o → ... [X](π2α2) ... in P_L^x ∪ {S'() → S^x()} do
(5)         if π2 = ε then o = [X](π2α2) end if
(6)         if π2 = ε and π' = ε then I = I ∪ {o}, F = F ∪ {o'}
(7)         else if α2 = α' then ◇=◇ ∪{(o, o')}
                                 1 1
(8)         else if α2 = γ and α' = ε then ≺=≺ ∪{(o, o')}
                                           1 1
(9)         else if α2 = ε and α' = γ then ≻=≻ ∪{(o, o')}
                                           1 1
(10)        end if
(11)      end do
(12)    end do
(13)  end procedure
```

**Table 3: The 1-relations $\overset{\gamma}{\underset{1}{\prec}}$, $\overset{\gamma}{\underset{1}{\succ}}$, and $\underset{1}{\diamond}$.**

Line (2) collects in $V_O^{LHS}$ the LHS objects. The loop at lines (3–12) examines each such LHS object $o'$ which is supposed to be an $X$-object. The embedded loop at lines (4–11) selects the productions with an $X$-object in RHS. Note that we have added a new production $S'() \rightarrow S^x()$ which introduces a new initial object $S^x()$ called the *start* object. This augmented LIG and its start object allow us to handle spines whose initial object non-terminal symbol is the LIG start symbol. The first component of a relation is an LHS object $o$, except when the RHS object $[X](\pi_2\alpha_2)$ is secondary (and therefore initial), this case is processed at line (5). The choice of the relations is governed by the relative values of the stack schemas $(\pi_2\alpha_2)$ and $(\pi'\alpha')$. Instead of building up the $\bowtie$ relation, we choose to build $I$ (resp. $F$) which is the set of valid initial (resp. final) objects. At line (6), when $o$ and $o'$ are secondary ($o$ is initial and $o'$ is final), they are respectively put into $I$ and $F$. Lines (7–9) select the appropriate valid level 1 relation. The case where $\alpha = \gamma$, $\alpha' = \gamma'$, and $\gamma \neq \gamma'$ (push of $\gamma$ immediately followed by a pop of $\gamma'$) is erroneous.

The function in Table 4 describes the way the level $k$ relations are computed from the level 1 and level $k-1$ relations. If a couple $(o_1, o_3)$ is a member of a level $k$ relation, this means that there is at least one string of length $k+1$, starting at $o_1$ and ending at $o_3$, which is a valid sub-string of an a-spine.

The loop body at lines (2–31) is executed twice[5]. Complete valid a-spines may only be reached by composing $\underset{h}{\diamond}$ and $\underset{k-h}{\diamond}$ relations at line (7) or by composing $\overset{\gamma}{\underset{h}{\prec}}$ and $\overset{\gamma}{\underset{k-h}{\succ}}$ relations at line (23). All other

[5]only once when $k = 2$.

valid compositions, as stated in Table 2, participate in the level $k$ relations. A new couple of objects is entered into its level $k$ relation at lines (9, 13, 20, or 25) only if this element is not already a member of the same relation at level $h$ with $h \leq k$. Though this condition can only be seen here as an optimization, it is mandatory when cyclic grammars are considered. This function returns true iff one of its level $k$ relation is not empty (i.e. there is sub-strings of length $k+1$ which are not yet complete valid a-spines).

```
(1)  function k-relations(k) return boolean
(2)    for each h in {1, k − 1} do
(3)      for each (o1, o2) in ◇_h do
(4)        if o2 in V_O^P then
(5)          for each (o2, o3) in ◇_{k−h} do
(6)            if o1 in V_O^S and o3 in V_O^S then
(7)              I = I ∪ {o1}, F = F ∪ {o3}
(8)            else
(9)              ◇_k = ◇_k ∪{(o1, o3)}
(10)           end if
(11)         end do
(12)         for each γ in V_l do
(13)           for each (o2, o3) in ≻^γ_{k−h} do ≻^γ_k = ≻^γ_k ∪{(o1, o3)} end do
(14)         end do
(15)       end if
(16)     end do
(17)     for each γ in V_l do
(18)       for each (o1, o2) in ≺^γ_h do
(19)         if o2 in V_O^P then
(20)           for each (o2, o3) in ◇_{k−h} do ≺^γ_k = ≺^γ_k ∪{(o1, o3)} end do
(21)           for each (o2, o3) in ≻^γ_{k−h} do
(22)             if o1 in V_O^S and o3 in V_O^S then
(23)               I = I ∪ {o1}, F = F ∪ {o3}
(24)             else
(25)               ◇_k = ◇_k ∪{(o1, o3)}
(26)             end if
(27)           end do
(28)         end if
(29)       end do
(30)     end do
(31)   end do
(32)   return ⋃_γ ≺^γ_k ∪ ⋃_γ ≻^γ_k ∪ ◇_k ≠ ∅
(33)  end function
```

**Table 4: The k-relations $\overset{\gamma}{\prec}_k$, $\overset{\gamma}{\succ}_k$, and $\diamond_k$.**

The main function which describes our recognizing algorithm is in Table 5.

Its parameters are a LIG $L$ and an input string $x$. At line (3), $G$ denotes its CF-backbone. The shared parse forest $G^x$ at line (4) is supposed to have been computed by any general CF-parsing algorithm. If $x \notin \mathcal{L}(G)$, it will not be in $\mathcal{L}(L)$ either (line (5)). At line (6), $L^x$ denotes the corresponding LIGed forest. The sets $I$ and $F$, which are going to hold the initial and final valid objects, are initialized to

(1) **function** *recognize* $(L, x)$ **return boolean**
(2) **let** $L = (V_N, V_T, V_I, P_L, S)$
(3) **create** $G = (V_N, V_T, P_G, S)$ /* its CF-backbone */
(4) **create** $G^x = (V_N^x, V_T^x, P_G^x, S^x)$ /* its shared parse forest for $x$ */
(5) **if** $\mathcal{L}(G^x) = \emptyset$ **then return false end if**
(6) **create** $L^x = (V_N^x, V_T^x, V_I, P_L^x, S^x)$ /* its LIGed forest */
(7) $I = F = \emptyset$
(8) $\overset{\gamma}{\underset{1}{\prec}} = \overset{\gamma}{\underset{1}{\succ}} = \underset{1}{\diamond} = \emptyset$
(9) $k = 1$
(10) **call** 1-relations()
(11) **do**
(12) $k = k + 1$
(13) $\overset{\gamma}{\underset{k}{\prec}} = \overset{\gamma}{\underset{k}{\succ}} = \underset{k}{\diamond} = \emptyset$
(14) **while** k-relations($k$)

(15) **if** $S^x()$ **not in** $I$ **then return false end if**
(16) **for each** $r_p() = o_0 \rightarrow \ldots o_h \ldots$ **in** $P_L^x$ **do**
(17) **if** $o_0$ **in** $V_O^S$ **and** $o_0$ **not in** $F$ **or**
(18) $o_h$ **in** $V_O^S$ **and** $o_h$ **not in** $I$ **then**
(19) erase $r_p$ **in** $P_G^x$
(20) **end if**
(21) **end do**
(22) **return** useless-symbol-elimination$(P_G^x) \neq \emptyset$
(23) **end function**

**Table 5: The Recognition Algorithm.**

the empty set at line (7), so are the collection of level 1 relations at line (8). The loop at lines (11–14) computes all the level $k$ relations. The ultimate goal is the computation of sets $I$ and $F$. When the start object $S^x()$ is not an element of $I$, this means that there is no valid spine starting at the root and therefore the recognizer failed (line (15)).

Since $G^x$ is the CF-backbone of $L^x$, each time a production $r_p()$ in $P_L^x$ contains a non valid initial or final object, its corresponding production $r_p$ in $P_G^x$ is erased (see lines (16–21)). At line (22) we assume that a classical algorithm eliminates from $P_G^x$ all the useless symbols[6]. If the resulting production set is not empty, it contains a production of the form $(S, x_{1..n+1}) \rightarrow \ldots$ which shows that $x$ is a sentence of that reduced production set and therefore that $x$ is an element of $L^x$ and hence an element of $L$.

## 5 Its Complexity

Objects in LIGed forest are of the form $(A, x_{i..j})(\pi\alpha)$. The maximum number of split $x_{i..j}$ is $\mathcal{O}(n^2)$ where $|x| = n$. All other parameters (non-terminals and stack schemas) are constant for a given LIG $L$. Therefore, the size of any set which contains objects has an $\mathcal{O}(n^2)$ upper bound, especially $I$, $F$, and $V_O^{LHS}$.

### 5.1 Complexity of the 1-*relations* procedure

In Table 3, we have:

**line (2)** A single pass over $P_L^x$ computes $V_O^{LHS}$, whose size is $\mathcal{O}(n^2)$, in time $\mathcal{O}(n^3)$.

[6] A symbol $X$ is useless if it does not appear in any $S/x$-derivation.

**lines (5–10)** Each activation of this body is performed in constant time.

**lines (4–11)** For a given non-terminal $[X]$ there are at most $\mathcal{O}(n)$ occurrences of $[X]$ in the RHSs of $P_L^{\sharp}$. Therefore, each activation of that block takes $\mathcal{O}(n)$ time.

**lines (3–12)** The body of that loop is executed $\mathcal{O}(n^2)$ time so that block takes $\mathcal{O}(n^3)$ time.

**lines (1–13)** At the end the time complexity of the *1-relations* is $\mathcal{O}(n^3)$.

Since the body part (lines (7–9)) where the 1-relations are computed is executed at most $\mathcal{O}(n^3)$ time, the size of these relations is $\mathcal{O}(n^3)$. We notice that in each such relation, for a given object, say $o$, there are at most $\mathcal{O}(n)$ pairs whose first member or second member is $o$.

### 5.2 Complexity of the *k-relations* function

When $k > 1$, the size of the level $k$ relations, since they contain pair of objects is at most $\mathcal{O}(n^4)$.

In Table 4, we have:

**lines (5–11), (13), (20), (21–27)** For each intermediate primary object $o_2$, these loops are executed a number of time which depends on the value of $h$ since we refer to either level 1 relations or level $k-1$ relations. In the case where $k - h > 1$, these loops are executed $\mathcal{O}(n^2)$ time, else, when $k - h = 1$, these loops are executed $\mathcal{O}(n)$ time. Since their body sets individual relations in constant time, the overall complexity is not changed.

**lines (12–14)** Since line (13) is executed a bounded (i.e $|V_l|$) number of times, the complexity of the body extends to this loop.

**lines (3–16), (18–29)** For each activation of these loops, their body is executed $\mathcal{O}(n^3)$ time when $h = 1$ or $\mathcal{O}(n^4)$ when $h > 1$. We see that, in all cases, we get an execution time of $\mathcal{O}(n^5)$ for each activation.

**lines (17–30)** The complexity of its body extends to this loop (i.e. $\mathcal{O}(n^5)$).

**lines (2–33)** This loop is executed at most twice (when $k > 2$), therefore this block takes $\mathcal{O}(n^5)$.

**line (34)** This return condition may easily be get as a side effect of the setting of the relations (is there at least one element?), and therefore does not change the overall complexity.

In the worst case, the time complexity of the *k-relations* function is $\mathcal{O}(n^5)$.

### 5.3 Complexity of the Recognition Algorithm

In Table 5, we have:

**line (4)** Can take $\mathcal{O}(n^3)$ with the appropriate CF-parsing algorithm since the length of the longest RHS is two.

**line (6)** The LIGed forest is almost simply a copy of the shared parse forest and therefore takes $\mathcal{O}(n^3)$.

**line (10)** Takes $\mathcal{O}(n^3)$ (see 5.1).

**lines (11–14)** In order to evaluate the complexity of that loop we should know the maximum value of $k$. Recall that $k + 1$ is the length of valid sub-strings and therefore its maximum value corresponds to the length of the longest a-spine. Since spines are specialized path in parse trees and the height of parse trees (for non cyclic grammar) is $\mathcal{O}(n)$, this loop is executed $\mathcal{O}(n)$ times and since each execution of the *k-relations* function takes $\mathcal{O}(n^5)$ (see 5.2), this loop takes at most $\mathcal{O}(n^6)$.

**lines (16–21)** Takes $\mathcal{O}(n^3)$.

**line (23)** The classical algorithm for the elimination of useless symbol is performed in time linear with the size of the grammar, so in our case it will take $\mathcal{O}(n^3)$.

So, in the worst case, for a non cyclic grammar, the time complexity of our recognition algorithm is $\mathcal{O}(n^6)$.

If the CF-backbone of a LIG is unambiguous, the shared parse forest can be built in time $\mathcal{O}(n^2)$ by an Earley or generalized LR parsing algorithm (see [5]). In such a case, the shared parse forest is a simple (parse) tree whose size is $\mathcal{O}(n)$. Therefore, the objects cannot be shared among spines, and the cumulated length of all the spines is $\mathcal{O}(n)$. With this hypothesis, the size of our $k$-relations for $k \geq 1$ is $\mathcal{O}(n)$ and it can easily be seen that, for a given value $k$, their construction takes $\mathcal{O}(n)$ time, and that the complete check could therefore be performed in $\mathcal{O}(n^2)$ time[7]. Therefore, for unambiguous grammars, a total recognition time of $\mathcal{O}(n^2)$ is reached by our algorithm.

We can wonder whether intermediate values between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^6)$ are reached for some sub-classes of LIGs. When the number of non-terminal symbols is $\mathcal{O}(n)$ in a shared parse forest it is not difficult to see that our recognizer has an $\mathcal{O}(n^4)$ worst time bound, but unfortunately we are not aware of any grammatical characterization of such a sub-class!

It should be pointed out that our algorithm is valid, even without restricting the maximum length $l$ of the RHSs. The only consequence is that the recognizing time can be increased since the CF-parsing time (and the size of the shared parse forest) can be of the order $\mathcal{O}(n^{l+1})$. Moreover, though the cardinalities of the $k$-relations with $k \geq 2$ stay in $\mathcal{O}(n^4)$, the cardinalities of the 1-relations increase to $\mathcal{O}(n^4)$ and therefore induce a checking of the LIG conditions in time $\mathcal{O}(n^7)$. Finally, without restriction, a fair LIG can be recognized by our algorithm in time $\max(\mathcal{O}(n^{l+1}), \mathcal{O}(n^7))$.

## 6 An Example

In this section, we illustrate our algorithm with a LIG $L = (\{S, T\}, \{a, b, c\}, \{\gamma_a, \gamma_b, \gamma_c\}, P_L, S)$ where $P_L$ contains the following productions:

$$\begin{array}{llll} S(..) \rightarrow S(..\gamma_a)a & S(..) \rightarrow S(..\gamma_b)b & S(..) \rightarrow S(..\gamma_c)c & S(..) \rightarrow T(..) \\ T(..\gamma_a) \rightarrow aT(..) & T(..\gamma_b) \rightarrow bT(..) & T(..\gamma_c) \rightarrow cT(..) & T() \rightarrow c \end{array}$$

It is easy to see that its CF-backbone $G$, whose production set $P_G$ is:

$$\begin{array}{llll} S \rightarrow Sa & S \rightarrow Sb & S \rightarrow Sc & S \rightarrow T \\ T \rightarrow aT & T \rightarrow bT & T \rightarrow cT & T \rightarrow c \end{array}$$

defines the language $\mathcal{L}(G) = \{wcw' \mid w, w' \in \{a, b, c\}^*\}$. We remark that the stacks of symbols in $L$ constrain the string $w'$ to be equal to $w$ and therefore the language $\mathcal{L}(L)$ is $\{wcw \mid w \in \{a, b, c\}^*\}$.

We can remark that in $L$ the key part is played by the middle $c$, introduced by the last production $T() \rightarrow c$, and that this grammar is non ambiguous, while in $G$ the symbol $c$, introduced by the last production $T \rightarrow c$, is only a separator between $w$ and $w'$ and that this grammar is ambiguous (any occurrence of $c$ may be this separator).

Let $x = ccc$ be an input string, we wish to know whether $x$ is an element of $\mathcal{L}(L)$.

Since $x$ is an element of $\mathcal{L}(G)$, its shared parse forest $G^x$ is not empty. Its production set $P_G^x$ is:

$$\begin{array}{ll} (S, x_{1..4}) \rightarrow (S, x_{1..3})x_{3..4} & (S, x_{1..4}) \rightarrow (T, x_{1..4}) \\ (S, x_{1..3}) \rightarrow (S, x_{1..2})x_{2..3} & (S, x_{1..3}) \rightarrow (T, x_{1..3}) \\ (S, x_{1..2}) \rightarrow (T, x_{1..2}) & (T, x_{1..4}) \rightarrow x_{1..2}(T, x_{2..4}) \\ (T, x_{2..4}) \rightarrow x_{2..3}(T, x_{3..4}) & (T, x_{3..4}) \rightarrow x_{3..4} \\ (T, x_{1..3}) \rightarrow x_{1..2}(T, x_{2..3}) & (T, x_{2..3}) \rightarrow x_{2..3} \\ (T, x_{1..2}) \rightarrow x_{1..2} & \end{array}$$

We can observe that this shared parse forest denotes in fact three different parse trees. Each one corresponding to a different cutting out of $x = wcw'$ (i.e. $w = \varepsilon$ and $w' = cc$, or $w = c$ and $w' = c$, or $w = cc$ and $w' = \varepsilon$).

The corresponding LIGed forest whose start symbol is $S^x = (S, x_{1..4})$ and production set $P_L^x$ is:

[7] Remark that an obvious algorithm which evaluates stacks on this single parse tree will take $\mathcal{O}(n)$.

$(S, x_{1..4})(..) \rightarrow (S, x_{1..3})(..\gamma_c)x_{3..4}$ $(S, x_{1..4})(..) \rightarrow (T, x_{1..4})(..)$
$(S, x_{1..3})(..) \rightarrow (S, x_{1..2})(..\gamma_c)x_{2..3}$ $(S, x_{1..3})(..) \rightarrow (T, x_{1..3})(..)$
$(S, x_{1..2})(..) \rightarrow (T, x_{1..2})(..)$ $(T, x_{1..4})(..\gamma_c) \rightarrow x_{1..2}(T, x_{2..4})(..)$
$(T, x_{2..4})(..\gamma_c) \rightarrow x_{2..3}(T, x_{3..4})(..)$ $(T, x_{3..4})() \rightarrow x_{3..4}$
$(T, x_{1..3})(..\gamma_c) \rightarrow x_{1..2}(T, x_{2..3})(..)$ $(T, x_{2..3})() \rightarrow x_{2..3}$
$(T, x_{1..2})() \rightarrow x_{1..2}$

In this LIGed forest there are three a-spines which are shown below with their objects separated by the appropriate level 1 relation:

$$(S, x_{1..4})() \underset{1}{\diamond\!\!-} (S, x_{1..4})(..) \overset{\gamma_c}{\underset{1}{\prec}} (S, x_{1..3})(..) \overset{\gamma_c}{\underset{1}{\prec}} (S, x_{1..2})(..) \underset{1}{\diamond\!\!-} (T, x_{1..2})()$$
$$(S, x_{1..4})() \underset{1}{\diamond\!\!-} (S, x_{1..4})(..) \overset{\gamma_c}{\underset{1}{\prec}} (S, x_{1..3})(..) \overset{\gamma_c}{\underset{1}{\succ}} (T, x_{1..3})(..\gamma_c) \underset{1}{\diamond\!\!-} (T, x_{2..3})()$$
$$(S, x_{1..4})() \underset{1}{\diamond\!\!-} (S, x_{1..4})(..) \overset{\gamma_c}{\underset{1}{\succ}} (T, x_{1..4})(..\gamma_c) \overset{\gamma_c}{\underset{1}{\succ}} (T, x_{2..4})(..\gamma_c) \underset{1}{\diamond\!\!-} (T, x_{3..4})()$$

Though these a-spines are not computed by our algorithm, it is easier to see what happens directly on them. In particular we can see that the first and last a-spine are not valid since there is $\overset{\gamma_c}{\prec}$ (or $\overset{\gamma_c}{\succ}$) without corresponding $\overset{\gamma_c}{\succ}$ (or $\overset{\gamma_c}{\prec}$) and that only the middle a-spine is valid. In fact the algorithm computes the level 1 relations (shown within the a-spines), the level 2 relations:

$$(S, x_{1..3})(..) \overset{\gamma_c}{\underset{2}{\prec}} (T, x_{1..2})() \quad (S, x_{1..4})(..) \underset{2}{\diamond\!\!-} (T, x_{1..3})(..\gamma_c) \quad (S, x_{1..4})() \overset{\gamma_c}{\underset{2}{\succ}} (T, x_{1..4})(..\gamma_c)$$

the level 3 relations:

$$(S, x_{1..4})() \underset{3}{\diamond\!\!-} (T, x_{1..3})(..\gamma_c) \quad (S, x_{1..4})(..) \underset{3}{\diamond\!\!-} (T, x_{2..3})()$$

while the computing of the level 4 leads to empty relations with sets $I = \{(S, x_{1..4})()\}$ and $F = \{(T, x_{2..3})()\}$.

Since the start object $(S, x_{1..4})()$ is in $I$, the execution of lines (16–21) in Table 5 leads to erase the productions $(T, x_{1..2}) \rightarrow x_{1..2}$, and $(T, x_{3..4}) \rightarrow x_{3..4}$ in $P_G^x$. The *useless-symbol-elimination* function called at line (22) returns the following (non empty) production set:

$(S, x_{1..4}) \rightarrow (S, x_{1..3})x_{3..4}$ $(S, x_{1..3}) \rightarrow (T, x_{1..3})$
$(T, x_{1..3}) \rightarrow x_{1..2}(T, x_{2..3})$ $(T, x_{2..3}) \rightarrow x_{2..3}$

which shows that $ccc \in \mathcal{L}(L)$.

We can remark that, with that example, our recognition algorithm is in fact a parsing algorithm (i.e. all resulting a-spines are valid). This is not always the case. Assume a LIGed forest with the following four a-spines: $s_1 = (o_1, \ldots, o_3)$, $s_2 = (o_1, \ldots, o_4)$, $s_3 = (o_2, \ldots, o_3)$, and $s_4 = (o_2, \ldots, o_4)$. Moreover assume that the only valid a-spines are $s_1$ and $s_4$, therefore, the algorithm will consider that $o_1$ and $o_2$ are valid initial objects and that $o_3$ and $o_4$ are valid final objects and that no production elimination should take place. Therefore, the LIGed forest is left unchanged but could not be considered as a representation of the shared parse forest for the initial LIG since there are a-spines $s_2$ and $s_3$ which are not valid.

## 7 Conclusion

In this paper we have presented a new recognition algorithm which works for the class of mildly context-sensitive languages. Though its worst case complexity does not improve over previous ones (i.e. a $\mathcal{O}(n^6)$ time is achieved), the recognizer behaves in practice much faster than its worst case.

The advantages of this algorithm can mainly be summarized as follows:

- parsing of the input string with the underlying CFG and checking of the LIG conditions are split into separate phases;

- LIG conditions checking relies upon a very simple principle which can be expressed by binary relations;
- the recognition test is simply performed by composition of the previous relations;
- therefore, no symbol stack computation is needed;
- it can be applied to unrestricted fair LIGs (though the $\mathcal{O}(n^6)$ limit can then be exceeded).

We can wonder whether the first point is really an advantage since it can be retorted that illegal paths should be aborted as soon as possible. Our argument is that it wastes time to compute symbol stacks in $\mathcal{O}(n^6)$ along paths which can be discovered as syntactically illegal in $\mathcal{O}(n^3)$.

This algorithm is implemented in a prototype system which is part of an ongoing effort to get a set of parsers for various NL formalisms.

## References


[1] ABEILLÉ, A., and SCHABES, Y. 1989. Parsing idioms in lexicalized TAGs. *Proceedings of the fourth conference of the ACL.*

[2] AHO, A. V. 1968. Indexed grammars—An extension to context free grammars. *J. ACM*, Vol. 15, pp. 647-671.

[3] EARLEY, Jay C. 1968. An efficient context-free parsing algorithm. *Ph.D. thesis*, Carnegie-Mellon University, Pittsburgh, PA.

[4] KASAMI, T. 1965. An efficient recognition and syntax algorithm for context-free languages. *Technical Report AF-CRL-65-758*, Air Force Cambridge Research Laboratory, Bedford, MA.

[5] KIPPS, J. R. 1989. Analysis of Tomita's algorithm for general context-free parsing. *International Parsing Workshop'89*, pp. 193-202.

[6] KILGER, A., and FINKLER, W. 1993. TAG-based incremental generation. *German Research Center for Artificial Intelligence (DFKI), Technical Report*, Saarbrücken (Germany).

[7] LANG, B. 1974. Deterministic techniques for efficient non-deterministic parsers. *Automata, Languages and Programming, 2nd Colloquium*, Lectures Notes in Computer Science, Springer-Verlag, Vol. 14, pp. 255-269.

[8] LANG, B. 1991. Towards a uniform formal framework for parsing. *Current Issues in Parsing Technology*, edited by M. Tomita, Kluwer Academic Publishers, pp. 153-171.

[9] LANG, B. 1994. Recognition can be harder than parsing. *Computational Intelligence*, Vol. 10, No. 4, pp. 486-494.

[10] PAROUBEK, P., SCHABES, Y., and JOSHI, A. K. 1992. XTAG-a graphical workbench for developing tree-adjoining grammars. *Third Conference on Applied Natural Language Processing*, Trento (Italy).

[11] POLLER, P. 1994. Incremental parsing with LD/TLP-TAGs. *Computational Intelligence*, Vol. 10, No. 4, pp. 549-562.

[12] REKERS, J. 1992. Parser generation for interactive environments. *Ph.D. thesis*, University of Amsterdam.

[13] SCHABES, Y. 1994. Left to right parsing of lexicalized tree-adjoining grammars. *Computational Intelligence*, Vol. 10, No. 4, pp. 506-524.

[14] TOMITA, M. 1987. An efficient augmented context-free parsing algorithm. *Computational Linguistics*, Vol. 13, pp. 31-46.

[15] VIJAY-SHANKER, K. 1987. A study of tree adjoining grammars. *PhD. thesis*, University of Pennsylvania.

[16] VIJAY-SHANKER, K., and JOSHI, A. K. 1985. Some computational properties of tree adjoining grammars. *23rd Meeting of the Association for Computational Linguistics*, Chicago, pp. 82-93.

[17] VIJAY-SHANKER, K., and WEIR D. J. 1994. Parsing some constrained grammar formalisms. *ACL Computational Linguistics*, Vol. 19, No. 4, pp. 591-636.

[18] YOUNGER, D. H. 1965. Recognition and parsing of context-free languages in time $n^3$. *Information and Control*, Vol. 10, No. 2, pp. 189-208.

# DEVELOPING AND EVALUATING A PROBABILISTIC LR PARSER OF PART-OF-SPEECH AND PUNCTUATION LABELS*

Ted Briscoe

John Carroll

Computer Laboratory, University of Cambridge
Pembroke Street, Cambridge CB2 3QG, UK
ejb / jac@cl.cam.ac.uk

We describe an approach to robust domain-independent syntactic parsing of unrestricted naturally-occurring (English) input. The technique involves parsing sequences of part-of-speech and punctuation labels using a unification-based grammar coupled with a probabilistic LR parser. We describe the coverage of several corpora using this grammar and report the results of a parsing experiment using probabilities derived from bracketed training data. We report the first substantial experiments to assess the contribution of punctuation to deriving an accurate syntactic analysis, by parsing identical texts both with and without naturally-occurring punctuation marks.

## 1 Introduction

This work is part of an effort to develop a robust, domain-independent syntactic parser capable of yielding the one correct analysis for unrestricted naturally-occurring input. Our goal is to develop a system with performance comparable to extant part-of-speech taggers, returning a syntactic analysis from which predicate-argument structure can be recovered, and which can support semantic interpretation. The requirement for a domain-independent analyser favours statistical techniques to resolve ambiguities, whilst the latter goal favours a more sophisticated grammatical formalism than is typical in statistical approaches to robust analysis of corpus material.

Briscoe and Carroll (1993) describe a probablistic parser using a wide-coverage unification-based grammar of English written in the Alvey Natural Language Tools (ANLT) metagrammatical formalism (Briscoe *et al.*, 1987), generating around 800 rules in a syntactic variant of the Definite Clause Grammar formalism (DCG, Pereira & Warren, 1980) extended with iterative (Kleene) operators. The ANLT grammar is linked to a lexicon containing about 64K entries for 40K lexemes, including detailed subcategorisation information appropriate for the grammar, built semi-automatically from a learners' dictionary (Carroll & Grover, 1989). The resulting parser is efficient, capable of constructing a parse forest in what seems to be roughly quadratic time, and efficiently returning the ranked $n$-most likely analyses (Carroll, 1993, 1994). The probabilistic model is a refinement of probabilistic context-free grammar (PCFG) conditioning CF 'backbone' rule application on LR state and lookahead item. Unification of the 'residue'

*Some of this work was carried out while the first author was visting Rank Xerox, Grenoble. The work was also supported by DTI/SALT project 41/5808 'Integrated Language Database'. Geoff Nunberg provided encouragement and much advice on the analysis of punctuation, and Greg Grefenstette undertook the original tokenisation and segmentation of Susanne. Bernie Jones and Kiku Ribas made helpful comments on an earlier draft. We are responsible for any mistakes.

of features not incorporated into the backbone is performed at parse time in conjunction with reduce operations. Unification failure results in the associated derivation being assigned a probability of zero. Probabilities are assigned to transitions in the LALR(1) action table via a process of supervised training based on computing the frequency with which transitions are traversed in a corpus of parse histories. The result is a probabilistic parser which, unlike a PCFG, is capable of probabilistically discriminating derivations which differ only in terms of order of application of the same set of CF backbone rules, due to the parse context defined by the LR table.

Experiments with this system revealed three major problems which our current research is addressing. Firstly, although the system is able to rank parses with a 75% chance that the correct analysis will be the most highly ranked, further improvement will require a 'lexicalised' system in which (minimally) probabilities are associated with alternative subcategorisation possibilities of individual lexical items. Currently, the relative frequency of subcategorisation possibilities for individual lexical items is not recorded in wide-coverage lexicons, such as ANLT or COMLEX (Grishman *et al.*, 1994). Secondly, removal of punctuation from the input (after segmentation into text sentences) worsens performance as punctuation both reduces syntactic ambiguity (Jones, 1994) and signals non-syntactic (discourse) relations between text units (Nunberg, 1990). Thirdly, the largest source of error on unseen input is the omission of appropriate subcategorisation values for lexical items (mostly verbs), preventing the system from finding the correct analysis. The current coverage of this system on a general corpus (e.g. Brown or LOB) is estimated to be around 20% by Briscoe (1994). We have developed a variant probabilistic LR parser which does not rely on subcategorisation and uses punctuation to reduce ambiguity. The analyses produced by this parser could be utilised for phrase-finding applications, recovery of subcategorisation frames, and other 'intermediate' level parsing problems.

## 2 Part-of-speech Tag Sequence Grammar

Several robust parsing systems exploit the comparative success of part-of-speech (PoS) taggers, such as Fidditch (Hindle, 1989) or MITFP (de Marcken, 1990), by reducing the input to a determinate sequence of extended PoS labels of the type which can be practically disambiguated in context using a (H)MM PoS tagger (e.g. Church, 1988). Such approaches, by definition, cannot exploit subcategorisation, and probably achieve some of their robustness as a result. However, such parsers typically also employ heuristic rules, such as 'low' attachment of PPs to produce unique 'canonical' analyses. This latter step complicates the recovery of predicate-argument structure and does not integrate with a probabilistic approach to parsing.

We utilised the ANLT metagrammatical formalism to develop a feature-based, declarative description of PoS label sequences for English. This grammar compiles into a DCG-like grammar of approximately 400 rules. It has been designed to enumerate possible valencies for predicates (verbs, adjectives and nouns) by including separate rules for each pattern of possible complementation in English. The distinction between arguments and adjuncts is expressed, following X-bar theory (e.g. Jackendoff, 1977), by Chomsky-adjunction of adjuncts to maximal projections (XP $\rightarrow$ XP Adjunct) as opposed to government of arguments (i.e. arguments are sisters within X1 projections; X1 $\rightarrow$ X0 Arg1... ArgN). Although the grammar enumerates complementation possibilities and checks for global sentential well-formedness, it is best described as 'intermediate' as it does not attempt to associate 'displaced' constituents with their canonical position / grammatical role.

The other difference between this grammar and a more conventional one is that it incorporates some rules specifically designed to overcome limitations or idiosyncrasies of the PoS tagging process. For example, past particles functioning adjectivally, as in *The disembodied head*, are frequently tagged as past participles (VVN) i.e. *The_AT disembodied_VVN head_NN1*, so the grammar incorporates a rule which parses past participles as adjectival premodifiers in this context. Similar idiosyncratic rules are incorporated for dealing with gerunds, adjective-noun conversions, idiom sequences, and so forth.

This grammar was developed and refined in a corpus-based fashion (e.g. see Black, 1993) by testing against sentences from the Susanne corpus (Sampson, 1994), a 138K word treebanked and balanced subset of the Brown corpus[1].

## 3 Text Grammar and Punctuation

Nunberg (1990) develops a partial 'text' grammar for English which incorporates many constraints that (ultimately) restrict syntactic and semantic interpretation. For example, textual adjunct clauses introduced by colons scope over following punctuation, as (1a) illustrates; whilst textual adjuncts introduced by dashes cannot intervene between a bracketed adjunct and the textual unit to which it attaches, as in (1b).

(1) a *He told them his reason: he would not renegotiate his contract, but he did not explain to the team owners. (vs. but would stay)
b *She left – who could blame her – (during the chainsaw scene) and went home.

We have developed a declarative grammar in the ANLT metagrammatical formalism, based on Nunberg's procedural description. This grammar captures the bulk of the text-sentential constraints described by Nunberg with a grammar which compiles into 26 DCG-like rules. Text grammar analyses are useful because they demarcate some of the syntactic boundaries in the text sentence and thus reduce ambiguity, and because they identify the units for which a syntactic analysis should, in principle, be found; for example, in (2), the absence of dashes would mislead a parser into seeking a syntactic relationship between *three* and the following names, whilst in fact there is only a discourse relation of elaboration between this text adjunct and pronominal *three*.

(2) The three – Miles J. Cooperman, Sheldon Teller, and Richard Austin – and eight other defendants were charged in six indictments with conspiracy to violate federal narcotic law.

The rules of the text grammar divide into three groups: those introducing text-sentences, those defining text adjunct introduction and those defining text adjuncts (Nunberg, 1990). An example of each type of rule is given in (3a–c).

(3) a T/txt-scl : TxtS → (Tu[+sc])* Tu[-sc] (+pex|+pqu)
b Ta/dash- : Tu[-sc] → T[-sc, -cl, -da] Ta[+da, da-]
c T/t_ta-da-_t : Ta[+da, da-] → +pda Tu[-sc, -da]

[1]The grammar currently covers more than 75% of the sentences. Many of the remaining failures for shorter text sentences are a consequence of the root S node requirement, since they represent elliptical noun or prepositional phrases in dialogue. Other failures on sentences are a consequence of incorporation of complementation constraints for auxiliary verbs into the grammar but the lack of any treatment of unbounded dependencies. Nevertheless, we tolerate these 'deficiencies', since they have the effect of limiting the number of analyses recovered in other cases, and will not, for example, affect unduly the recovery of subcategorisation frames from the resulting analyses.

These rules are phrase structure schemata employing iterative operators, optionality and disjunction, preceded by a mnemonic name. Non-terminal categories are text sentences, units or adjuncts which carry features mostly representing the punctuation marks which occur as daughters in the rules (e.g. +sc represents presence of a semi-colon marker), whilst terminal punctuation is represented as +pxx (e.g. +pda, dash). (3a) states that a text sentence can contain zero or more text units with a semi-colon at their right boundary followed by a text unit optionally followed by a question or exclamation mark. (3b) states that a text unit not containing a semi-colon can consist of a text unit or adjunct not containing dashes, colons or semi-colons followed by a text adjunct introduced by a dash. This type of 'unbalanced' adjunct can only be expanded by (3c) which states that it consists of a single opening dash followed by a text unit which does not itself contain dashes or semi-colons. The features on the first daughter of (3b) force dash adjuncts to have lower precedence and narrower scope than colons or semi-colons, blocking interpretations of multiple dashes as sequences of 'unbalanced' adjuncts.

Nunberg (1990) invokes rules of (point) absorption which delete punctuation marks (inserted according to a simple context-free text grammar) when adjacent to other 'stronger' punctuation marks. For instance, he treats all dash interpolated text adjuncts as underlyingly balanced, but allows a rule of point absorption to convert (4a) into (4b).

(4) a *Max fell – John had kicked him –.
b Max fell – John had kicked him.

The various rules of absorption introduce procedurality into the grammatical framework and require the positing of underlying forms which are not attested in text. For this reason, 'absorption' effects are captured through propagation of featural constraints in parse trees. For instance, (4a) is blocked by including distinct rules for the introduction of balanced and unbalanced text adjuncts and only licensing the latter text sentence finally.

The text grammar has been tested on Susanne and covers 99.8% of sentences. (The failures are mostly text segmentation problems). The number of analyses varies from one (71%) to the thousands (0.1%). Just over 50% of Susanne sentences contain some punctuation, so around 20% of the singleton parses are punctuated. The major source of ambiguity in the analysis of punctuation concerns the function of commas and their relative scope as a result of a decision to distinguish delimiters and separators (Nunberg 1990:36). Therefore, a text sentence containing eight commas (and no other punctuation) will have 3170 analyses. The multiple uses of commas cannot be resolved without access to (at least) the syntactic context of occurrence.

## 4 The Integrated Grammar

Despite Nunberg's observation that text grammar is distinct from syntax, text grammatical ambiguity favours interleaved application of text grammatical and syntactic constraints. The integration of text and PoS sequence grammars is straightforward and remains modular, in that the text grammar is 'folded into' the PoS sequence grammar, by treating text and syntactic categories as overlapping and dealing with the properties of each using disjoint sets of features, principles of feature propagation, and so forth. The text grammar rules are represented as left or right branching rules of 'Chomsky-adjunction' to lexical or phrasal constituents. For example, the simplified rule for combining NP appositional or parenthetical text adjuncts is N2[+ta] → H2 Ta[+bal] which states that a NP containing a textual adjunct consists of a head NP followed by a textual adjunct with balanced delimiters (dashes, brackets or commas). Rules

of this form ensure that syntactic and textual analysis are mutually 'transparent' and orthogonal so, for example, any rules of semantic interpretation associated with syntactic rules continue to function unmodified. Such rules attach text adjuncts to the constituents over which they semantically scope, so it would be possible, in principle, to develop a semantics for them. In addition to the core text grammatical rules which carry over unchanged from the stand-alone text grammar, 44 syntactic rules (of pre- and post- posing, and coordination) now include (often optional) comma markers corresponding to the purely 'syntactic' uses of punctuation.

The approach to text grammar taken here is in many ways similar to that of Jones (1994). However, he opts to treat punctuation marks as clitics on words which introduce additional featural information into standard syntactic rules. Thus, his grammar is thoroughly integrated and it would be harder to extract an independent text grammar or build a modular semantics. The coverage of the integrated version of the text grammar is described in more detail in Briscoe & Carroll (1994).

## 5 Parsing the Susanne and SEC Corpora

The integrated grammar has been used to parse Susanne and the quite distinct SEC Corpus (Taylor & Knowles, 1988), a 50K word treebanked corpus of transcribed British radio programmes punctuated by the corpus compilers. Both corpora were retagged with determinate punctuation and PoS labelling using the Acquilex HMM tagger (Elworthy, 1993, 1994) trained on text tagged with a slightly modified version of CLAWS-II labels (Garside *et al.*, 1987).

### 5.1 Coverage and Average Ambiguity

To examine the efficiency and coverage of the grammar we applied it to our retagged versions of Susanne and SEC. We used the ANLT chart parser (Carroll, 1993), but modified just to count the number of possible parses in the parse forests (Billot & Lang, 1989) rather than actually unpacking them. We also imposed a per-sentence time-out of 30 seconds CPU time, running in Franz Allegro Common Lisp 4.2 on an HP PA-RISC 715/100 workstation with 96 Mbytes of physical memory.

We define the 'coverage' of the grammar to be the inverse of the proportion of sentences for which no analysis was found—a weak measure since discovery of one or more global analyses does not entail that the correct analysis is recovered. For both corpora, the majority of sentences analysed successfully received under 100 parses, although there is a long tail in the distribution. Monitoring this distribution is helpful during grammar development to ensure that coverage is increasing but the ambiguity rate is not. A more succinct though less intuitive measure of ambiguity rate for a given corpus is what we call the average parse base (APB), defined as the geometric mean over all sentences in the corpus of $\sqrt[n]{p}$, where $n$ is the number of words in a sentence, and $p$, the number of parses for that sentence[2]. Thus, given a sentence $n$ tokens long, the APB raised to the $n$th power gives the number of analyses that the grammar can be expected to assigned to a sentence of that length in the corpus. Table 1 gives these measures for all of the sentences in Susanne and in SEC.

As the grammar was developed solely with reference to Susanne, coverage of SEC is quite robust. The two corpora differ considerably since the former is drawn from American written

[2]Black *et al.*(1993:13) define an apparently similar measure, *parse base*, as the "geometric mean of the number of parses per word for the entire corpus", but in the immediately following sentence talk about raising it to the power of the number of words in a sentence, which is inappropriate for a simple ratio.

| | Susanne | | SEC | |
|---|---|---|---|---|
| Parse fails | 1745 | 24.9% | 898 | 33.1% |
| 1–9 parses | 1566 | 22.3% | 607 | 22.3% |
| 10–99 parses | 1306 | 18.6% | 418 | 15.4% |
| 100–999 parses | 893 | 12.7% | 299 | 11.0% |
| 1K–9.9K parses | 611 | 8.7% | 197 | 7.3% |
| 10K–99K parses | 413 | 5.9% | 108 | 4.0% |
| 100K+ parses | 475 | 6.8% | 189 | 7.0% |
| Time-outs | 5 | 0.07% | 1 | 0.04% |
| Number of sentences | 7014 | | 2717 | |
| Mean sentence length (MSL) | 20.1 | | 22.6 | |
| MSL – fails | 21.7 | | 27.6 | |
| MSL – time-outs | 67.2 | | 79.0 | |
| Average Parse Base | 1.256 | | 1.239 | |

Table 1: Grammar coverage on Susanne and SEC

text whilst the latter represents British transcribed spoken material. The corpora overall contain material drawn from widely disparate genres / registers, and are more complex than those used in DARPA ATIS tests and more diverse than those used in MUC. The APBs for Susanne and SEC of 1.256 and 1.239 respectively indicate that sentences of average length in each corpus could be expected to be assigned of the order of 97 and 126 analyses (i.e. $1.256^{20.1}$ and $1.239^{22.6}$). Black *et al.*(1993:156) quote a *parse base* of 1.35 for the IBM grammar for computer manuals applied to sentences 1–17 words long. Although, as mentioned above, Black's measure may not be exactly the same as our APB measure, it is probable that the IBM grammar assigns more analyses than ours for sentences of the same length. Black achieves a coverage of around 95%, as opposed to our coverage rate of 67–74% on much more heterogeneous data and longer sentences.

The parser throughput on these tests, for sentences successfully analysed, is around 45 words per CPU second on an HP PA-RISC 715/100. Sentences of up to 30 tokens (words plus punctuation) are parsed in an average under 0.6 seconds each, whilst those around 60 tokens take on average 4.5 seconds. Nevertheless, the relationship between sentence length and processing time is fitted well by a quadratic function, supporting the findings of Carroll (1994) that in practice NL grammars do not evince worst-case parsing complexity.

### Coverage, Ambiguity and Punctuation

We have also run experiments to evaluate the degree to which punctuation is contributing useful information. Intuitively, we would expect the exploitation of text grammatical constraints to both reduce ambiguity and extend coverage (where punctuation cues discourse rather than syntactic relations between constituents). Jones (1994) reports a preliminary experiment evaluating reduction of ambiguity by punctuation. However, the grammar he uses was developed only to cover the test sentences, drawn entirely from the SEC corpus which was punctuated post hoc by the corpus developers (Taylor and Knowles, 1988).

We took all in-coverage sentences from Susanne of length 8–40 words inclusive containing internal punctuation; a total of 2449 sentences. The APB for this set was 1.273, mean length 22.5 words, giving an expected number of analyses for an average sentence of 225. We then re-

moved all sentence-internal punctuation from this set and re-parsed it. Around 8% of sentences now failed to receive an analysis. For those that did (mean length 20.7 words), the APB was now 1.320, so an average sentence would be assigned 310 analyses, 38% more than before. On closer inspection, the increase in ambiguity is due to two factors: a) a significant proportion of sentences that previously received 1–9 analyses now receive more, and b) there is a much more substantial tail in the distribution of sentence length vs. number of parses, due to some longer sentences being assigned many more parses. Manual examination of 100 depunctuated examples revealed that in around a third of cases, although the system returned global analyses, the correct one was not in this set (Briscoe & Carroll, 1994). With a more constrained (sub-categorised) syntactic grammar, many of these examples would not have received any global syntactic analysis.

### 5.2 Parse Selection

A probabilistic LR parser was trained with the integrated grammar by exploiting the Susanne treebank bracketing. An LR parser (Briscoe and Carroll, 1993) was applied to unlabelled bracketed sentences from the Susanne treebank, and a new treebank of 1758 correct and complete analyses with respect to the integrated grammar was constructed semi-automatically by manually resolving the remaining ambiguities. 250 sentences from the new treebank were kept back for testing. The remainder, together with a further set of analyses from 2285 treebank sentences that were not checked manually, were used to train a probabilistic version of the LR parser, using Good-Turing smoothing to estimate the probability of unseen transitions in the LALR(1) table (Briscoe and Carroll, 1993; Carroll, 1993). The probabilistic parser can then return a ranking of all possible analyses for a sentence, or efficiently return just the $n$-most probable (Carroll, 1993).

The probabilistic parser was tested on the 250 sentences held out from the manually-created treebank (with mean length 18.2 tokens, mean number of parses per sentence 977, and APB 1.252); in this test 85 sentences (34%) had the correct analysis ranked in the top three[3]. This figure rose to 51% for sentences of less than 20 words. Considering just the highest ranked analysis for each sentence, in Sampson, Haigh & Atwell's (1989) measure of correct rule application the parser scored a mean of 83.5% correct over all 250 sentences. Table 2 shows the results of this test—with respect to the original Susanne bracketings—using the Grammar Evaluation Interest Group scheme (GEIG, see e.g. Harrison *et al.*, 1991). This compares unlabelled bracketings derived from corpus treebanks with those derived from parses for the same sentences by computing *recall*, the ratio of matched brackets over all brackets in the treebank; *precision*, the ratio of matched brackets over all brackets found by the parser; 'crossing' brackets, the number of times a bracketed sequence output by the parser overlaps with one from the treebank but neither is properly contained in the other; and *minC*, the number of sentences for which all of the analyses had one or more crossings. The table also gives an indication of the best and worst possible performance of the disambiguation component of the system, showing the results obtained when parse selection is replaced by a simple random choice, and the results of evaluating the manually-created treebank against the corresponding Susanne bracketings. In this latter figure, the mean number of crossings is greater than zero mainly because of compound noun bracketing ambiguity which our grammar does not attempt to resolve, always returning

[3]This is a strong measure, since it not only accounts for structural identity between trees, but also correct rule application at every node.

| | minC | Crossings | Recall (%) | Precision (%) |
|---|---|---|---|---|
| *Probabilistic parser analyses* | | | | |
| Top-ranked 3 analyses, weighted = | 150 | 2.62 | 76.47 | 42.35 |
| Random 3 analyses, weighted = | 155 | 3.87 | 67.05 | 37.40 |
| *Manually-disambiguated analyses* | | | | |
| Single analysis | 91 | 0.88 | 91.51 | 50.73 |

Table 2: GEIG evaluation metrics for test set of 250 unseen sentences (lengths 3–56 words, mean length 18.2)

a right-branching binary analysis.

Black (1993:7) uses the crossing brackets measure to define a notion of structural consistency, where the structural consistency rate for the grammar is defined as the proportion of sentences for which at least one analysis contains no crossing brackets, and reports a rate of around 95% for the IBM grammar tested on the computer manual corpus. The problem with the GEIG scheme and with structural consistency is that both are still weak measures (designed to avoid problems of parser/treebank representational compatibility) which lead to unintuitive numbers whose significance still depends heavily on details of the relationship between the representations compared (c.f. the compound noun issue mentioned above).

Schabes *et al.* (1993) and Magerman (1995) report results using the GEIG evaluation scheme which are numerically superior to ours. However, their experiments are not strictly compatible because they both utilise more homogeneous and probably simpler corpora. In addition, Schabes *et al.* do not recover tree labelling, whilst Magerman has developed a parser designed to produce identical analyses to those used in the Penn Treebank, removing the problem of spurious errors due to grammatical incompatibility. Both these approaches achieve better coverage by constructing the grammar fully automatically. No one has yet shown that any robust parser is practical and useful for some NLP task. However, it seems likely that say rule-to-rule semantic interpretation will be easier with hand-constructed grammars with an explicit, determinate ruleset. A more meaningful comparison will require application of different parsers to an identical and extended test suite and utilisation of a more stringent standard evaluation procedure sensitive to node labellings.

### Parse Selection and Punctuation

In order to assess the contribution of punctuation to the selection of the correct analysis, we applied the same trained version of the integrated grammar to the 106 sentences from the test set which contain internal punctuation, both with and without the punctuation marks in the input. A comparison of the GEIG evaluation metrics for this set of sentences punctuated and unpunctuated gives a measure of the contribution of punctuation to parse selection on this data. (The results for the unpunctuated set were computed against a version of the Susanne treebank from which punctuation had also been removed.) As table 3 shows, recall declines by 10%, precision by 5% and there are an average of 1.27 more crossing brackets per sentence. These results indicate clearly that punctuation and text grammatical constraints can play an important role in parse selection.

| | minC | Crossings | Recall (%) | Precision (%) |
|---|---|---|---|---|
| *With punctuation* Top-ranked 3 analyses, weighted = | 78 | 3.25 | 74.38 | 40.78 |
| *Punctuation removed* Top-ranked 3 analyses, weighted = | 82 | 4.52 | 65.54 | 35.95 |

Table 3: GEIG evaluation metrics for test set of 106 unseen punctuated sentences (mean length with punctuation 21.4 words; without, 19.6)

## 6 Conclusions

Briscoe and Carroll (1993) and Carroll (1993) showed that the LR model, combined with a grammar exploiting subcategorisation constraints, could achieve good parse selection accuracy but at the expense of poor coverage of free text. The results reported here suggest that improved coverage of heterogeneous text can be achieved by exploiting textual and grammatical constraints on PoS and punctuation sequences. The experiments show that grammatical coverage can be greatly increased by relaxing subcategorisation constraints, and that text grammatical or punctuation-cued constraints can reduce ambiguity and increase coverage during parsing.

To our knowledge these are the first experiments which objectively demonstrate the utility of punctuation for resolving syntactic ambiguity and improving parser coverage. They extend work by Jones (1994) and Briscoe and Carroll (1994) by applying a wide-coverage text grammar to substantial quantities of naturally-punctuated text and by quantifying the contribution of punctuation to ambiguity resolution in a well-defined probabilistic parse selection model.

Accurate enough parse selection for practical applications will require a more lexicalised system. Magerman's (1995) parser is an extension of the history-based parsing approach developed at IBM (e.g. Black, 1993) in which rules are conditioned on lexical and other (essentially arbitrary) information available in the parse history. In future work, we intend to explore a more restricted and semantically-driven version of this approach in which, firstly, probabilities are associated with different subcategorisation possibilities, and secondly, alternative predicate-argument structures derived from the grammar are ranked probabilistically. However, the massively increased coverage obtained here by relaxing subcategorisation constraints underlines the need to acquire accurate and complete subcategorisation frames in a corpus-driven fashion, before such constraints can be exploited robustly and effectively with free text.

## References

Billot, S. and Lang, B. 1989. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Meeting of Association for Computational Linguistics*, 143–151. Vancouver, Canada.

Black, E., Garside, R. and Leech, G. (eds.) 1993. *Statistically-Driven Computer Grammars of English: The IBM/ Lancaster Approach.* Rodopi, Amsterdam.

Briscoe, E. 1994. Prospects for practical parsing of unrestricted text: robust statistical parsing techniques. In Oostdijk, N & de Haan, P. eds. *Corpus-based Research into Language.* Rodopi, Amsterdam: 97–120.

Briscoe, E. and Carroll, J. 1993. Generalised probabilistic LR parsing for unification-based grammars. *Computational Linguistics* 19.1: 25–60.

Briscoe, E. and Carroll, J. 1994. *Parsing (with) Punctuation.* Rank Xerox Research Centre, Grenoble, MLTT-TR-007.

Briscoe, E., Grover, C., Boguraev, B. and Carroll, J. 1987. A formalism and environment for the development of a large grammar of English. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, 703–708. Milan, Italy.

Carroll, J. 1993. *Practical unification-based parsing of natural language.* Cambridge University, Computer Laboratory, TR-314.

Carroll, J. 1994. Relating complexity to practical performance in parsing with wide-coverage unification grammars. In *Proceedings of the 32nd Meeting of Association for Computational Linguistics*, 287–294. Las Cruces, NM.

Carroll, J. and Grover, C. 1989. The derivation of a large computational lexicon for English from LDOCE. In Boguraev, B. and Briscoe, E. eds. *Computational Lexicography for Natural Language Processing.* Longman, London: 117–134.

Church, K. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the 2nd Conference on Applied Natural Language Processing*, 136–143. Austin, Texas.

Elworthy, D. 1993. *Part-of-speech tagging and phrasal tagging.* Acquilex-II Working Paper 10, Cambridge University Computer Laboratory (can be obtained from cide@cup.cam.ac.uk).

Elworthy, D. 1994. Does Baum-Welch re-estimation help taggers?. In *Proceedings of the 4th Conf. Applied NLP.* Stuttgart, Germany.

Garside, R., Leech, G. and Sampson, G. 1987. *Computational analysis of English.* Longman, London.

Grishman, R., Macleod, C. and Meyers, A. 1994. Comlex syntax: building a computational lexicon. In *Proceedings of the International Conference on Computational Linguistics, COLING-94*, 268–272. Kyoto, Japan.

Grover, C., Carroll, J. and Briscoe, E. 1993. *The Alvey Natural Language Tools Grammar (4th Release).* Cambridge University Computer Laboratory, TR-284.

Harrison, P., Abney, S., Black, E., Flickenger, D., Gdaniec, C., Grishman, R., Hindle, D., Ingria, B., Marcus, M., Santorini, B. and Strzalkowski, T. 1991. Evaluating syntax performance of parser/grammars of English. In *Proceedings of the Workshop on Evaluating Natural Language Processing Systems.* ACL.

Hindle, D. 1989. Acquiring disambiguation rules from text. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, 118–25. Vancouver, Canada.

Jackendoff, R 1977. *X-bar Syntax.* MIT Press; Cambridge, MA..

Jones, B 1994. Can punctuation help parsing?. In *Proceedings of the Coling94.* Kyoto, Japan.

Magerman, D. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd annul Meeting of the Association for Computational Linguistics.* Boston, MA.

de Marcken, C. 1990. Parsing the LOB corpus. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, 243–251. New York.

Nunberg, G. 1990. *The linguistics of punctuation.* CSLI Lecture Notes 18, Stanford, CA.

Pereira, F. and Warren, D. 1980. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence* 13.3: 231–278.

Sampson, G. 1994. Susanne: a Doomsday book of English grammar. In Oostdijk, N & de Haan, P. eds. *Corpus-based Research into Language.* Rodopi, Amsterdam: 169–188.

Sampson, G., Haigh, R., and Atwell, E. 1989. Natural language analysis by stochastic optimization: a progress report on Project APRIL. *Journal of Experimental and Theoretical Artificial Intelligence* 1: 271–287.

Schabes, Y., Roth, M. and Osborne, R. 1993. Parsing of the Wall Street Journal with the inside-outside algorithm. In *Proceedings of the Meeting of European Association for Computational Linguistics.* Utrecht, The Netherlands.

Taylor, L. and Knowles, G. 1988. *Manual of information to accompany the SEC corpus: the machine-readable corpus of spoken English.* University of Lancaster, UK, Ms..

# AN ABSTRACT MACHINE FOR ATTRIBUTE-VALUE LOGICS

Bob Carpenter Yan Qu
Computational Linguistics Program, Carnegie Mellon University
carp+@cmu.edu yqu@cs.cmu.edu

## Abstract

A direct abstract machine implementation of the core attribute-value logic operations is shown to decrease the number of operations and conserve the amount of storage required when compared to interpreters or indirect compilers. In this paper, we describe the fundamental data structures and compilation techniques that we have employed to develop a unification and constraint-resolution engine capable of performance rivaling that of directly compiled Prolog terms while greatly exceeding Prolog in flexibility, expressiveness and modularity.

In this paper, we will discuss the core architecture of our machine. We begin with a survey of the data structures supporting the small set of attribute-value logic instructions. These instructions manipulate feature structures by means of features, equality, and typing, and manipulate the program state by search and sequencing operations. We further show how these core operations can be integrated with a broad range of standard parsing techniques.

Feature structures improve upon Prolog terms by allowing data to be organized by feature rather than by position. This encourages modular program development through the use of sparse structural descriptions which can be logically conjoined into larger units and directly executed. Standard linguistic representations, even of relatively simple local syntactic and semantic structures, typically run to hundreds of substructures. The type discipline we impose organizes information in an object-oriented manner by the multiple inheritance of classes and their associated features and type value constraints. In practice, this allows the construction of large-scale grammars in a relatively short period of time.

At run-time, eager copying and structure-sharing is replaced with lazy, incremental, and localized branch and write operations. In order to allow for applications with parallel search, incremental backtracking can be localized to disjunctive choice points within the description of a single structure, thus supporting the kind of conditional mutual consistency checks used in modern grammatical theories such as HPSG, GB, and LFG. Further attention is paid to the byte-coding of instructions and their efficient indexing and subsequent retrieval, all of which is keyed on type information.

## 1 Motivation

Modern attribute-value constraint-based grammars share their primary operational structure with logic programs. In the past decade, Prolog compilers, such as Warren's Abstract Machine (Aït-Kaci 1990), have supplanted interpreters as the execution method of choice for logic programs. This is in large part due to a 50-fold speed up in execution times and a reduction by an order of magnitude in terms of space required. In addition to efficiency, compilation also brings the opportunity for static error detection.

The vast majority of the time and space used by traditional unification-based grammar interpreters is spent on copying and unifying feature structures. For example, in a bottom-up chart parser, the standard process would be first to build a feature structure for a lexical entry, then to build the feature structures for the relevant rules, and then to unify the matching structures. The principal drawback to this approach is that complete feature structures have to be constructed, even though unification may result in failure. In the case of failure, this can amount to a substantial amount of wasted time and space. By adopting an incremental compiled approach, a description is compiled into a set of abstract machine instructions. At run-time a description is evaluated incrementally, one instruction at a time. In this way, conflicts can be detected as early as possible, before any irrelevant structure has been introduced. In practice, this often means that the inconsistency of a rule with a category can often be detected very

soon into the evaluation of its left corner (the leftmost daughter) or head. The reason that descriptions are compiled rather than entire representations is for the same reason as terms are compiled into low-level abstract machine instructions in the WAM, namely that it (1) allows unification to be replaced by the much faster abstract machine operations, and (2) it simplifies backtracking in the face of local non-determinism, which is the primary bottleneck for most unification-based parsers.

Compilation also allows static typing mechanisms to be exploited to reduce the amount of space used by a structure and to detect conceptual errors in user type declarations. For instance, as pointed out by Carpenter and Penn (1995), declaring the features appropriate for a given type of structure allows record-like structures to be used rather than lists of attribute-value pairs, resulting in a three-fold decrease in memory used per structure, as well as great savings in managing allocation and deallocation. In addition to the savings in space, the positions of the features are also known at compile time which eliminates the need to compare features and construct structure on the fly, another area which occupies a significant amount of time in interpreters. With type unification determined statically, type information can be incorporated by means of table look up, which often avoids the need for more complex structural unification. Errors such as undefined features, inappropriate types, inconsistent pairings of features, and so on can be easily detected at compile time.

## 2 Abstract Machine Architecture

There is more than a passing similarity between the structure and execution of logic programs and that of "unification-based" grammars; (Carpenter 1992) shows that the resolution model of Prolog execution can be generalized to grammars based on typed attribute-value logics.

Our architecture is based on a typed, breadth-first extension of Warren's Abstract Machine for compiling Prolog (Aït-Kaci 1991). The WAM defines a mapping between logical rules expressed as Horn clauses, and the execution of a sequential machine machine (von Roy 1990). The development of the WAM has brought Prolog into the mainstream as an efficient programming language for industrial strength tasks. Similar gains were made by Lisp and ML when adequate compilers were developed for those functional languages.

A first pass an abstract machine architecture for grammars based on typed attribute-value logic was used for the ALE system (Carpenter and Penn 1994, 1995). ALE compiled typed attribute-value logic grammars down to Prolog clauses, which were then compiled using the WAM. The problem with Prolog as a target language is the lack of true random access, destructive memory operations, and pointers, all of which are necessary to generate efficient compiled code. On the other hand, Prolog is a handy intermediary because of its built-in handling of search and simple pattern matching. The architecture we describe here is the result of cutting out the middle man in the compilation of ALE, while retaining the efficiencies arising from the WAM.

Following ALE, categories are expressed using attribute-value logic descriptions. These descriptions contain feature information, type information, and equality information, logically structured by conjunction (sequencing) and disjunction (non-deterministic choice). In this paper, we show how these descriptions are incrementally executed. A description is decomposed to a number of operations which are then applied one by one to a feature structure; at each stage of execution, either the information is successfully added to the structure, or a failure flag is raised and backtracking ensues.

Various parsing mechanisms, such as the chart parser built into ALE or a left-corner parser can then be run off the basic description resolution mechanism. Any choice or copy points in the parser can simply be interleaved with the choice points for the descriptions in the rules or lexical entries.

Aït-Kaci and Di Cosmo (1993) developed a compiler for sorted terms that are similar to ours, but without any type declarations for feature appropriateness. Their representation of structures is based on lists of feature/value pairs, which consumes around three times as much memory space as our encoding, and prevents the precompilation of unifications (see Carpenter

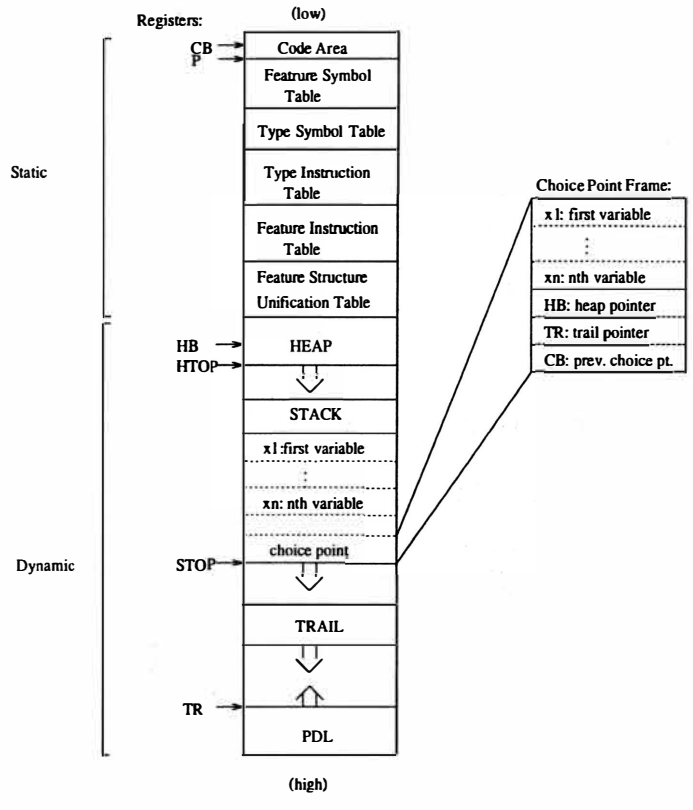



Figure 1: **Abstract Machine Memory Layout and Registers**

and Penn 1995). Wintner and Francez (1994) present a representation of feature structures based in large part on our approach (Qu 1994; Carpenter and Penn 1994). But their grammars are given in terms of Aït-Kaci's $\psi$-terms, which are suboptimal in terms of their control over choice points and "anonymous" variables; they are currently working to integrate disjunction into their architecture in a manner similar to ours (Wintner p.c.).

In the rest of this section, we describe the memory layout, instruction set, and execution mechanism of our system.

### 2.1 Memory areas

In this section, we describe the allocation of memory in the machine. Memory is divided into two major logical areas (Figure 1): static and dynamic. The static memory holds the program and its associated type declarations. This includes the symbol tables, type unification and type inference information, as well as the instructions associated with lexical entries and phrase structure schemata.

The dynamic memory holds the program execution state. Its primary areas include the **heap**, which contains the representations of feature structures, the **stack**, which keeps track of the local bindings of arguments, variables and structures (a stack provides a natural (though not the most economical) method for handling the allocation and deallocation of space for nested feature values; the values of variables reside at the bottom of the stack), the **trail**, which keeps track of information that needs to be restored on backtracking (pairs of addresses and their previous values), the **choice point stack**, which holds information about variable and stack bindings, an indicator of where to resume control, how far to unwind the trail, as well as information on recovery point in the heap and the latest choice point, and finally, the **push down list**, another stack structure which is used to store sequences of feature structures remaining to be unified for the unification algorithm.

```
% Type Signature

bot sub [t,s].
 t sub [t1,t2]
   intro [h:bot].
  t1 sub [t3]
     intro [f:bot].
  t2 sub [t3]
     intro [g:bot].
    t3 sub []
       intro [j:bot].
 s sub [s1,s2].
  s1 sub [].
  s2 sub [].
```

```
% Lexicon

word1 ---> t1.
word2 ---> f:t.
word3 ---> t1, f:(t2, g:t2), h:t3.
word4 ---> t1, f:(X, g:t2), h:X.
word5 ---> t1; t2.

% Grammar

rule1 rule
(s)
==>
cat> (t1, f:X),
cat> (t2, g:X).
```

Figure 2: **An Example Grammar in ALE Format**

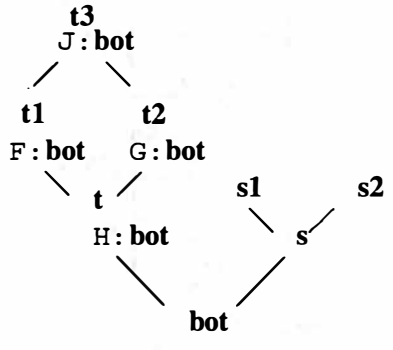



Figure 3: **An Example Type Hierarchy**

### 2.2 Representation of feature structures

In this section, we define an internal representation for feature structures in the abstract machine. The global block storage for representing feature structures is an addressable heap. The heap is an array of data cells, and is managed as a stack.

We distinguish three kinds of terms for representing feature structures: a *variable*, a *structure*, and a *pointer*. As in the WAM, we use explicit *tags* as part of the format of some heap cells to discriminate between these three sorts of heap data. The tags for the three types of data are STR, VAR, and PTR.

Recall that a feature structure Fs consists of two components: a type and a list of $n$ feature values, $v_1, \ldots, v_n$, each of which is itself a feature structure. Such a structure is represented on the heap by means of $n+1$ contiguous cells. The type value is represented as a cell tagged by STR, as denoted as **<STR, $\tau$>**, where $\tau$ is the value for the feature structure Fs. The $n$ other cells contain references to the $n$ appropriate features (in alphabetical order in the current compiler). The $n$ heap cells representing the values will typically contain pointers to other structures, of the form **<PTR, $p$>**, where $p$ is the index of another heap cell. Alternatively, if the value is a structure with no features, it can fit into one cell, and will thus not need to employ a level of indirection through the pointers. Similarly, we use a variable structure, where **<VAR, $\tau$>** represents a structure of type $\tau$ whose feature values are unknown. By using variable structures of this kind, we are able to postpone some of the eager type inference that is automatically performed in ALE; if nothing is known about a structure's feature's values, we can leave them as variables, thus conserving space and time in the short run (these areas may be later overwritten with actual values and consume space on backtracking). For example, suppose we have a type hierarchy specification as in Figure 2. The diagram in Figure 3 displays the type signature in Figure 2 graphically. Figure 4 shows a heap representation for a feature structure satisfying the description `t1, f:(t2, g:t2), h:t3` starts at heap address 11. Note that the value for feature F is represented at heap address 12 as a PTR to a STR cell at address 14 because the

| | | |
|---|---|---|
| 11 | STR | $t_1$ |
| 12 | PTR | 14 |
| 13 | VAR | $t_3$ |
| 14 | STR | $t_2$ |
| 15 | VAR | $t_2$ |
| 16 | VAR | *bot* |

Figure 4: **Example Heap Representation**

| | |
|---|---|
| HTOP | Top of the heap |
| STOP | Top of the stack |
| P | Program counter |
| CB | Continuation code counter |
| HB | Heap pointer |
| TR | Trail pointer |
| $x_1, x_2, \ldots, x_n$ | Registers for argument passing and temporary storage |

Figure 5: **Execution State**

feature value for feature G of type **t2** is explicitly specified. Unspecified information on the type appropriateness of feature H of type **t2** is automatically recovered at address 16 with the default type **bot**, which is taken to be the most general type in the hierarchy. The value for feature H of type **t1** is represented as a VAR cell at address 13 rather than a STR cell, as no explicit feature information is specified for the structure of type **t3**.

### 2.3 Execution state

A description of the information stored to represent a given execution state of the machine is given in Figure 5.

### 2.4 The Instruction set

Figure 6 contains the instruction set for the abstract machine with a brief description of what each instruction does. The registers and arguments following some instructions are left out in the table. For details of these instructions, see (Qu 1994).

## 3 Compiling

In this section, we discuss how the typed feature structure logic can be compiled using the abstract machine. The syntactic representation form we adopt here follows the ALE format as described in (Carpenter and Penn 1994). We will first describe how type definitions are compiled, then we will explain how the descriptions are compiled, and lastly, we will discuss how the grammar rules are compiled.

### 3.1 Compiling type definitions

The first component of a grammar is a type definition. Type definitions include types for feature structures and declarations of appropriate features for each type. Enforcing an inheritance-based type discipline on feature structures yields the following characteristics for type and feature structure interaction:

- *Constraint Inheritance*: Type constraints on more general types are inherited by their more specific subtypes.

| Instructions for adding a type | |
|---|---|
| BIND | Bind a new feature cell to an old feature cell |
| FRESH | Create a new feature cell |
| **Instructions for getting a feature from a type** | |
| GET | Give the position of the feature for a feature structure |
| ADD | Create a new feature structure with the type as value |
| **Instructions for procedural control** | |
| ALLOCATE | Allocate a new environment on the stack |
| DEALLOCATE | Deallocate an environment on the stack |
| JUMP | Jump to labeled clause and continue execution |
| **Instructions for unification of feature structures** | |
| SKIP1 | Skip one heap cell for feature structure one |
| SKIP2 | Skip one heap cell for feature structure two |
| UNIFY | Unify two heap cells |
| COPY1 | Copy the heap cell of structure one |
| COPY2 | Copy the heap cell of structure two |
| **Choice instructions** | |
| TRY | Allocate a new choice point |
| RETRY | Unwind heap and try next alternative (branch) |
| LASTTRY | Unwind heap and try last alternative (tail recursive) |
| **Unifying instructions** | |
| ADDNEW | Add a new type onto the heap |
| PUSH | Push the feature value to the stack |
| POP | Pop a value off stack |
| UNIFY_VAR | Update the binding of variables |

Figure 6: **Abstract Machine Instructions**

- *Feature Appropriateness*: Each type must specify which features it can be defined for, and which types of value the features must take. The feature appropriateness is inherited along the type hierarchy in two ways.
  - Feature restrictions are inherited. That is, if a feature is appropriate for a type, then it is appropriate for all of the subtypes of the type.
  - Type restrictions on feature values are inherited. That is, if a type is the appropriate value for a feature, then all of its subtypes are appropriate values for the feature.

For example, consider the type inheritance relations in Figure 2, which are represented as a graph in Figure 3. The type t introduces one feature H and appropriate value constraints. The subtype t1 of t inherits the feature H and introduces a new feature F.

Compiling signatures involves the compilation of the type hierarchy and appropriateness conditions. The five global static storage areas are associated with this compilation: the symbol tables for types and features, the table for feature structure unification instructions, the table for type instructions, and the table for feature-value instructions. These instructions are all byte coded.

The feature structure unification table is indexed on pairs of types, and indicates first whether the types are consistent, and if so, what the value type is and how the features match up for further unification and type inference. The type instructions are also indexed on pairs of types, and indicate how to add a type's information to another structure and perform the relevant type inferences. The feature instructions are indexed by a feature and a type and indicate how to take the feature's value at the type, including any necessary type coercions. All instructions might also indicate failure.

### 3.2 Compiling descriptions

The compilation of descriptions follows the five clauses in the recursive definition of descriptions for types, features, variables, conjunction, and disjunction. Figure 7 shows the BNF grammar

```
⟨ desc ⟩ ::= ⟨ type ⟩
           | ⟨ variable ⟩
           | (⟨ feature ⟩: ⟨ desc ⟩)
           | (⟨ desc ⟩, ⟨ desc ⟩) ;; conjunction
           | (⟨ desc ⟩; ⟨ desc ⟩) ;; disjunction
```

Figure 7: **BNF Grammar for Descriptions in ALE**

| | | |
|---|---|---|
| 14 | STR | $t_2$ |
| 15 | VAR | $t_2$ |
| 16 | VAR | *bot* |
| 17 | STR | $t_3$ |
| 18 | VAR | *bot* |
| 19 | PTR | 15 |
| 20 | PTR | 16 |
| 21 | VAR | *bot* |

Figure 8: **Adding a Type to Heap**

for a description. A description is always evaluated by adding its information to the structure indicated by the top of the stack, which will point into the heap.

### Compiling a type

The abstract machine instruction for adding a type to a feature structure is ADDNEW. This instruction adds the information associated with the type to a structure located at a specific heap address *heapaddr*. After dereferencing the structure by following PTR values, the result may be either a VAR cell or a STR cell. If the result is a VAR cell, we can just look up the result of unification in a table and update the variable structure's type (trailing if it is new), and backtrack if unification fails. Otherwise, assuming the type being added is $\tau$ and the resulting structure cell represents a feature structure of type $\tau'$, we need to consider four possibilities. If $\tau$ and $\tau'$ can't unify, we simply backtrack. If the new type $\tau$ is subsumed by the type $\tau'$ of the structure in the heap, nothing needs to be done and the command merely succeeds. Otherwise, we need to create a new structure on the heap for the result of the unification if $\tau'$ subsumes $\tau$ or if $\tau$ and $\tau'$ unify to a new type.

For example, suppose type **t1** is to be added to heap address 14 in Figure 8. From the type hierarchy in Figure 3, we know that **t1** and **t2** unify to a new type **t3**. Thus a new structure has to be created at heap address 17 and the old structure must be redirected to it by means of a PTR cell replacing the old value at 14 (which must be recorded on the trail if it might be needed for backtracking). Then we look up the precompiled instructions for adding a type $\tau'$ to a structure of type $\tau$, which might involve creating new features with their most general values (which can be represented as VAR cells). The sequence of instructions for adding type **t1** to a structure of type **t2** is: FRESH (creating a new feature F for type **t3**), BIND (binding to previous value for feature H), BIND (binding to previous value for feature G), and FRESH (creating a new feature J for type **t3**); recall that the features are in alphabetic order. In general, the FRESH commands will indicate the appropriate type of the variable structure to be created.

### Compiling feature/value pairs

To add a description of the form of `<feat>:<desc>` to a structure at a given heap address, we need to first find the value of the feature and then apply the description to the resulting value. If the structure is a variable cell, it must first have its appropriate features added with variable

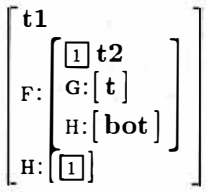



Figure 9: **Structure Sharing: Feature Structure Notation.**

```
      addnew    t1
1:    push      f
1:    unify_var 1
      addnew    t2
2:    push      g
      addnew    t
2:    pop
1:    pop
3:    push      h
2:    unify_var 1
3:    pop
```

(a)

| | | |
|---|---|---|
| 22 | STR | $t_1$ |
| 23 | PTR | 25 |
| 24 | PTR | 23 |
| 25 | STR | $t2$ |
| 26 | VAR | $t$ |
| 27 | VAR | *bot* |

(b)

Figure 10: **Structure Sharing: (a) Machine Instructions and (b) HEAP Representations After Dereferencing.**

values of the appropriate type. Next, if the feature is absent from the structure, the structure is coerced to one that is appropriate for the type by adding the most general type appropriate for the feature, which may result in failure and backtracking. Finally, we PUSH the value of the feature that is now guaranteed to exist onto the top of the stack and recursively add the description. After all of the information in a description is added to a feature value, we can POP the value from the stack. Note that if we do not need to reuse the structure currently at the top of the stack, we can overwrite the current stack position rather than pushing a new value.

### Compiling variables

A variable occurring in two locations in a description has the same interpretation as in Prolog — the structures must be identical. For example, the variable `X` in the description `(t1, f:(X, g:t2), h:X)` indicates that feature `h` and feature `f` share the same value (see Figure 9, which represents the most general structure that satisfies this description).

In the abstract machine, space is allocated at the bottom of the stack for all of the local variables in a description. When the first instance of a variable is encountered, the variable's position on the stack is bound to the current structure on the top of the stack (a more clever compiler for variables, such as that found in the WAM, could avoid some of this duplication of representations on the stack). Any later attempts to add the same variable will simply unify the current structure with the value of the variable. A variable's lifespan is the entire phrase structure rule in which it occurs, which allows sharing across daughter and mother categories. Note that live variables (those whose last occurrence has not been encountered) must be maintained as part of the information in a choice point. The value of a variable need not be stored beyond its last occurrence in a clause.

The instruction UNIFY_VAR deals with a variable; note that it takes an argument that indicates the position of the variable in the stack to allow random access. The "unification" of a variable either sets the value of the variable on first encounter, or unifies the current structure with the previously established value of the variable. As can be seen with descriptions such as `(f:X ; g:Y), h:X`, the first use of a variable cannot be predicted statically; if the first branch

of the disjunction is taken, then `X` is set to the value of `f`, and then the value of `h` is unified with the value set by `f`, but if the second branch of the disjunction is taken, the value of `X` must be set by `h`. Note that a reasonable compiler can reuse the space allocated for a variable after its last instance is encountered. For instance, in the description `f:X, g:X, h:Y, j:Y`, the variables `X` and `Y` can use the same space, as long as it is reset to null after the last occurrence of `X`.

The instructions for the description `t1, f:(X, g:t2), h:X` are given in Figure 10(a). The resulting heap representation after their execution is shown in Figure 10(b). 1:UNIFY_VAR(1) sets the binding of `X`, and 2:UNIFY_VAR(1) unifies the current structure with the structure previously bound to `X`.

**Compiling unification**

It is significant to note that shared variables are the only instructions that call the unification operation. All other structural manipulation is carried out by the other primitive instructions. Instructions for unifying two feature structures at two different heap addresses are stored in a table, which can be represented as a (sparse) two dimensional array. There are a number of cases depending on whether the structures are variables or not, and whether they fail to unify, unify to a new type, or unify to one of their existing types. If one of the structures is a variable, it is simply updated as a pointer pointing to the other structure, and its type is added to the other structure. If one of the structures has a type subsuming the other, the more general structure is made to point to the more specific structure, and the features that need to be unified are read out of the table entry for the two types. These are indicated in terms of UNIFY and SKIP instructions, indicating whether the features in the more specific structure should be unified with the next feature in the more general structure or skipped. Finally, if the two structures unify to a new type of structure, a new structure must be allocated of the appropriate type and both previous structures are made to point to it. Then the new values are constructed as either pointers to a value in one of the existing structures or by creating a fresh variable cell. If both structures have a feature, these values must be pushed onto the unification stack so that their values are eventually unified. The instruction used is COPY($n$) if the value is simply pointing to the value in the first or second structure (indicated by $n$), UNIFY if the value is determined by unifying the next value in both structures, or FRESH($\tau$) if a new variable structure of type $\tau$ should be created. As usual, any updates are trailed, and any failures cause backtracking.

**Compiling conjunctions of descriptions**

Conjunction amounts to simple sequencing of operations, which is the standard method by which the program pointer moves through the code area. The way in which information is packaged through conjunction and backtracking has a significant impact on run-time efficiency.

Different descriptions may be logically equivalent, but vary in performance, as do the examples in Figure 11(a), all of which generate the structure in Figure 11(b). The shortest form (1) is the most efficient while the longest description (3) is the least efficient. This can be seen from the size of the code in Figure 11(a) for each description. From the size of instructions, we can see that (2) and (3) involve more steps to represent the same feature structure. This ability to use concise descriptions is one significant way in which attribute-value logic descriptions provide finer control over term evaluation than Prolog terms; the order and amount of information added can be controlled by the programmer in a logically transparent fashion.

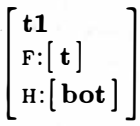
**Compiling disjunctions of descriptions**

When disjunction exists in descriptions, a failure of unification no longer yields irrevocable abortion of execution. Like the WAM, the execution of a description is left to right, and depth-first. When encountering a failure or a call for additional solutions, execution returns to the last considered choice point, restores information about variable binding, and continues execution.

Unlike the WAM, which only has one representation on the heap at any execution time, many parsing strategies rely on the dynamic programming technique of storing intermediate

| (1) f:t | (2) t1, f:t | (3) t1, f:t, h:bot |
|---|---|---|
| push f<br>addnew t<br>pop | addnew t1<br>push f<br>addnew t<br>pop | addnew t1<br>push f<br>addnew t<br>pop<br>push h<br>addnew bot<br>pop |

(a)




(b)

Figure 11: **Different Descriptions, Same Feature Structure**

results. To accommodate this behavior, we have built in mechanisms for copying structures and reusing them — our memory architecture is not sensitive to the memory area into which a feature structure pointer is directed. In addition, the trail allows such "permanent" memory to be modified incrementally and copied later, thus avoiding one of the major time sinks of ALE (Carpenter and Penn 1995).

We now consider how disjunction executes. First, when a disjunct is evaluated, it may create side effects on the stack and heap by updating the variable binding on the stack and changing the feature structure values of the heap. These effects must be undone when considering an alternative. The data area TRAIL is used to keep track of all the heap cells that have been updated for the chosen disjunct and that need to be restored for another disjunct. TRAIL($a$), which is the operation on heap address $a$, allocates two TRAIL cells to record the heap address and its current value.

Only conditional bindings, those affecting a variable existing before creation of the current choice point, need to be trailed. In our case, a conditional binding is one affecting the content of the heap cell before creation of the current choice point. To determine this, we use a global register HB to contain value of HTOP set at the time of latest choice point.

Like the WAM, we use a choice point frame to remember all the necessary states for restoring and continuing execution upon backtracking. Every choice point frame is initially pushed on the top of the choice point stack. The following information need to be stored in a choice point frame:

- *The next clause* (value of register CB): the next clause to try upon backtracking.
- *The current trail pointer* (value of register TR): the boundary where to *unwind* the trail upon alternative definition building.
- *The current top of heap* (value of register HB): the point before which the copying algorithm should consider. Heap cells after this point are created by the current choice point and need not to be copied.
- *Variable bindings* $x_1, \ldots, x_n$, where $n$ is the number of variables in this description, including the implicit values created by feature values. If $x_i$ is bound after the choice point, it is updated by the current heap address to which it is bound.

There are four instructions related to compiling disjunctions: TRY, RETRY, LASTTRY and JUMP. Descriptions of these instructions can be found in Figure 6. Consider how these instructions are used for compiling description `(t1;t2;t3), h:t`, the instructions for which are given in Figure 12. TRY allocates a choice point frame on the stack, then continues execution with the following instructions. JUMP directs the execution to the labeled instruction. RETRY resets all necessary information after backtracking, allocate a new choice point, then continues execution of the following instructions. LASTTRY is like TRY, but it indicates that variables need no longer be trailed because there are no more backtracking alternatives.

In evaluating disjunctions, two alternatives are possible. First, disjunctions can be handled by backtracking as in the WAM, using the structures for choice points we have described.

```
        try
        addnew   t1
        jump     l
        retry
        addnew   t2
        jump     l
        lasttry
        addnew   t3
l:      push     h
        addnew   t
        pop
```

Figure 12: **Compilation of disjunctions**

In this case, a choice point is recorded before the first alternative is considered. The second alternative is to eagerly copy, which allows chart-like memoization, thus avoiding the need for recording a choice point. The copying algorithm is not trivial, as a feature structure may be dispensed among different areas of heap spaces, and we need to constant looking up the trail to see whether a particular heap cell has been changed by the previous alternative or not. Details of how copying is done can be found in (Qu 1994).

## 4 Compiling a Parser

The architecture we have outlined for the processing of descriptions, with its features of incremental execution and efficient backtracking, make it an ideal candidate for integration with either a parser or a definite clause grammar system, or both, as found in ALE.

As described by Carpenter and Penn (1995), grammar rules can be compiled so as to have their own choice points interleaved with those of their embedded descriptions. In general, choice will be a matter of which grammar rule to apply, and which category to apply it to. In a simple chart parsing regime, the only matter that needs to be attended to is that of how the indexing will be done. For instance, in a bottom-up left-to-right chart-parser, after an edge is completed, all active edges immediately preceding it must be found and tested against it. To resume parsing an active edge, all that is needed is a pointer into the program space and a record of all of the variable instantiations. In testing the application of a grammar rule against a new completed edge, we must start by executing the first daughter description on the complete category. Thus choice points arising between different edges and between rules can be naturally accommodated in either a breadth-first (queue) or depth-first (stack) control strategy.

Memoizing parsers are not the only possibility for our description compilation scheme. Built into the model are the kind of management of non-determinism via a trail that lead to efficient implementations of backtracking parsers; for instance, LR or left-corner varieties. In the same way, definite clauses can be naturally integrated into our grammars and our control strategy, as demonstrated by Carpenter and Penn (1995).

## 5 Conclusion

We demonstrated how an abstract machine can be constructed for typed attribute-value logics along the lines of the Warren Abstract Machine for Prolog. By directly implementing the appropriate data structures, indexing mechanisms, and search engines, a parser can be built for feature structures that is as fast as the execution of a Prolog compiler. This greatly improves on the current state of the art, which is represented by either direct interpreted grammars or ones which are indirectly compiled by means of a detour through Prolog.

# References


[1] H. Aït-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction.* MIT Press, 1991.

[2] H. Aït-Kaci and R. Di Cosmo. Compiling order-sorted feature term unification. Technical Report PRL 7, Digital Paris Research Laboratory, 1993.

[3] B. Carpenter. *The Logic of Typed Feature Structures.* Cambridge Tracts in Theoretical Computer Science 32. Cambridge University Press, New York, NY, 1992.

[4] B. Carpenter and G. Penn. Ale 2.0 users' guide - the attribute-logic engine. Technical report, Computational Lingusitcs, Carnegie Mellon University, 1994.

[5] B. Carpenter and G. Penn. Compiling typed attribute-value logic grammars. In H. Bunt and M. Tomita, editors, *Current Issues in Parsing Technologies*, volume 2. Kluwer, Philadelphia, 1995.

[6] Y. Qu. An abstract machine for typed feature structure theories. Master's thesis, Carnegie Mellon University, 1994.

[7] P. von Roy. *Can Logic Programming Execute as Fast as Imperative Programming?* PhD thesis, University of California at Berkeley, 1990.

[8] S. Wintner and N. Francez. Abstract machine for typed feature structures. In *Proceedings of the Conference on Natural Language Understanding and Logic Programming*, 1994.

# A CHUNKING-AND-RAISING PARTIAL PARSER

Hsin-Hsi Chen
Yue-Shi Lee
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.
E-mail: hh_chen@csie.ntu.edu.tw; leeys@nlg.csie.ntu.edu.tw

## Abstract

Parsing is often seen as a combinatorial problem. It is not due to the properties of the natural languages, but due to the parsing strategies. This paper investigates a Constrained Grammar extracted from a Treebank and applies it in a non-combinatorial partial parser. This parser is a simpler version of a chunking-and-raising parser. The chunking and raising actions can be done in linear time. The short-term goal of this research is to help the development of a partially bracketed corpus, i.e., a simpler version of a treebank. The long-term goal is to provide high level linguistic constraints for many natural language applications.

## 1 Introduction

Recently, many parsers [1-10] have been proposed. Of these, some [1-7] belong to full parsers and some [8-10] partial parsers. Because the polycategory of a word and the use of the formal grammar, parsing is often seen as a combinatorial problem [11]. A feasible way to treat this problem is to separate the work of category determination from a parser and adopt a new parsing scheme. That is, automatic part-of-speech tagging serves as preprocessing of the parser. The tagging problem has been investigated by many researchers [12-18], and many interesting results have been demonstrated. Thus the remaining problem is how to construct a new non-combinatorial parser to increase the parsing efficiency and decrease the parsing ambiguity. This paper will propose a chunking-and-raising partial parser for such a goal. Section 2 introduces the framework of this parser. Section 3 specifies the training corpus - Lancaster Parsed Corpus, and Section 4 touches on how to extract Constrained Grammar from this corpus. Section 5 presents a simplified parsing algorithm based on Constrained Grammar. Before concluding the experimental results and the related works are shown.

## 2 Framework of a Chunking-and-Raising Parser

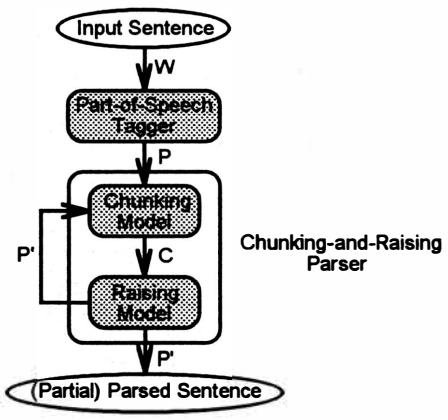



**Fig. 1. The Chunking-and-Raising Scheme**

In this scheme, parsing can be regarded as a sequence of actions of chunking and raising. Fig. 1 shows the configuration. An input sentence W is input to a part-of-speech tagger and a (lexical) tag sequence P is produced. The output of the tagger is the input of the parser. The chunking model of the parser groups some tags into chunks. The raising model assigns a (syntactic) tag to each chunk and generates a new tag sequence P'. The chunking and raising actions are repeated until no new chunking sequence is generated.

Consider an example. Let the input sentence be "Mr. Macleod went on with the conference at Lancaster House despite the crisis which had blown up .". The corresponding part-of-speech sequence is shown as follows.

NPT NP VBD RP IN ATI NN IN NP NPL IN ATI NN WDT HVD VBN RP .

The chunking model produces a chunking sequence shown below.

[ NPT NP ] [ VBD ] [ RP ] IN ATI NN IN [ NP NPL ] IN ATI NN [ WDT ] [ HVD VBN ] [ RP ] .

Seven parts-of-speech which cannot be formed into chunks at this step remain in the sequence. The raising model then generates the following chunking-and-raising sequence.

[ **N** NPT NP **N** ] [ **V** VBD **V** ] [ **R** RP **R** ] IN ATI NN IN [ **N** NP NPL **N** ] IN ATI NN [ **Nq** WDT **Nq** ] [ **V** HVD VBN **V** ] [ **R** RP **R** ] .

[ **N** NPT NP **N** ] denotes that the chunk [ NPT NP ] is raised to N. Similarly, the chunks [ VBD ], [ RP ], [ NP NPL ], [ WDT ], [ HVD VBN ] and [ RP ] are raised to V, R, N, Nq, V and R, respectively. The seven syntactic tags and the remaining lexical tags form a new tag sequence and it is sent to the next chunking-and-raising cycle. If the word information is put back into the sequence, a partial parsed sentence is generated as follows.

[ **N** Mr._NPT Macleod_NP **N** ] [ **V** went_VBD **V** ] [ **R** on_RP **R** ] with_IN the_ATI conference_NN at_IN [ **N** Lancaster_NP House_NPL **N** ] despite_IN the_ATI crisis_NN [ **Nq** which_WDT **Nq** ] [ **V** had_HVD blown_VBN **V** ] [ **R** up_RP **R** ] ._.

After one more chunking-and-raising cycle, the partial parsed sentence is generated as follows.

[ **N** Mr._NPT Macleod_NP **N** ] [ **V** went_VBD **V** ] [ **R** on_RP **R** ] with_IN the_ATI conference_NN [ **P** at_IN [ **N** Lancaster_NP House_NPL **N** ] **P** ] despite_IN the_ATI crisis_NN [ **Fr** [ **Nq** which_WDT **Nq** ] [ **V** had_HVD blown_VBN **V** ] [ **R** up_RP **R** ] **Fr** ] ._.

In other words, a new tag sequence "N V R IN ATI NN P IN ATI NN Fr ." is generated. We repeat these two actions until no more chunking sequence is generated.

A Constrained Grammar is extracted from a Treebank and is applied in a simpler version of chunking-and-raising parser. The chunking and raising actions are applied only once in this parser. Thus it only produces a linear chunking-and-raising sequence, not a hierarchical annotated tree. The experimental framework is shown in Fig. 2.

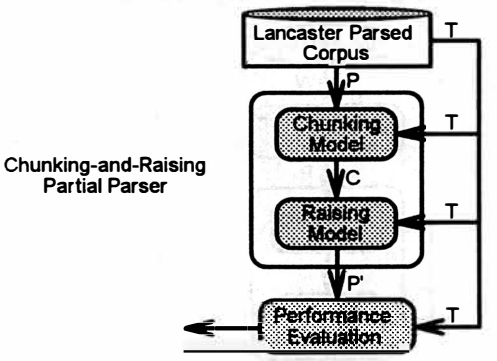



**Fig. 2. The Experimental Framework**

In this experiment, the Lancaster Parsed Corpus is adopted to train the chunking and raising models. Besides, it is also used in the performance evaluation.

## 3 Lancaster Parsed Corpus

The Lancaster Parsed Corpus is a modified and a condensed version of Lancaster-Oslo/Bergen (LOB) Corpus. It only contains one sixth of LOB Corpus, but involves more information than LOB Corpus. The corpus consists of fifteen kinds of texts (about 150,000 words). Each category corresponds to one file. The tagging set of Lancaster Parsed Corpus is extended and modified from LOB Corpus. The following shows a snapshot of Lancaster Parsed Corpus.

**A01 1**
[S[P by_IN [N Trevor_NP Williams_NP N]P] ._. S]

**A01 2**
[S[N a_AT move_NN [Ti[Vi to_TO stop_VB Vi][N \0Mr_NPT Gaitskell_NP N][P from_IN [Tg[Vg nominating_VBG Vg][N any_DTI more_AP labour_NN life_NN peers_NNS N]Tg]P]Ti]N][V is_BEZ V][Ti[Vi to_TO be_BE made_VBN Vi][P at_IN [N a_AT meeting_NN [Po of_INO [N labour_NN \0MPs_NPTS N]Po]N]P][N tomorrow_NR N]Ti] ._. S]

**A01 3**
[S&[N \0Mr_NPT Michael_NP Foot_NP N][V has_HVZ put_VBN V][R down_RP R][N a_AT resolution_NN [P on_IN [N the_ATI subject_NN N]P]N][S+ and_CC [Na he_PP3A Na][V is_BEZ V][Ti[Vi to_TO be_BE backed_VBN Vi][P by_IN [N \0Mr_NPT Will_NP Griffiths_NP ,_, [N \0MP_NPT [P for_IN [N Manchester_NP Exchange_NP N]P]N]N]P]Ti]S+] ._. S&]

**A01 4**
[S[Fa though_CS [Na they_PP3AS Na][V may_MD gather_VB V][N some_DTI left-wing_JJB support_NN N]Fa] ,_, [N a_AT large_JJ majority_NN [Po of_INO [N labour_NN \0MPs_NPTS N]Po]N][V are_BER V][J likely_JJ J][Ti[Vi to_TO turn_VB Vi][R down_RP R][N the_ATI Foot-Griffiths_NP resolution_NN N]Ti] ._. S]

**A01 5**
*'_*' [S[V abolish_VB V][N Lords_NPTS N] **'_**' ._. S]

These are extracted from the first five sentences of category A. Before each sentence, a unique reference number, e.g., "A01 1", denotes its source. Each word is appended with a lexical tag, e.g., "by_IN", "Trevor_NP". The syntactic tag is shown by opening and closing brackets.

To indicate that phrases or clauses are coordinated, the symbols "&", "-" or "+" will be used at the end of a phrase or a clause tag. An example is listed as follows.

[ N& mothers_NNS ,_, [ N- children_NNS N- ] [ N+ and_CC sick_JJ people_NNS N+ ] N& ]

The first coordinated phrase is not labeled any tag. The second and the third coordinated phrases are labeled N- and N+, respectively. This is because N- or N+ tends to include ellipsis. Table 1 gives an overview of the Lancaster Parsed Corpus. In our experiment, those parsed sentences that don't begin with "[S" and end with "S]" are removed from the training corpus. Thus "A01 5" is deleted.

**Table 1. The Overview of Lancaster Parsed Corpus**

| Category | # of Sentences | # of Words | Category | # of Sentences | # of Words |
|---|---|---|---|---|---|
| A | 3403 | 9410 | J | 2713 | 8336 |
| B | 3648 | 9999 | K | 5065 | 13587 |
| C | 2870 | 8225 | L | 5541 | 15556 |
| D | 3534 | 10110 | M | 3434 | 9179 |
| E | 2990 | 9356 | N | 5944 | 15751 |
| F | 2962 | 8562 | P | 6209 | 16766 |
| G | 2185 | 6813 | R | 3398 | 9443 |
| H | 2266 | 6524 | Total | 56162 | 157617 |

## 4 The Constrained Grammar

A Constrained Grammar is extracted from the Lancaster Parsed Corpus. Because the chunking and raising actions are applied only once in the preliminary experiment, only those rules that appear on the lowest level of the parsing trees form a Constrained Grammar.

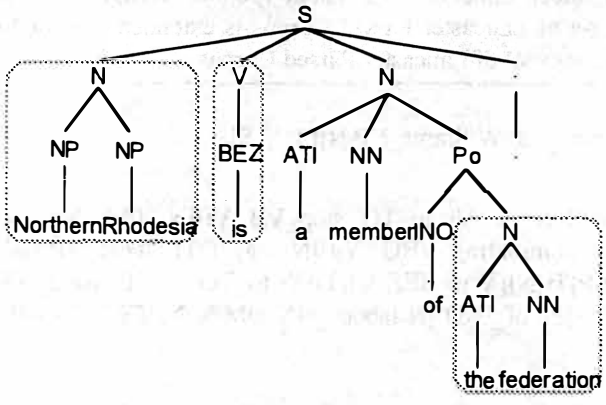



**Fig. 3. The Parsing Tree**

Consider a sentence "Northern Rhodesia is a member the federation .". Its parsing tree is shown in Fig. 3. Three constrained rules shown below are extracted from this parsing tree.

(*) NP NP (BEZ) -> N
(NP) BEZ (ATI) -> V
(INO) ATI NN (.) -> N

Two constraints enclosed in parentheses, i.e., the left and the right constraints, are added into each constrained rule. For example, the constrained rule, (NP) BEZ (ATI) -> V, has the left constraint NP and the right constraint ATI. It means that chunk [ BEZ ] can be raised to V when its left tag is NP and its right tag is ATI. The other two rules have the similar interpretations. The asterisk marks the beginning of the sentence. A more complicated example is given as follows:

[S[N a_AT move_NN [Ti[Vi to_TO stop_VB Vi][N \0Mr_NPT Gaitskell_NP N][P from_IN [Tg[Vg nominating_VBG Vg][N any_DTI more_AP labour_NN life_NN peers_NNS N]Tg]P]Ti]N][V is_BEZ V][Ti[Vi to_TO be_BE made_VBN Vi][P at_IN [N a_AT meeting_NN [Po of_INO [N labour_NN \0MPs_NPTS N]Po]N]P][N tomorrow_NR N]Ti] ._. S]

The following constrained rules are extracted from this example:

(NN) TO VB (NPT) -> Vi
(IN) VBG (DTI) ->Vg
(NNS) BEZ (TO) -> V
(INO) NN NPTS (NR) -> N
(VB) NPT NP (IN) -> N
(VBG) DTI AP NN NN NNS (BEZ) -> N
(BEZ) TO BE VBN (IN) -> Vi
(NPTS) NR (.) -> N

Furthermore, the same constrained rules are grouped into one. Under this way, total 20,002 constrained rules are extracted from the Lancaster Parsed Corpus. All the constrained rules are examined and 219 conflict rules are found. The conflicts result from the inconsistent annotations in the corpus. Some conflict rules are listed below. The number enclosed in the parentheses denotes the frequency of the rule.

( NNS ) VB ( ATI ) -> V (6), Vr (1)
( NNS ) VBN ( . ) -> Vn (10), Vr (1)
( NNS ) VBN ( RB ) -> Vn (4), V (1)
( NNS ) VB ( IN ) -> V (24), Vr (1)
( NNS ) VBN ( IN ) -> Vn (54), V (3)

In the above samples, ( NNS ) VB ( ATI ) -> V (6), Vr (1), means that VB can be raised to V (Vr) with frequency 6 (1). To avoid this inconsistency, some rules having lower frequencies are deleted. Finally, a decision tree is used to model the remaining unconflict rules. Fig. 4 shows the decision tree for the following rules:

( DT ) BEDZ ( WDT ) -> V
( DT ) BEZ ( , ) -> V
( DT ) BEDZ ( WRB ) -> V
( DT ) BEZ ( ABL ) -> V
( DT ) BEG ( PN ) -> Vg
( DT ) BEZ ( ABN ) -> V

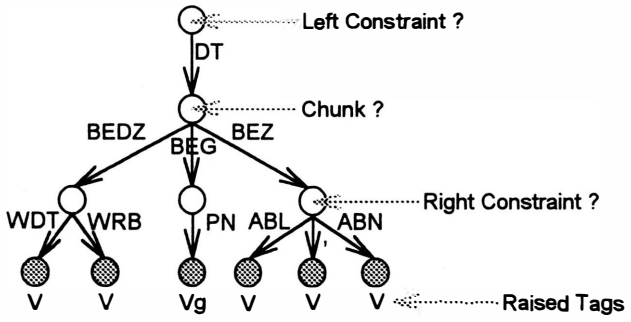



**Fig. 4. The Decision Tree**

A rule can be applied when its left constraints, chunks and its right constraints are satisfied. That is, a path is found in the decision tree.

## 5 The Partial Parsing Algorithm

The partial parsing algorithm based on Constrained Grammar is proposed below.

```
Partial_Parser(Tag_Sequence)
Begin
    C_Position=1;
    While C_Position<=N Do
    Begin
        Find=0;
        For Chunk_Length=8, ..., 1 Do
        Begin
            If (C_Position+Chunk_Length-1)<=N Then
            Begin
                If Search Decision Tree for
                  Tag_Sequence[C_Position-1] as Left Constraint,
                  Tag_Sequence[C_Position~(C_Position+Chunk_Length-1)] as Chunk and
                  Tag_Sequence[C_Position+J] as Right Constraint
                  Is Successful Then
                Begin
                    Output "[";
                    Output Raised Tag;
                    For Position=C_Position, ..., (C_Position+Chunk_Length-1) Do
                        Output Tag_Sequence[Position];
                    Output Raised Tag;
                    Output "]";
                    Find=1;
                    Goto Done;
                End
            End
        End

Done:   If Find=1 Then C_Position=C_Position+Chunk_Length-1;
        Else Output Tag_Sequence[C_Position];
        C_Position=C_Position+1;
    End
End
```

Variable *Find* denotes whether a chunk is found in the decision tree or not and Variable *C_Position* means current position. Assume that the input sentence contains N words, and the symbol * is added to the beginning position (0) and the ending position (N+1) to facilitate the process. The processes for N=6, C_Position=1 and Chunk_Length=8 (7 and 6) are shown in Fig. 5.

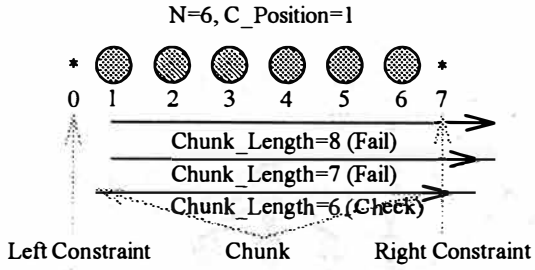



**Fig. 5. The Processes**

Because the length of the largest chunk in the training corpus is 8 and the larger chunks are preferred, the algorithm checks the chunks from length 8 to 1.

## 6 The Experimental Results

The performance evaluation model compares the chunking-and-raising result P' with the corresponding syntactic structure T. The evaluation criterion is to count how many tags are assigned correctly. For example, there is a parsing tree - say, [ A [ B [ C W1_P1 W2_P2 C ] W3_P3 [ D W4_P4 D ] B ] [ E W5_P5 W6_P6 E ] A ]. If the parsing result is [ F W1_P1 W2_P2 F ] [ G W3_P3 G ] W4_P4 [ E W5_P5 W6_P6 E ], then 2 tags, i.e., P5 and P6, are assigned correctly. The tags P1 and P2 are wrong because the raised tag is wrong, i.e., it must be C. The tag P3 is wrong because P3 cannot be a chunk. Similarly, the tag P4 is wrong because it must be chunked and raised to D. According to this criterion, the experimental results are shown in Table 2.

**Table 2. The Experimental results for Inside Test**

| Category | Correct | Wrong | Total | Correct Rate (%) |
|---|---|---|---|---|
| A | 7792 | 492 | 8284 | 94.06% |
| B | 8839 | 553 | 9392 | 94.11% |
| C | 7183 | 555 | 7738 | 92.83% |
| D | 9137 | 611 | 9748 | 93.73% |
| E | 8658 | 614 | 9272 | 93.38% |
| F | 7810 | 562 | 8372 | 93.29% |
| G | 5794 | 472 | 6266 | 92.47% |
| H | 5872 | 652 | 6524 | 90.01% |
| J | 7600 | 711 | 8311 | 91.45% |
| K | 10338 | 463 | 10801 | 95.71% |
| L | 11394 | 596 | 11990 | 95.03% |
| M | 6958 | 369 | 7327 | 94.96% |
| N | 11147 | 506 | 11653 | 95.66% |
| P | 11549 | 548 | 12097 | 95.47% |
| R | 7878 | 502 | 8380 | 94.01% |
| Total | 127949 | 8206 | 136155 | 93.97% |

If the inconsistency problem of the corpus does not occur, the performance can be better. When we remove one file from training corpus and use this file as the testing corpus, the experimental results are listed in Table 3.

**Table 3. The Experimental results for Outside Test**

| Category | Correct | Wrong | Total | Correct Rate (%) |
|---|---|---|---|---|
| K | 8324 | 2477 | 10801 | 77.07% |
| P | 9366 | 2731 | 12097 | 77.42% |
| N | 9166 | 2487 | 11653 | 78.66% |

In these experiments, K, P or N are removed from training corpus. The performance is decreased. It reveals that the training corpus is still not large enough. Structure Mapping between different treebanks [19] provides a feasible way to obtain a larger corpus. In this way, much more reliable

statistic information can be trained from the large-scale treebanks, so that the feasibility of the parser is assured.

## 7 Related Works

Chen and Chen [20] propose a probabilistic chunker to decide the implicit boundaries of constituents and utilize the linguistic knowledge to extract the noun phrases by a finite state mechanism. Rather than using a treebank as a training corpus, Chen and Lee [21] also propose a probabilistic chunker based on parts-of-speech information only. However, the evaluation adopted in these two papers is not very strict. Consider the following parsed sentence, which is extracted from Susanne Corpus.

[ S [ Nns:s The_ATI [ Nns Fulton_NP County_NPL Nns ] Grand_JJ Jury_NN Nns:s ] [ Vd said_VBD Vd ] [Nns:t Friday_NR Nns:t ] [ Fn:o [ Ns:s an_AT investigation_NN [ Po of_IN [ Ns [ G Atlanta's_NP$ G ] recent_JJ primary_JJ election_NN Ns ] Po ] Ns:s ] [ Vd produced_VBD Vd ] [ Ns:o +no_ATI evidence_NN [ Fn that_CS [ Np:s any_DTI irregularities_NNS Np:s ] [ Vd took_VBD Vd ] [Ns:o place_NPL Ns:o ] Fn ] Ns:o ] Fn:o ] S ]

By the method proposed by Chen and Chen [20], the result is shown as follows.

[ The_ATI Fulton_NP County_NPL ] [ Grand_JJ Jury_NN ] [ said_VBD ] [ Friday_NR ] [ an_AT investigation_NN ] [ of_IN Atlanta's_NP$ ] [ recent_JJ primary_JJ election_NN ] [ produced_VBD ] [ +no_ATI evidence_NN ] [ that_CS any_DTI irregularities_NNS ] [ took_VBD ] [ place_NPL ]

By their evaluation criterion, only chunk [ of_IN Atlanta's_NP$ ] is wrong. But, it is clear that some chunks are wrong. By our criterion, the correct output should be:

The_ATI [ **Nns** Fulton_NP County_NPL **Nns** ] Grand_JJ Jury_NN [ **Vd** said_VBD **Vd** ] [ **Nns:t** Friday_NR **Nns:t** ] an_AT investigation_NN of_IN [ **G** Atlanta's_NP$ **G** ] recent_JJ primary_JJ election_NN [ **Vd** produced_VBD **Vd** ] +no_ATI evidence_NN that_CS [ **Np:s** any_DTI irregularities_NNS **Np:s** ] [ **Vd** took_VBD **Vd** ] [ **Ns:o** place_NPL **Ns:o** ]

The key issue is: when the chunked results are erroneous on the lowest level, the effects will be propagated to the upper level. Besides, the interpretation of chunks is another problem. Consider a sequence of chunks, i.e., [ A ] [ B C ] [ D ]. There may be at least two possible interpretations shown in Figs. 6 and 7. That makes the chunker difficult to scale up to a full parser.

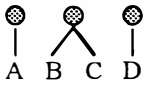



**Fig. 6. Interpretation 1**

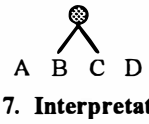



**Fig. 7. Interpretation 2**

## 8 Concluding Remarks

This paper proposes a linear-time partial parser. It is a simple version of a chunking-and-raising parser, but it can be extended to a full parser easily by performing more chunking and raising actions. Basically, the Constrained Grammar is provided to each level of the chunking-and-raising parser. Because each rule in the Constrained Grammar has left and right constraints, the grammar is different from the LL(k) grammar although they have the similar concepts, i.e., left to right scanning and lookahead. In contrast to the Inside-Outside optimization algorithm [5] which is very computationally intensive, this kind of parser is very simple but effective. The short-term goal of this research is to help the development of a partially bracketed corpus, i.e., a simpler version of a treebank. The long-term goal is to provide high level linguistic constraints for many natural language applications.

## References


[1] A. Corazza, *et al.*, "Stochastic Context-Free Grammars for Island-Driven Probabilistic Parsing," *Proceedings of International Workshop on Parsing Technologies*, 1991, pp. 210-217.

[2] S.K. Ng and M. Tomita, "Probabilistic LR Parsing for General Context-Free Grammars," *Proceedings of International Workshop on Parsing Technologies*, 1991, pp. 154-163.

[3] D.M. Magerman and C. Weir, "Efficiency, Robustness and Accuracy in Picky Chart Parsing," *Proceedings of ACL*, 1992, pp. 40-47.

[4] R. Bod, "Using an Annotated Corpus as a Stochastic Grammar," *Proceedings of EACL*, 1993, pp. 37-44.

[5] Y. Schabes, M. Roth and R. Osborne, "Parsing the Wall Street Journal with the Inside-Outside Algorithm," *Proceedings of EACL*, 1993, pp. 341-347.

[6] J. Dowding, R. Moore, F. Andry and D. Moran, "Interleaving Syntax and Semantics in an Efficient Bottom-Up Parser," *Proceedings of ACL*, 1994, pp. 110-116.

[7] D. Tugwell, "A State-Transition Grammar for Data-Oriented Parsing," *Proceedings of EACL*, 1995, pp. 272-277.

[8] D.D. McDonald, "An Efficient Chart-Based Algorithm for Partial Parsing of Unrestricted Texts," *Proceedings of Applied Natural Language Processing*, 1992, pp. 193-200.

[9] C. Jacquemin, "Recycling Terms into a Partial Parser," *Proceedings of Applied Natural Language Processing*, 1994, pp. 113-118.

[10] C. Lyon and B. Dickerson, "A Fast Partial Parse of Natural Language Sentences Using a Connectionist Method," *Proceedings of EACL*, 1995, pp. 215-222.

[11] J. Vergne. "A Non-Recursive Sentence Segmentation, Applied to Parsing of Linear Complexity in Time." *Proceedings of NeMLaP*, 1994, pp. 234-241.

[12] K.W. Church, "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text," *Proceedings of Applied Natural Language Processing*, 1988, pp. 136-143.

[13] S.J. DeRose, "Grammatical Category Disambiguation by Statistical Optimization," *Computational Linguistics*, 1988, 14(1), pp. 31-39.

[14] E. Brill, "A Simple Rule-Based Part-of-Speech Tagger," *Proceedings of Applied Natural Language Processing*, 1992, pp. 152-155.

[15] D. Cutting, *et al.*, "A Practical Part-of-Speech Tagger," *Proceedings of Applied Natural Language Processing*, 1992, pp. 133-140.

[16] D. Elworthy, "Does Baum-Welch Re-Estimation Help Taggers?" *Proceedings of Applied Natural Language Processing*, 1994, pp. 53-58.

[17] B. Merialds, "Tagging English Text with a Probabilistic Model," *Computational Linguistics*, 1994, 20(2), pp. 155-171.

[18] P. Tapanainen and A. Voutilainen, "Tagging Accurately - Do'nt Guess If You Know," *Proceedings of Applied Natural Language Processing*, 1994, pp. 47-52.

[19] E. Atwell, *et al.*, "AMALGAM: Automatic Mapping Among Lexico-Grammatical Annotation Models," *Proceedings of the Balancing Act - Combining Symbolic and Statistical Approaches to Language*, 1994, pp. 11-20.

[20] K.H. Chen and H.H. Chen, "Extracting Noun Phrases from Large-Scale Texts: A Hybrid Approach and its Automatic Evaluation," *Proceedings of ACL*, 1994, pp. 234-241.

[21] H.H. Chen and Y.S. Lee, "Development of Partially Bracketed Corpus with Part-of-Speech Information Only," *Proceedings of Workshop on Very Large Corpora*, 1995.

# Distributed Parsing With HPSG Grammars*

**Abdel Kader Diagne, Walter Kasper, Hans-Ulrich Krieger**
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{diagne,kasper,krieger}@dfki.uni-sb.de

## 1 Introduction

A fundamental concept of Head-Driven Phrase Structure Grammar (HPSG: [PS87, PS94]) is the notion of a SIGN. A SIGN is a structure integrating information from all levels of linguistic analysis such as phonology, syntax and semantics. This structure also specifies interactions between these levels by means of coreferences which indicate the sharing of information and how the levels constrain each other mutually. Such a concept of linguistic description is attractive for several reasons:

- it supports the use of common formalisms and data structures on all levels of linguistics
- it provides declarative and reversible interface specifications between the levels
- all information is available simultaneously
- no procedural interaction between linguistic modules needs to be set up

Similar approaches especially for the syntax-semantics interface have been suggested for all major unification-based theories of grammar, such as LFG or CUG. For these theories and their underlying formalisms it was shown how to provide at least partial and underspecified semantic descriptions in parallel to syntax. [HK88] call such approaches *codescriptive* in contrast to the approach of *description by analysis* which is closely related to sequential architectures where linguistic levels correspond to components which operate on the basis of the (complete) analysis results of lower levels.

Unification-based theories of grammar are expressed in feature-structure formalisms by equational constraints. Semantic descriptions are expressed there by additional constraints.

Though theoretically very attractive, codescription has its price: The grammar is difficult to modularize and there is a computational overhead when parsers use the complete descriptions.

Problems of these kinds which were already noted by [Shi85] motivated the research described here. The goal is to develop more flexible ways of using codescriptive grammars than having them applied by a parser with full informational power. The underlying observation is that constraints in such grammars can play different roles:

- "genuine" constraints which take effect as filters on the input. These relate directly to the grammaticality (wellformedness) of the input. Typically, these are the syntactic constraints.
- "spurious" constraints which basically build representational structures. These are less concerned with wellformedness of the input but rather of the output for other components in the overall system. Much of semantic descriptions is of this kind.

*This work was funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verbmobil Project under Grant *01 IV 101 K/1*. The responsibility for the contents of this study lies with the authors. We are grateful to three anonymous IWPT referees.

If the parser generated from such a grammar specification treats all constraints on a par it cannot distinguish between the structure building and the analytical constraints. Since unification based formalisms are monotonic, large structures are built up and have to undergo all the steps of unification, copying and undoing in the processor. The cost of these operations (in time and space) increase exponentially with the size of the structures.

In the VERBMOBIL project ([Wah93, KGN94]) the grammar parser is used in the context of a speech translation system. The parser input consists of word lattices of hypotheses from speech recognition. The parser has to identify those paths in the lattice which represent a grammatically acceptable utterance. Parser and recognizer are incremental and interactively running in parallel. Even for short utterances the lattices can contain several hundred of word hypotheses and paths most of which are not acceptable grammatically.

The basic idea presented here is to distribute the labour of evaluating the constraints in the grammar on several processes. Important considerations in the design of the system were:

- increase in performance
- maintenance of an incremental and interactive architecture of the system[1]
- minimize the overhead in communication between the processors

Several problems must be solved for such a system. First, it must be able to work with partial (incomplete) analyses. Also, synchronization of the processors for the exchange of information about success and failure in analysis is necessary.

In the following sections we will discuss these constraints in more detail. After that we will describe the communication protocol between the parsing processes. Then several options for creating subgrammars from the complete grammar will be discussed. The subgrammars represent the distribution of information across the parsers. Finally, some experimental results will be reported.

We used a mid-size German grammar written in the typed-feature formalism $\mathcal{TDL}$ ([KS94]) which covers dialogs collected in VERBMOBIL. In this system *principles* of HPSG are defined as types which are inherited by the grammar rules. The grammar cospecifies syntax and semantics in the attributes SYN and SEM. To facilitate experimentation with distributed processing a slightly unconventional SIGN-structure was chosen. Additionally to the SYN- and SEM attributes for syntactic/semantic descriptions some features such as SUBCAT were singled out as control structures for the derivation processes which are shared by the subgrammars. Furthermore, unique identifiers for grammar rules and lexical entries had to be provided for the communication between the parsers, as will be explained below.

# 2 The Architecture

The most important aspect for the distribution of analysis and for defining modes of interaction between the analysis processes (parsers) was that one of the processes was to work as a filter on the word lattices reducing the search space. The other component then would work only with successful analysis results of the other one. This means that the two parsers would not run really in parallel on the input word lattices. Rather one parser would be in control over the second one which would not be exposed directly to the word lattices. For reasons which will become obvious below we will call the first of these parsers the SYN-*parser*, the second one controlled by the SYN-parser, the SEM-*parser*.

Another consideration to be taken into account is that the analysis should be *incremental* and *time-synchronous*. For the interaction of the parsers this implied that the SYN-parser must not send its results only when it is completely finished with its analysis, forcing the SEM-parser to wait.[2]

[1] The *system* alluded to here and below which provided the context of this work, was the INTARC-II-prototype of VERBMOBIL which was officially presented in April 1995.

[2] Another problem in incremental processing is that it is not known in advance when the utterance is finished or a new utterance starts. To deal with this, prosodic information is taken into account. This will not discussed here.

*Interactivity* is another aspect we had to consider. The SEM-parser must be able to report back to the SYN-parser at least when its hypotheses failed. This would not be possible when the SEM-parser has to wait till the SYN-parser is finished. This requirement also constraints the exchange of messages.

Incrementality and interactivity imply a steady exchange of messages between the parsers. An important consideration then is that the overhead for this communication does not outweigh the gains of distributed processing. This consideration rules out that the parsers should communicate by exchanging their analysis results in terms of resulting feature structures. There are no good ways of communicating feature structures across distinct processes except as (large) strings. This means that the parsers would need the possibility to build strings from feature structures and parse strings into feature structures. Also, on each communication event the parsers would have to analyze the structures to detect changes, whether a structure is part of other already known structures etc. It is hard to see how this kind of communication can be interleaved with normal parsing activity in efficient ways.

In contrast to this, our approach allows to exploit the fact that the grammars employed by the parsers are derived from the same grammar and thereby similar in structure. This makes it possible to restrict the communication between the parsers to information about what rules were successfully or unsuccessfully applied. Each parser then can reconstruct on his side the state the other parser is in: how its chart or analysis tree looks like. Both parsers try to maintain or arrive at isomorphic charts.

The approach allows that the parsers never need to exchange analysis results in terms of structures as the parsers should always be able to reconstruct these if necessary. On the other hand, this reconstructibility poses constraints on how the codescriptive grammar can be split up in subgrammars.

The requirements of incrementality, interactivity and efficient communication show that our approach does not emulate the *description by analysis*-methodology in syntax-semantics interfaces on the basis of codescriptive grammars.

# 3 The Parsers

The SYN-parser and the SEM-parser are agenda driven chart parsers. For speech parsing the nodes represent points of times and edges represent word hypotheses and paths in the word lattice.

The parsers communicate by exchanging *hypotheses*, bottom-up hypotheses from syntax to semantics and top-down hypotheses from semantics to syntax.

- **Bottom-up hypotheses** are emitted by the SYN-parser and sent to the SEM-parser. They undergo verification at the semantic level. A bottom-up hypothesis describes a passive edge (complete subtree) constructed by the syntax parser and consists of the identifier of the rule instantiation that represents the edge and the *completion history* of the constructed passive edge. Having passive status is a necessary but not sufficient condition for an edge to be sent as hypothesis. Whether a hypothesis is sent depends also on other criteria such as its score. In the actual system the SYN-parser is a probabilistic chart parser using a statistic language model as additional knowledge source ([HW94]).

- **Top-Down hypotheses** result from activities of SEM-parser trying to verify bottom-up-hypotheses. To keep the communication efforts low only failures are reported back to the SYN-parser by sending simply the hypothesis' identifier. This should start a chart revision process on the SYN-parser's side.[3]

The central data structure by which synchronization and communication between the parsers is achieved is that of a **completion history** containing a record on how a subtree was completed. Basically it tells us for each edge in the chart which other edges are spanned.

The nodes in the chart correspond to points in time and edges to time intervals spanned. In contrast to standard chart parsing *gaps* may occur between two consecutive edges. The relaxation of this constraint helps to reduce the number of bottom-up-hypotheses.

[3]This revision process is currently under investigation.

Completion histories are described by the following BNF:

```
edge_id := "E" INTEGER
rule_id := "R"INTEGER
node_id := "N" INTEGER
edge_list := rule_id ((rule_id node_id node_id) | edge_id)*
completion_history := (edge_list M)*
```

E, R, N and M are delimiters and INTEGER is an integer used as identifier. E marks an identifier for a subtree which has already been sent as hypothesis (hypothesis' identifier) to SEM-parser, R an identifier for the rule used to build the subtree, N the nodes in the subtree and M delimits a list of edges representing the completion history of a completed part of the current edge. The following example illustrates a complex completion history:

```
R 1   R 10052 N 0 N 1   R 10032 N 1 N 2   M
R 4   R 10275 N 2 N 3   R 10311 N 3 N 4   M
R 2   R 1 N 0 N 2   R 4 N 2 N 4   M
R 0   R 2 N 0 N 4   E 1   M
```

This protocol allows the parsers efficiently to exchange information about the structure of their chart without having to deal with explicit analysis results as feature structures.

Since the SEM-parser does not work directly on linguistic input but is fed by the SYN-parser there are some differences to ordinary chart-parsing. The SEM-parser uses a *two-level agenda-mechanism.* The low-level agenda manages the bottom-up hypotheses from the syntax. It is currently a queue, that is, hypotheses are treated in a FIFO manner. The high-level agenda is an agenda as known from chart parsers with scanning, prediction and combination tasks. What is special here is that the structure of the high-level-agenda is guided mainly by the low-level-agenda. Since the SEM-parser is controlled by the SYN-parser there are two parsing modes:

- *non-autonomous parsing:* The parsing process consists mainly of constructing the tree described by the completion history by using the semantic counterparts of the rules which led to the syntactic hypothesis. If this fails because of semantic constraints this is reported back to the SYN-parser.
- *quasi-autonomous parsing:* If no syntactic hypotheses are present the parser extends the chart on its own using its rules by completion and prediction steps. Obviously, this is only possible after some initial information by the SYN-parser, since the SEM-parser is not directly connected to the input utterance.

We ignore here that SYN-parser and SEM-parser also receive hypotheses from prosody about phrase boundaries and utterance mood which also influence the parsing process.

# 4 Compilation of Subgrammars

In the following sections we discuss possible options and problems for the distribution of information in a cospecifying grammar. In our approach the question arises in the form that we have to specify which of the parsers uses what information. This set of information is what we call a *subgrammar.* These subgrammars are generated from a common source grammar.

## 4.1 Reducing Representational Overhead by Separation of Syntax and Semantics

An obvious choice for splitting up the grammar was to separate the linguistic levels (strata), such as syntax and semantics. This choice was also motivated by the observation that typically the most important constraints on grammaticality of the input are in the syntactic part while most of the semantics was purely representational.[4] A straightforward way to achieve this is

[4]This must be taken *cum grano salis* as it depends on how a specific grammar draws the line between syntax and semantics: selectional constraints, e.g. for verb arguments, typically are part of semantics and are true constraints. Also, semantic constraints would have a much larger impact if, for instance, agreement constraints would be considered as semantic, too, as [PS94] suggest.

by manipulating grammar rules and lexicon entries: for the SYN-parser, we delete recursively the information under the SEM attributes and similarly clear the SYN attributes to obtain the subgrammar for the SEM-parser. We abbreviate these subgrammars by $G_{syn}$ and $G_{sem}$ and the original grammar by $G$.

This methodology reduces the size of the structures for the SYN-parser for lexical entries to about 30% of the complete structure. One disadvantage of this simple approach is that coreferences between the linguistic levels *syntax* and *semantics*—which will be referred to as the *coreference skeleton*—disappear. This might lead to several problems which we address in Section 4.2. Sections 4.3 then discusses possible solutions.

Another, more sophisticated way to keep the structures small is due to the type expansion mechanism in TDL ([KS95]). Instead of destructively modifying the feature structures beforehand, we can employ type expansion to let SYN or SEM unexpanded. This has the desired effect that we do not lose the coreference constraints and furthermore are free to expand parts of the feature structure afterwards. We will discuss this feature in Section 4.4. In the actual system this option was not available as its SYN-parser ([HW94]) employed a simpler formalism which does not provide type expansion. Therefore the TDL (sub-)grammar had to be expanded beforehand in order to transform the structures to that formalism.

## 4.2 Problems

Obviously, the biggest advantage of our method is that unification and copying becomes faster during processing, due to the smaller structures. We can even estimate the speed-up in the best case, viz., quasi-linear w.r.t. input structure if only conjunctive structures are used. Clearly, if many disjunctions are involved, the speed-up might even be exponential.

However, the most important disadvantage of the compilation method is that it no longer guarantees *correctness*, that is, the subgrammar(s) might accept utterances which are ruled out by the full grammar. This is due to the simple fact that certain constraints are neglected in the subgrammars. If at least one such constraint is a filtering constraint, we automatically enlarge the language accepted by this subgrammar w.r.t. complete grammar. Clearly, *completeness* is not affected, since we do not add further constraints to the subgrammars.

At this point, let us focus on the estimation above, since it is only a best-case forecast. Sure, the structures become smaller; however, due to the possible decrease of filter constraints, we must expect an increase of hypotheses in the parser. And in fact, our experimental results in Section 5 show that our approach has a different impact on the SYN-parser and the SEM-parser (see Figure 1). Our hope here, however, is that the increase of non-determinism inside the parser is compensated by the processing of smaller structures (see [MK91] for more arguments on this theme).

In general, even the intersection of the languages accepted by $G_{syn}$ and $G_{sem}$ does not yield the language accepted by $G$, but only the weaker relation $\mathcal{L}(G) \subset \mathcal{L}(G_{syn}) \cap \mathcal{L}(G_{sem})$ holds.

This behaviour is an outcome of our compilation schema, namely, cutting reentrancy points. Thus, even if an utterance $S$ is accepted by $G$ with analysis $fs$ (feature structure), we can be sure that the unification of the corresponding results for $G_{syn}$ and $G_{syn}$ will subsume $fs$: $fs \preceq fs_{syn} \wedge fs_{sem}$

Let us mention further problems. First, *termination properties* might change in case of the subgrammars. Consider a subgrammar which contains empty productions or unary rules. Assume that such rules were previously "controlled" by constraints that are no longer present. Obviously, if a parser is not restricted through additional, non-grammatical constraints, the iterated application of these rules could lead to an infinite computation, i.e., a loop. This was sometimes the case during our experiments.

Second, recursive rules could introduce infinitely many solutions for a given utterance. Theoretically, this might not pose a problem, since the intersection of two infinite sets of parse trees might be finite. However in practice this problem is hard to avoid.

## 4.3 Solutions

In this section, we will discuss three solution to the problems mentioned in the last section.

**Feedback Loop.** Although semantics construction is driven by the speech parser, the use of different subgrammars suggest that the speech parser should also be guided by the SEM-parser. This can be achieved by sending back *falsified* hypotheses. Because hypotheses are uniquely identified in our framework, we must only send the integer that identifies the falsified chart edge. However, such an approach presumes that the SYN-parser is able to revise its chart (cf. [Wir92]). This idea is currently under implementation.

**Coref Skeleton.** In order to guarantee correctness of the analysis, we might unify the results of both parsers with the corresponding coref skeletons at the end. This strategy will not be pursued here since it introduces an additional processing step during parsing. It would be better to employ type expansion here in order to let SYN or SEM unexpanded so that coreferences can be preserved. Exactly this treatment will be investigated in the next section.

**Full-Size Grammar** The most straightforward way to guarantee correctness is simply by employing the full-size grammar in one of the two parsers. This might sound strange, but recall that we process speech input so that even a small grammar constrains possible word hypotheses. We suggest that the SEM-parser should operate on the full-size grammar since the speech parser directly communicates with the word recognizer and must process an order of magnitude more hypotheses than the SEM-parser. Because the SEM-parser passes its analysis on to the semantic evaluation module, it makes further sense to guarantee correctness here. This has been the final set-up during our experiments.

## 4.4 Improvements

This section investigates several improvements of our compilation approach which solve the problems mentioned before.

**Identifying Functional Strata Manually.** Normally, the grammarian "knows" which information needs to be made explicit. Hence, instead of differentiating between the linguistic strata SYN and SEM, we let the linguist identify which constraints filter and which only serve as a means for representation (cf. [Shi85]). In contrast to the separation along linguistic levels this approach adopts a functional view cutting across linguistic strata. On this view, the syntactic constraints together with e.g. semantic selection constraints would constitute a subgrammar.

**Bookkeeping Unifications.** A semi-automatic way to determine true constraints w.r.t. a training corpus is simply by bookkeeping feature unification. Features that occur only once on top of the input feature structures do not specialize the information in the resulting structure (actually the values of these features). Furthermore, features typed to $\top$ (top; = [ ]) do not constrain the result. For instance

$$\begin{bmatrix} \mathrm{A}\ s \\ \mathrm{B}\ \begin{bmatrix} t \\ \mathrm{D}\ w \end{bmatrix} \\ \mathrm{C}\ u \end{bmatrix} \wedge \begin{bmatrix} \mathrm{A}\ v \\ \mathrm{B}\ \top \end{bmatrix} = \begin{bmatrix} \mathrm{A}\ s \wedge v \\ \mathrm{B}\ \begin{bmatrix} t \\ \mathrm{D}\ w \end{bmatrix} \\ \mathrm{C}\ u \end{bmatrix}$$

This unification indicates that only the path A needs to be made explicit, since its value is more specific than the corresponding input values: $s \wedge v \preceq s$ and $s \wedge v \preceq v$.

**Partial Evaluation.** Partial evaluation, as known from logic programming ([War92]), is a method of carrying out part of the computation at compile time that would otherwise normally be done at run time, hence improving run time performance of logic programs. Analogous to partial evaluation of definite clauses, we can partially evaluate annotated grammar rules, since they drive the derivation. Take for instance the following grammar rule in $\mathcal{TDL}$:

```
np-det-rule :=
  max-head-1-rule --> < det-cat-type, n-bar-cat-type >.
```

`max-head-1-rule`, `det-cat-type`, and `n-bar-cat-type` are types which participate in type inheritance and abbreviate complex typed feature structure. Partial evaluation means here to substitute type symbols through their expanded definitions.

Because a grammar contains finitely many rules of the above form and because the daughters (the right hand side of the rule) are type symbols (and there are only finitely many of them), this partial evaluation process can be performed off-line. Now, only those features are made

| number of sentences: 56 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| average length: 7.6 | | | | | | | | | |
| | **SynSem** | **Syn** | | **Sem** | | **Sem-I** | | **SynSem-I** | |
| | | | % | | % | | % | | % |
| run time: | 30.6 | 15.2 | 50 | 45.8 | 150 | 51.8 | 169 | 32.1 | 105 |
| #readings: | 1.7 | 2.1 | 123 | 1.8 | 105 | 4.2 | 247 | 3.8 | 224 |
| #hypotheses: | 53.0 | 58.1 | 110 | 81.3 | 153 | *4.2 | | *4.6 | |
| #chart edges: | 192.0 | 215.0 | 112 | 301.1 | 156 | 361.7 | 188 | 227.7 | 119 |

Figure 1: Experimental results of SYN/SEM separation. The two last columns indicate the performance in incremental mode. * indicates the number of top-down hypotheses which are sent back by the feedback loop to the SYN-parser (see Section 4.3). The percentage values are relative to **SynSem**.

explicit which actively participate in unification during partial evaluation. In contrast to the previous method, partial evaluation is corpus-independent.

**Type Expansion.** We have indicated earlier that type expansion can be fruitfully employed to preserve the coref skeleton. Type expansion can also be chosen to expand parts of a feature structure on the fly at run time.

The general idea here is as follows. Guaranteeing that the lexicon entries and the rules are consistent, we let everything unexpanded unless we are enforced to make structure explicit. As was the case for the previous two strategies, this is only necessary if a path is introduced in the resulting structure which value is more specific than the value(s) in the input structure(s).

The biggest advantage of this approach is obvious—only those constraints must be touched which are involved in restricting the set of possible solutions. Clearly, such a test should be done every time the chart is extended. The cost of such tests and the on-line type expansions need further investigation.

# 5 Experimental Results

This section presents experimental results (Fig. 1) of our compilation method which indicate that the simple SYN/SEM separation does not match the distinction between true and spurious constraints. The measurements have been obtained w.r.t. a corpus of 56 sentences from 4 dialogs. For the measurements we used as SYN-parser a simple bottom-up chart-parser. Also, no language model was used, nor other information from acoustics.

The column **Syn** shows that parsing with syntax only takes 50% of the time of parsing with the complete grammar (**SynSem**). The number of readings, hypotheses, and chart edges only slightly increase here. Surprisingly however, by employing $G_{sem}$ only, run time efficiency decreases massively (**Sem**: 150%), due to the increase in the number of hypotheses (153%). This indicates that most of the filtering constraints are specified in the syntax. Consequently, the incremental version of semantics construction (**Sem-I**) is *in total* even more worse, due to the incremental behaviour of the SEM-parser, namely, to create readings as early as possible and to pass these results immediately on. **SynSem-I** depicts the measurements for the incremental version of the SEM-parser with the full-size grammar $G$.

The apparent increase in the number of readings in the incremental mode is a bit misleading. Since in absence of information about the length of the utterance in incremental mode the parser used as sole criterion for a reading that there is an edge from the start to the current point of time which represents a maximal, saturated sign. This means for instance that it will always emit the sentence topic in verb-second sentences as a separate reading. Also, if a sentence ends with a sequence of free adjuncts, the parser will assume as many readings as there are adjuncts. So a sentence like *I have a meeting on monday with John* in incremental mode will get the following sequence of partial pseudo-"readings": *I*, *I have a meeting*, *I have a meeting on monday*, *I have a meeting on monday with John*. In general, there is only a small overhead in incremental mode because we are operating over a chart which allows to reuse the parts already analyzed.

In the context of the INTARC-II system with word lattice parsing the combination of running the SYN-parser with the syntactic part of the grammar on the word lattices and running the SEM-parser with the full grammar (**Synsem-I**) proved to be quite efficient. That the SEM-parser has to deal with larger analysis structures in this setup was by far made up for by the fact that it had much lesser hypotheses to evaluate than the SYN-parser on the word lattice.

# 6 Conclusions

Linguistic theories like HPSG provide an integrated view on linguistic objects by providing a framework with a uniform formalism for all levels of linguistic analysis. All information is integrated in a single information structure, the SIGN . Though attractive from a theoretical point of view, it raises questions of computational tractability. We subscribe to that integrated view on the level of linguistic descriptions and specifications. On the other hand, from a computational view, we think that for special tasks not all that information is useful or required, at least not all at the same time.

In this paper we described first attempts to make a more flexible use of integrated linguistic descriptions by splitting them up into subpackages that are handled by special processors. We also devised an efficient protocol for communication between such processors. First results have been encouraging. On the other hand, we addressed a number of problems and possible solutions. Only further research can show which one to prefer.

# References


[HK88] Per-Kristian Halvorsen and Ronald M. Kaplan. Projections and semantic description in lexical-functional grammar. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 1116–1122, Tokyo, 1988.

[HW94] Andreas Hauenstein and Hans Weber. An investigation of tightly coupled time synchronous speech language interfaces using a unification grammar. Verbmobil-Report 9, Universität Erlangen/Universität Hamburg, 1994.

[KGN94] Martin Kay, Jean Mark Gawron, and Peter Norvig. *Verbmobil. A Translation System for Face-to-Face Dialog*, volume 33 of *CSLI Lecture Notes*. Chicago University Press, 1994.

[KS94] Hans-Ulrich Krieger and Ulrich Schäfer. *TDL*—a type description language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics, COLING-94, Kyoto, Japan*, pages 893–899, 1994.

[KS95] Hans-Ulrich Krieger and Ulrich Schäfer. Efficient parameterizable type expansion for typed feature formalisms. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95, Montreal, Canada*, 1995. To appear.

[MK91] John T. Maxwell III and Ronald M. Kaplan. The interface between phrasal and functional constraints. In Mike Rosner, C.J. Rupp, and Rod Johnson, editors, *Proceedings of the Workshop on Constraint Propagation, Linguistic Description, and Computation*, pages 105–120. Instituto Dalle Molle IDSIA, Lugano, 1991. Also in Computational Linguistics, Vol. 19, No. 4, 571–590, 1993.

[PS87] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics*. Vol. 1: Fundamentals, volume 13 of *CSLI Lecture Notes*. Stanford: CSLI, 1987.

[PS94] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press, 1994.

[Shi85] Stuart M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics, ACL-85*, pages 145–152, 1985.

[Wah93] Wolfgang Wahlster. Verbmobil: Übersetzung von Verhandlungsdialogen. Verbmobil-Report 1, DFKI, Saarbrücken, 1993.

[War92] David S. Warren. Memoing for logic programs. *Communications of the ACM*, 35(3):93–111, March 1992.

[Wir92] Mats Wirén. *Studies in Incremental Natural-Language Analysis*. PhD thesis, Department of Computer and Information Science, Linköping University, 1992.

# Chart-based Incremental Semantics Construction with Anaphora Resolution Using λ-DRT

Ingrid Fischer, Bernd Geistert and Günther Görz
University of Erlangen-Nuremberg, IMMD II and VIII (Computer Science)
Am Weichselgarten 9, D-91058 ERLANGEN
Phone: (+49 9131) 85-9909; Fax: 85-9905
E-Mail: goerz@informatik.uni-erlangen.de

In our institute, a student project aiming at the construction of a prototypical natural language processing (NLP) system is being carried out. Its fundamental goal is the modular design and implementation of a simple but complete NLP system which is able to work incrementally and interactively. Due to its pedagogic character, particular attention is paid to a clear and transparent design of all modules in order to demonstrate the basic algorithms, even if this is not sufficient to achieve a high run time efficiency and a very broad linguistic coverage. The system, which is completely implemented in Scheme (SCM), is able to analyze short paragraphs of texts and to answer questions about it with a processing time of a few seconds per sentence. The text sort we chose are logical puzzles taken from the weekly published German newspaper "Die Zeit".

We present an approach for semantic construction and resolution based on an active chart-parser which analyzes syntax and semantics of natural language sentences incrementally. The parser consists of a syntactic component using a unification grammar formalism, an augmented variant of PATR-II, and a semantic component on the basis of Discourse Representation Theory (DRT) [Kamp, Reyle 1993]. In order to provide intersentential anaphora resolution, our incremental construction procedure builds discourse representations in which representations of consecutive sentences are embedded.

**Central Design Goals.** The design of our parsing system was governed by two main goals: On the one hand syntactic and semantic representations should be constructed *in parallel*, on the other hand input text should be analyzed *incrementally*, which is also extended to anaphora resolution. We chose a *co-descriptive approach* using a different formalism for each level.

**Incrementality.** A problem of any compositional semantics construction is that the order of construction steps may be disadvantageous for anaphora resolution, because, e.g., the construction for a supposed intrasentential antecedent would have been completed before the anaphoric NP is considered (c.f. [Johnson 1986]). Due to the incrementality of the parsing process it is guaranteed that information progresses from left to right.

**λ-DRT.** For the semantic formalism a combination of λ-calculus and DRT as developed by [Reyle 1986], [Asher 1993] and [Bos et al. 1994] was used. We adopted the last version, where DRSs are represented as feature structures. A DRS consists of a λ-list, the DRS and a special feature BDR whose value is a list of bindable discourse referents containing discourse referents of unsolved anaphora. Furthermore, we added features which are required for anaphora resolution: An ANCHOR list, an AGRM list, CONTEXT, and WEIGHT (see below). Functional composition of DRSs is realized by unifying parts of the functor and argument feature structures. But in contrast to Pinkal, where for functional composition only one element of the argument's λ-list is satisfied, we abandoned this restriction.

We redesigned the semantic formalism such that a parallel construction of syntactic and semantic structures can be achieved — structures on both levels are built concurrently as opposed to its sequential use in the VERBMOBIL project. There are two generic operators for semantic construction, identity (one-place) and compose (two-place) which are applied during the execution of the fundamental rule: Since in our case chart edges are annotated by categories, syntactic feature structures and λ-DRSs, the applicability of functional composition on the semantic level is an additional condition for the introduction of new edges.

Within this framework, different linguistic phenomena of German have been investigated: processing of proper names, tense (in the style of Reichenbach), negation, verbs as verb valency fillers, scope ambiguities with two newly introduced DRS-operators, plural, and prepositional phrases and other adjuncts.

**Resolution of Anaphora.** For inter- and intrasentential anaphora resolution, personal pronouns, reflexive pronouns and definite NPs are being considered, for which indefinite NPs and proper names are admissible antecedents. There are basically two means to achieve resolution: restrictions and preference weights (c.f. [Carbonell, Brown 1988]).

Many coreferences of anaphors and antecedents can be excluded by strict linguistic restrictions like gender-number agreement (using AGRM), binding principles, intersentential DRS accessibility constraints and semantic type constraints. In general, only antecedents within the local syntactic domain are candidates for a coreference of reflexive pronouns. For intersentential anaphors, DRS accessibility constraints are modeled by considering only the atomic conditions on the topmost DRS level of the DRSs of the past discourse as possible antecedents.

In establishing reference, anaphors are checked for simple semantic type consistency as soon as the semantic construction has been proceeded so far that it is obvious which valency slot of a verb will be filled by the NP under consideration.

Antecedents obeying those constraints are being scored by preferences, the calculation of which comprises the recency of anaphor and antecedent, syntactic parallelism in coordinated clauses, a simple focus mechanism, and topicalization preference.

References are resolved incrementally by considering inactive chart edges which correspond to anaphoric NPs. Inactive edges are suitable for insertion because they correspond to syntactically completely analyzed parts of the sentence. In the case of unresolved coreferences the respective discourse reference are inserted into the BDR list. If suitable antecedents can be found, for each of them a new inactive edge with a corresponding coreference entry, an empty BDR list, and a WEIGHT entry with the calculated scoring is built. In order to limit the growth of the number of edges, the next steps of the construction procedure are checking for violations (binding principles, semantic type consistency) which could not be recognized before.

Referential ambiguity is handled in the following way: The context entry of the structure from which the antecedent has been taken is inserted into the CONTEXT edge list of the intersentential anaphoric $\lambda$-DRSs. A new edge will not be generated if the functional composition of two $\lambda$-DRSs, which have been uniquely assigned as the discourse context, fails for the reason of incompatible CONTEXT entries.

Due to its incremental realization, the results of reference resolution are already available before the completion of semantic construction. By multiplying the WEIGHT entries, only the best scored results on the sentence level are used in the following steps.

# References


[Asher 1993] Asher, N.: *Reference to Abstract Objects in Discourse.* Dordrecht: Kluwer Academic Press, 1993

[Bos et al. 1994] Bos, J. et al.: *A Compositional DRS-based Formalism for NLP Applications.* Verbmobil Report 59, Saarbrücken 1994

[Carbonell, Brown 1988] Carbonell, J., Brown, R. D.: *Anaphora Resolution: A Multi-Strategy Approach.* In: Proceedings of the 12th Conference on Computational Linguistics, 1988

[Fischer 1993] Fischer, I.: *Die kompositionelle Bildung von Diskursrepräsentationsstrukturen über einer Chart.* Masters thesis. University of Erlangen-Nuremberg, IMMD VIII, 1993

[Geistert 1995] Geistert, B.: *Auflösung anaphorischer Referenzen in der $\lambda$-DRT.* Bachelors thesis, University of Erlangen-Nuremberg, IMMD VIII, 1995

[Johnson 1986] Johnson, M., Klein, E.: *Discourse, Anaphora and Parsing.* Proceedings of the 11th Conference on Computational Linguistics, 1986

[Kamp, Reyle 1993] Kamp, H., Reyle, U.: *From Discourse to Logic.* Dordrecht: Kluwer Academic Press, 1993

[Reyle 1986] Reyle, U.: *Zeit und Aspekt bei der Verarbeitung natürlicher Sprache.* PhD Thesis, University of Stuttgart, Institute of Linguistics, 1986

# Term Encoding of Typed Feature Structures

Dale Gerdemann
Seminar für Sprachwissenschaft, Universität Tübingen
Kl. Wilhelmstr. 113
72074 Tübingen, Germany
Email: dg@sfs.nphil.uni-tuebingen.de

## 1 Introduction

I explore in this paper a variety of approaches to Prolog term encoding typed feature structure grammars. As in Carpenter [3], the signature for such grammars consists of a bounded complete partial order of types under subsumption ($\sqsupseteq$) along with a partial function Approp: Type $\times$ Feat $\rightharpoonup$ Type. The appropriateness specification is subject to the constraint that feature-value specifications for subtypes are at least as specific as those for supertypes: if $Approp(\mathbf{t}, \text{F}) = \text{s}$ and $\mathbf{t}$ subsumes $\mathbf{t}'$, then $Approp(\mathbf{t}', \text{F}) = \text{s}'$ for some $\text{s}'$ subsumed by s.[1]

Previous approaches to term encoding of typed feature structures ([1], [2], [7]), have assumed a similar signature plus additional restrictions such as: limitations on multiple inheritance, or exclusion of more specific feature-value declarations on subtypes. The encoding presented here is subject to no such restrictions. The encoding will ensure that every feature structure is well-typed (Carpenter [3]), i.e., for every feature F on a node with type $\mathbf{t}$, the value of this feature must be subsumed by $Approp(\mathbf{t}, \text{F})$. And furthermore, the encoding will ensure, as required by HPSG, that every feature structure is extendible to a maximally specific well-type feature structure.[2]

Previous approaches, discussed in §2, have adopted a technique from Mellish [18] [19] in which each type is encoded as an open-ended data structure representing the path taken through the type hierarchy to reach that type. Or, in other words, a type $\mathbf{t}$ is represented as a sequence of types, starting at the most general type below $\top$ and ending at $\mathbf{t}$, in which each consecutive pair consists of supertype followed by immediate subtype. By bundling features together with the types that introduce them, it is then possible to allow the number of features on a type to increase as the type is further instantiated. The disadvantage of this representation, though, is that there is no unique path leading to a multiply-inherited type or any of its subtypes. While a grammar with multiple-inheritance cannot generally be represented in this approach, it is still possible, as explained in §2.1, to compile the multiple inheritance out of the type hierarchy and then term-encode a semantically equivalent grammar. In this approach, multiple inheritance exists as a convenience to the grammar writer, but is not actually used at run time.

[1] Other more complex constraints can be compiled out [13]. So a system that only uses appropriateness conditions is more general than it might first appear.

[2] See Pollard & Sag [20], p. 21

> ... a feature structure can be taken as a partial description of any of the well-typed (or totally well-typed, or totally well-typed and sort-resolved) feature structures that it subsumes. We choose to eliminate this possible source of confusion by using only totally well-typed sort-resolved feature structures as (total) models of linguistic entities and AVM diagrams (not feature structures) as descriptions.

It follows from this, that for an AVM to describe something, it must be extendible to a totally well-typed sort-resolved (or *type-resolved*) feature structure. As is standard in computational linguistics, I use the term *feature structure* to mean what Pollard and Sag mean by AVM. It turns out that for computational purposes, we will never be interested in Pollard & Sag's notion of a feature structure. For computational purposes, we want more compact feature structures, which can provably be extended to (totally) well-typed and type-resolved feature structures. See [11] [16] for details.

The encoding presented here will in some instances require the introduction of disjunctions in order to ensure satisfiability of feature structures, i.e., to ensure that feature structures are extendible to maximally specific well-typed feature structures.[3] This introduction of disjunctions may, in the worst case, exponentially increase the size of the grammar. Practical experience with HPSG grammars on the Troll system ([9], [15]) has shown, however, that feature structure compaction techniques can be used to keep this increase reasonably small. In §3, I discuss how these techniques can be incorporated into a term unification approach. In §3.1, I discuss the technique of *unextension*, which involves replacing disjunctions of feature structures with maximally specific types on their nodes by smaller disjunctions of feature structures in which the nodes are labelled by more general types. In §3.2, I discuss the use of *unfilling* to remove uninformative feature-value pairs. I show how this technique can be used not only to make feature structures smaller, but also to eliminate disjunctions. Then, in §3.3, I show how the remaining disjunctions can sometimes be efficiently encoded as *distributed* (or *named*) disjunction. Distributed disjunctions have been discussed elsewhere in the literature, but not in the context of term encoded feature structures. Finally in §4, I summarize the approach taken in this paper and discuss directions of future research.

# 2 Types-as-Paths Encoding

The types-as-paths encoding, introduced by Mellish [18], uses an open-ended data structure representing the path taken through the type hierarchy to reach that type. For simplicity, let us first consider how to represent types that do not take any features, such as the types subsumed by **a** in fig. 1.

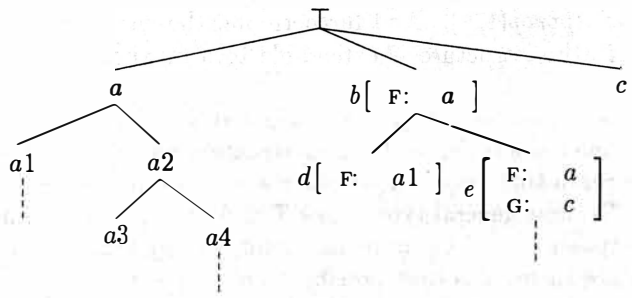



Figure 1: A Simple Type Hierarchy and Appropriateness Specification

| type | encoding |
|---|---|
| *a* | `a(_)` |
| *a*1 | `a(a1(_))` |
| *a*2 | `a(a2(_))` |
| *a*3 | `a(a2(a3))` |
| *a*4 | `a(a2(a4(_)))` |

It is clear that the encodings for **a**, **a2** and **a4** will all unify, whereas the encodings for **a2** and **a1** will not unify.[4] While, in general, the encoding employs open data structures, the example

[3] I am simplifying here for the sake of exposition. The notion of "satisfiable feature structure" is treated in full in King [16].

[4] The encoding used in ALE [4] is similar except that the paths may be gappy. Thus, **a4** could be encoded as any of the following: `_-a4-a2-a`, `_-a4-a`, `_-a4-a2` or `_-a4`. In this example, the path reflects the derivational history of how the **a4** got to be an **a4**. This same principle is used in the implementation of updateable arrays in the Quintus Prolog library. Since the encoding is not unique, a special purpose unifier must be used, which dereferences each type before unifying. Thus this gappy representation is not applicable for the goal of this paper, which is to use ordinary (Prolog-style) term unification.

shows that **a3**, since it is maximally specific, can be encoded as a ground term. If, however, we wish to be able to distinguish in a feature structure between reentrant and non-reentrant instances of **a3**, then we would need encode **a3** also as an open term: `a(a2(a3(_)))`.[5]

This encoding allows a particularly convenient encoding of feature information. If a type introduces $n$ features, then we simply add $n$ additional argument slots to that type in each path. For example, consider the encodings of **b** and **e**:

| type | encoding |
|---|---|
| *b* | `b(_,_F))` |
| *e* | `b(e(_,_G),_F))` |

As can be seen from this example, there is essentially no difference between the encoding of a simple type **t** and the encoding of a feature structure of type **t**. The encoding of each type has slots in it where all of the feature value information can be included. The encoding for **e** is particularly instructive. As can be seen, **e** takes two features, which are introduced at two different points along the path. Furthermore, **e** is a subtype of **b**, which has only a single argument position for the feature F. It is clear then that the number of slots for features can increase as a type is further instantiated.

The idea of bundling feature information along with the type that introduces that feature is certainly elegant. However, there is a drawback when not all of the information about a feature is located on a single type. For example, the type **d** inherits a feature F from the supertype **b**, but then adds a more specific value specification for this feature. Consider what happens when the feature structure $b[\text{F} : a]$ (encoded: `b(_,a(_))`) unifies with the feature structure $d$ (encoded: `b(d,_F)`). The unification of these two encodings is `b(d,a(_))`, i.e., $d[\text{F} : a]$, which is not well-typed. This, however, is not really a problem specific to the types-as-paths encoding. Any typed feature structure approach with appropriateness specifications needs to incorporate type inferencing. As discussed in §2.2 (see also [11]), one way to maintain appropriateness is to multiply out disjunctive possibilities. For example, $b[\text{F} : a]$ should be multiplied out to: $\{d[\text{F} : a1], e[\text{F} : a]\}$. As seen in the next section, handling multiple inheritance will also involve introducing disjunctions. So the problem of efficiently representing such disjunction (discussed in §3) will be of crucial importance.

## 2.1 Compiling Out Multiple Inheritance

Multiple inheritance is a genuine problem for the types-as-paths representation. The problem is that if a type can be reached by multiple paths through the type hierarchy, then there is no longer a unique representation for that type. A partial solution to this to use *multi-dimensional inheritance* as in Erbach [7] and Mellish [18]. This idea involves encoding types as a set of paths rather than as a single path. The intuition is supposed to be that each path in this set represents a different dimension in the inheritance hierarchy.

Consider, for example, the commonly used type hierarchy for lists in fig. 2. The types

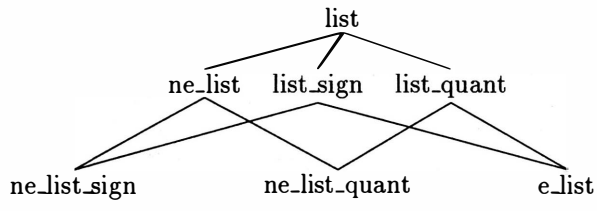



Figure 2: Multiple inheritance in type hierarchy for lists

**ne_list**, **list_sign** and **list_quant** are not mutually exclusive, so it seems reasonable to represent

[5] This idea is used, for example, in the ProFIT system [7] in order to distinguish between intensional and extensional types (see Carpenter [3] for this distinction). This distinction is certainly important. However, as it is a side issue for this paper, I will ignore it.

these types by a set of paths. **ne_list**, for example, could be represented as `list(ne_list,_,_)` and **ne_list_quant** as `list(ne_list,_,list_quant)`. This seems to work fine except that there would be no way to rule out the unification of the encodings for **ne_list_quant** and **ne_list_sign**. These two types unify to give the non-existent type `list(ne_list,list_sign,list_quant)`. So the restriction on this encoding is the following: if types **t** and **t'** have a subtype in common, then every subtype of **t** must be subsumed by **t'**. In the **list** example, **ne_list** and **list_sign** have a subtype in common (**ne_list_sign**). However, **list_sign** also subsumes **e_list**, which is not subsumed by **ne_list**. So while this approach may work for some special cases of multiple inheritance, it is not a general solution to the problem.[6]

Given the problems with representing multiple inheritance, it is reasonable to ask how important multiple inheritance is. In fact, given some reasonable closed world assumptions as discussed in the next section, it is possible to compile out all multiple inheritance. To see this, consider what happens when we remove from the list hierarchy the types **list_sign** and **list_quant**. As seen in fig. 3, removing these two types is sufficient to eliminate all multiple

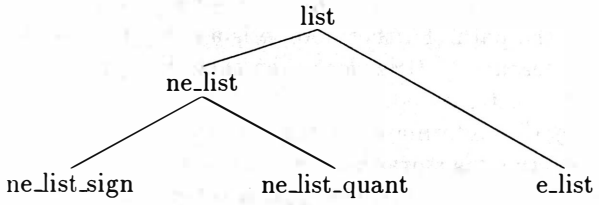



Figure 3: Type hierarchy for lists with multiple inheritance compiled out

inheritance. In order to legitimize removing types from the type hierarchy, two further steps are needed. First, some disjunctive appropriateness specifications must be introduced. For example, for any type **t** and feature F with $Approp(\mathbf{t}, \text{F}) = \mathbf{list\_sign}$, the new appropriateness conditions should include the disjunctive specification $Approp(\mathbf{t}, \text{F}) = \{\mathbf{ne\_list\_sign}, \mathbf{e\_list}\}$.[7] And second, any feature structure containing the type **list_sign** on a node must be compiled into two feature structures: one with **ne_list_sign** and one with **e_list**. So some lexical entries or rules may have to be compiled out into multiple instances. The question of how to deal with this introduced disjunction is treated in §3.

## 2.2 Closed World Assumptions

So far, we have seen that a closed world interpretation of the type hierarchy will allow us to compile out multiple inheritance at the cost of introducing some disjunctions into the grammar. But the closed world assumption is not a condition that is imposed solely for the purpose of term encoding. As shown in Gerdemann & King [10] [11], the closed world assumption is needed if types are to be used to encode any kind of feature cooccurrence restrictions. As noted by Copestake et al. [5] this deficiency of open-world type systems leads to serious problems for expressing any kind of linguistic constraints.

While the closed world assumption is clearly needed, it is also the case that there is a price to be paid for maintaining this condition. In particular, Gerdemann & King [10] [11] showed that maintaining this condition will sometimes involve multiplying out disjunctive possibilities. For example, consider the type hierarchy in fig. 4. Every feature structure of type **a** must ultimately be resolved to either a feature structure of type **a'** or of type **a''**. So the feature structure $a[\text{F}: \mathit{bool}, \text{G}: \mathit{bool}]$ really represents the set of resolvants $\{a'[\text{F}: +, \text{G}: -], a''[\text{F}: -, \text{G}: +]\}$ and the feature structure $a[\text{F}: \boxed{1}, \text{G}: \boxed{1}]$ really represents the empty set of feature structures.[8] To make our terminology precise, let us call feature structures such as $a'[\text{F}: +], a'[\text{F}: -], a''[\text{F}: +]$

[6] A second problem with the approach is that it provides no way to attach features to multiply inherited types. So if **ne_list_quant** has some features which are not inherited from either **ne_list** or **list_quant**, then there is no position in either path of types to which these features can be attached.

[7] Such disjunctive appropriateness specifications are allowed, at least internally, in the Troll system [9].

[8] Note that this last feature structure would be considered "well-typed" in ALE. For ALE, one must assume

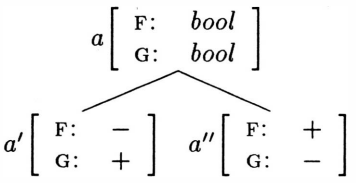



Figure 4: Type hierarchy requiring type resolution

and $a''[\text{F}: -]$ *extensions* of the feature structure $a[\text{F}: \mathit{bool}]$. So a feature structure $\alpha$ can be extended by replacing the type on each node by a species subsumed by that type. The set of resolvants is then the set of extensions which also satisfy the appropriateness conditions: in this case $\{a'[\text{F}: +], a''[\text{F}: -]\}$. Note that the resolvants of a feature structure need not be totally well typed in the sense of Carptenter [3], i.e., there can be appropriate features such as G which are not in the resolvant.[9] This fact will be of great importance when we consider efficient representations for sets of resolvants in the next section.

So if our term representation for typed feature structures is to maintain the constraints imposed by the appropriateness specification, it looks as if we will again have to expand feature structures out into multiple instances. In fact, as discussed in §3, there are methods both for reducing the need for disjunctions and for compactly representing those disjunctions that can't be eliminated. But before considering these methods, let us first consider one very desirable property of resolved feature structures, namely that the resolved feature structures are closed under unification.

If we use a specialized feature structure unifier, then there exists the possibility of building in type inferencing as part of unification. If all unification is simply term unification, however, then we don't have this option. We need all type inferencing to be static, i.e., applied at compile time.[10] So when two feature structures which satisfy appropriateness conditions unify, the result of this unification must also satisfy appropriateness conditions. As shown in Gerdemann & King [11] and King [16], this is indeed the case for resolved feature structures. Intuitively, the reason for this is quite simple. Type inferencing in a system like ALE has the effect of increasing the specificity of types on certain nodes in a feature structure. If all of these types are already maximally specific, then ALE-style type inferencing could only apply vacuously.

# 3 Minimizing Disjunctions

We have now seen two instances in which feature structures may need to be multiplied out into disjunctive possibilities. First, this may arise as a result of eliminating multiple inheritance from the type hierarchy. And second, as a result of the type resolution which is needed in order to ensure static typing. Now in this section, I discuss how the need for this disjunction can be reduced by the technique of *unextension* §3.1 and by *unfilling* §3.2. And then in §3.3, I show how the remaining disjunctions can at least sometimes be efficiently represented by using distributed disjunctions.

## 3.1 Unextension

First, consider the disjunction that is introduced by type resolution. Most of this disjunction can be eliminated by using the technique of *unextension*. Recall that in §2.2, I defined the

an open world semantics in which an object described by $a[\text{F}: \boxed{1}, \text{G}: \boxed{1}]$ is neither of type a′ nor of type a″.

[9] In fact, in general resolved feature structures cannot be totally well typed since total-well typing creates more nodes in a feature structure which would then need to be resolved and then total-well typed again. The result is an infinite loop [11].

[10] Note, for the sake of comparison, that type inferencing is not static in ALE. The unification of well-typed feature structures is not guaranteed to be well-typed. It is, in fact, not even guaranteed to be well-typable. Thus, the ALE unifier must do type inferencing at run time.

extension of a feature structure $\alpha$ to be a feature structure in which the type on each node of $\alpha$ has been replaced by a species subsumed by that type. Now, let us define the extensions of a set of feature structures $S$ as the set of all extensions of the members of $S$. And then define the unextension of a set of feature structures $S$ as the minimal cardinality set of feature structures $S'$ such that $\mathit{extensions}(S') = S$.[11] So whenever a feature structure is resolved to a large disjunction of feature structures $S$, we can normally compact this disjunction back down to a much smaller unextended set $S'$. Since the extensions of $S$ and $S'$ are the same, it is clear that these two sets of feature structures represent the same information.

As a simple—but extremely commonly occurring—example, consider a feature structure $F$, all of whose extensions are well-typed. In this case, the set of resolvants of $F$ is exactly the set of extensions of $F$. So the unextended set of resolvants of $F$ is simply $\{F\}$.[12] So resolving and then unextending $F$ appears at first to accomplish nothing. But, in fact, there is a gain. Initially, with the feature structure $F$ we had no guarantee that static typing would be safe. But with the resolved-unextended set $\{F\}$, we now know that there are no non-well typed extensions, so there will never be a need for run-time type inferencing.

As another example, consider the set of resolvants that HPSG allows for:

*head-struc*[HEAD-DTR: *sign*, COMP-DTRS: *elist*]

Since the only subtype of **head-struc** for which **elist** is an appropriate value on **comp-dtrs** is **head-comp-struc**, we get the following set of resolvants:

{*head-comp-struc*[HEAD-DTR: *word*, COMP-DTRS: *elist*],
*head-comp-struc*[HEAD-DTR: *phrase*, COMP-DTRS: *elist*]}.

This two element set corresponds exactly to the set of extensions for the following one element unextended set:

{*head-comp-struc*[HEAD-DTR: *sign*, COMP-DTRS: *elist*]}

So the combination of resolving and unextending simply has the effect of bumping the top-level type from **head-struc** to **head-comp-struc**. This then rules out the malformed feature structure that might have been obtained, for example, by unification with a feature structure of type **head-filler-struc**.

Consider now the disjunctions that are introduced by compiling out multiple inheritance. Recall that this technique involves eliminating intermediate types from the hierarchy. It does not remove any species from the hierarchy. Thus, regardless of whether or not multiple inheritance has been compiled out, the resolvants of a feature structure $F$ will be exactly the same. Since, unextension involves replacing species on nodes with intermediate types, the possibilities for unextending a set of feature structures will be reduced when some intermediate types have been removed. So it turns out that compiling out multiple inheritance actually introduces disjunction in a rather indirect manner.

## 3.2 Unfilling

Another operation that can be used to reduce the need for disjunction is *unfilling*, which is the reverse of Carpenter's [3] fill operation. In general, the purpose of unfilling is to keep

[11] The normal form result of Götz [12] shows, albeit rather indirectly, that there is a unique unextension for the set of resolvants of a feature structure—assuming there are no unary branches in the type hierarchy. For Götz, unextension is an implicit part of his function for resolving feature structures. I have abstracted unextension out as a separate operation purely for expository reasons. In an actual implementation (such as Troll [9]) it makes more sense to follow Götz's approach since it is not very efficient for the compiler to expand feature structures out into huge sets that then have to be collapsed back down.

[12] There is, however, one complication, namely, if there are unary branches in the type hierarchy, then a feature structure might be resolved and then unextended back to a slightly different, but semantically identical feature structure (see Carpenter [3], chap. 9). For our purposes here, it doesn't matter if the unextension of a set of feature structures is not unique.

feature structures small. If a feature in a feature structure has a value which is no more specific than the appropriateness specification would require, then—assuming no reentrancies would be eliminated and no dangling parts of the feature structure would be created—that feature and its value may be removed. So, in HPSG for example, if the AUX feature in a feature structure of type **verb** has the value **boolean**, then this AUX feature can be removed.

Unfilling is most important, however, when it turns out that eliminating features allows us to further apply unextension to eliminate disjunctions. For example, given the type hierarchy and appropriateness specification in fig. 4 above, the feature structure $a[\text{F} : \mathit{bool}, \text{G} : \mathit{bool}]$ is resolved to $\{a'[\text{F} : +, \text{G} : -], a''[\text{F} : -, \text{G} : +]\}$. However, both of the features F and G have uninformative values, i.e., values which tell us nothing more than we already know from the appropriateness specification. In fact, this is true both in the unresolved feature structure and in each of the resolvants. So these two features can be unfilled to give us the new set of resolvants $\{a', a''\}$. This new set of resolvants can now be unextended to the set $\{a\}$.

There is clearly a great deal more that can be said about unfilling and about the class of unfilled feature structures. The issues, however, are not specific to the problem of term encoding. So I will simply refer the reader to the discussion in Gerdemann & King [11] and Götz [12].

## 3.3 Distributed Disjunctions

Unextension and unfilling can be used to eliminate quite a lot of disjunctions. Unfortunately, not all disjunctions can be eliminated with these techniques. But still, as noted in Gerdemann & King [10] [11], the remaining disjunctions, have a rather special property. All of the resolvants of a feature structure have the same shape. They differ only in the types labelling the nodes. In a graph-unification based approach, this property can be used to allow the use of a relatively simple version of *distributed* (or *named*) disjunction. This device can be used to push top level disjunctions down to local, interacting disjunctions as in this example:[13]

$$\left\{ \begin{bmatrix} \mathit{ne\_list\_sign} \\ \text{HEAD: } \mathit{sign} \\ \text{TAIL: } \mathit{list\_sign} \end{bmatrix}, \begin{bmatrix} \mathit{ne\_list\_quant} \\ \text{HEAD: } \mathit{quant} \\ \text{TAIL: } \mathit{list\_quant} \end{bmatrix} \right\} \Rightarrow \begin{bmatrix} \langle 1\ \mathit{ne\_list\_sign}\ \mathit{ne\_list\_quant} \rangle \\ \text{HEAD: } \langle 1\ \mathit{sign}\ \mathit{quant} \rangle \\ \text{TAIL: } \langle 1\ \mathit{list\_sign}\ \mathit{list\_quant} \rangle \end{bmatrix}$$

The idea is that if the $n$th alternative is chosen for a disjunction of a particular name, then then $n$th alternative has to be chosen uniformly for all other instances of disjunction with the same name.

Distributed disjunctions have been discussed in a fair amount of recent literature ([17], [6], [8], [14]). This version of distributed disjunction, however, is particularly simple since only the type labels are involved. It is not at all difficult to modify a graph unification algorithm in order to handle such disjunctions. Such a modified unification algorithm is used, for example, in the Troll system [9].

For term-represented feature structures, however, it will not be possible to directly encode such distributed disjunctions.[14] So rather than encoding distributed disjunctions in a feature

[13] The example is a little unrealistic since the features HEAD and TAIL could be unfilled. To make the example more realistic, imagine the feature structures embedded in a larger feature structure and imagine that the values of HEAD and TAIL are reentrant with other parts of this feature structure, so that these features could not be unfilled without breaking these reentrancies.

[14] Actually there is an alternative representation for types that would allow a limited amount of direct encoding of distributed disjunctions. Following the basic idea of Mellish [18], one could represent each type as a set of species encoded as a *vset*, where a vset is a term $\mathit{vset}(X_0, X_1, \ldots, X_N)$, with the conditions that:

- $X_0 = 0$
- $X_N = 1$
- $X_i = X_{i-1}$ if the $i$'th possible element is not in the set

With this *types-as-vsets* representation, some dependencies can be represented as variable sharing across vsets. This idea is elaborated upon in an earlier (and longer) version of the present paper.

structure, we must encode them as definite clause attachments to the feature structure. The idea is fairly simple, to efficiently represent a set of feature structures $S$, we factor $S$ into first term-represented feature structure, $\sqcup S$, expressing the commonalities across all of the members of $S$, and second, a set of definite clause attachments expressing all of the allowable further extensions. There is, however, one hitch; namely, how do we know that $\sqcup S$ will be expressible as a Prolog term? If $\sqcup S$ is well typed, then there is no problem. But if $\sqcup S$ contains a node labeled with t with a with an inappropriate feature F, then the types-as-paths representation simply provides no argument position for this inappropriate feature.

A solution to this problem can be found by imposing the feature introduction condition of Carpenter [3]. This condition requires that for each feature F, there is a unique most general type $Intro(\text{F})$ for which this feature is appropriate. Or, the other way around, if F is appropriate for t and t′, then it will also be appropriate for $\mathbf{t} \sqcup \mathbf{t}''$. It is straightforward to see, then, that given the feature introduction condition, if the feature structures $FS$ and $FS'$ contain no inappropriate features, then $FS \sqcup FS'$ will also contain no inappropriate features.

As an example, consider again the type hierarchy in fig. 1. Suppose that we want to efficiently represent the set:[15]

$$\{d[\text{F}: a1], e[\text{F}: a]\}$$

The generalization of these two feature structures is then term encodable as follows:

$$b[\text{F}: a] \equiv \texttt{b(X, a(Y))}$$

Suppose now, that we want to use this feature structure as the lexical entry for the word $w$. For this simple example, we can encode this with just one definite clause attachment:[16]

```
lex(w, b(X, a(Y))) :-
   p(X, Y).

p(d, a1(_)).
p(e(_), _).
```

One should note here the similarity to distributed disjunctions. The term `b(X, a(Y))` represents the underlying shape of the feature structure and the defining clauses for `p` encode dependencies between types. It is, in fact, rather surprising that this division is even possible. One of the features of the types-as-paths encoding is that the features are bundled together with the types that introduce them. So one might not have expected to see these two types of information unbundled in this manner.

# 4 Conclusions

Some previous approaches to term encoding of typed feature structures have enforced restrictions against multiple inheritance and against having having more specific feature-value declarations on subtypes. But such restrictions make typing virtually useless for encoding any meaningful constraints. The only restriction imposed in the present approach is the feature introduction condition, which is also imposed in ALE ([4]). In fact, even this minor restriction could be eliminated if we were to allow somewhat less efficient definite clause attachments.

We have seen that in order to enforce constraints encoded in the appropriateness specifications, it will sometimes be necessary to use definite clause attachments to encode disjunctive possibilities. Practical experience with the Troll system suggests that not many such attachments will be needed. Nevertheless, they will arise and will therefore need to be processed

[15] Again, this is an unrealistic example (see footnote 13). One would not normally need distributed disjunctions for such a simple case.

[16] Multiple attachments would correspond to named disjunctions with different names.

efficiently. Certainly options such as delaying these goals or otherwise treating them as constraints can be explored. In fact, one of the main advantages of having a term encoding is that so many options are available from all of the literature on efficient processing of logic programs. So the approach to term encoding presented here should really be viewed as just the first step in the direction of efficient processing of typed feature structure grammars.

# References


[1] H. Alshawi, editor. *The Core Language Engine.* MIT Press, Cambridge, Mass, 1991.

[2] H. Alshawi, D.J. Arnold, R. Backofen, D. M. Carter, J. Lindop, K. Netter, J. Tsujii, and H. Uszkoreit. Eurotra 6/1: Rule formalism and virtual machine design study. final report. Technical report, SRI International, Cambridge, 1991. The platform specification for ALEP. Typing in ALEP is explained in a variety of succeeding LRE technical reports.

[3] Bob Carpenter. *The Logic of Typed Feature Structures.* Cambridge Tracts in Theoretical Computer Science 32. Cambridge University Press, 1992.

[4] Bob Carpenter and Gerald Penn. ALE *The Attribute Logic Engine, User's Guide*, 1994.

[5] Ann Copestake, Antonio Sanfilippo, Ted Briscoe, and Valeria de Paiva. The ACQUILEX LKB: An introduction. In *Inheritance, Defaults, and the Lexicon*, pages 148–163. Cambridge University Press, 1993.

[6] Jochen Dörre and Andreas Eisele. Feature logic with disjunctive unification. In *COLING-90 vol. 2*, pages 100–105, 1990.

[7] Gregor Erbach. ProFIT: Prolog with features, inheritance and templates. In *EACL Proceedings, 7th Annual Meeting*, pages 180–187, 1995.

[8] Dale Gerdemann. *Parsing and Generation of Unification Grammars.* PhD thesis, University of Illinois, 1991. Published as Beckman Institute Cognitive Science Technical Report CS-91-06.

[9] Dale Gerdemann and Thilo Götz. Troll: Type resolution system, user's guide, 1994.

[10] Dale Gerdemann and Paul John King. Typed feature structures for expressing and computationally implementing feature cooccurrence restrictions. In *Proceedings of 4. Fachtagung der Sektion Computerlinguistik der Deutschen Gesellschaft für Sprachwissenschaft*, pages 33–39, 1993.

[11] Dale Gerdemann and Paul John King. The correct and efficient implementation of appropriateness specifications for typed feature structures. In *COLING 94, Proceedings*, pages 956–960, 1994.

[12] Thilo Götz. A normal form for typed feature structures. Master's thesis, Universität Tübingen, 1993.

[13] Thilo Götz and Walt Detmar Meurers. Compiling hpsg type constraints into definite clause programs. In *Proceedings of the Thrirty-Third Annual Meeting of the ACL*, Boston, USA, 1995. ACL.

[14] John Griffith. Optimizing feature structure unification with dependent disjunctions. In John Griffith, Erhard Hinrichs, and Tsuneko Nakazawa, editors, *Topics in Constraint Grammar Formalism for Computational Linguistics: Papers Presented at the Workshop on Grammar Formalism for Natural Language Processing held at ESSLLI-94, Copenhagen.*, number SfS-Report-04-95 in SfS Technical Report. Seminar für Sprachwissenschaft: Universität Tübingen, 1994.

[15] Erhard Hinrichs, Detmar Meurers, and Tsuneko Nakazawa. Partial-vp and split-np topicalization in german – an HPSG anaylsis and it's implementation. Technical Report 58, SFB 340, 1994. Arbeitspapiere des Sonderforschungsbereichs 340, Sprachtheoretische Grundlagen für die Computerlinguistik.

[16] Paul John King. Typed feature structures as descriptions. In *COLING 94, Proceedings*, pages 1250–1254, 1994.

[17] John T. Maxwell III and Ronald M Kaplan. An overview of disjunctive constraint satisfaction. In *Proceedings of International Workshop on Parsing Technologies*, pages 18–27, 1989.

[18] Christopher S. Mellish. Implementing systemic classification by unification. *Computational Linguistics*, 14(1):40–51, 1988.

[19] Christopher S. Mellish. Term-encodable description spaces. In D.R. Brough, editor, *Logic Programming New Frontiers*, pages 189–207. Intellect, Oxford, 1992.

[20] Carl Pollard and Ivan Sag. *Head Driven Phrase Structure Grammar*. CLSI/University of Chicago Press, Chicago, 1994.

# GENERIC RULES AND NON-CONSTITUENT COORDINATION

Julio Gonzalo
UNED
julio@ieec.uned.es

Teresa Solías
Universidad de Valladolid
solias@cpd.uva.es

**Abstract**

We present a metagrammatical formalism, *generic rules*, to give a default interpretation to grammar rules. Our formalism introduces a process of *dynamic binding* interfacing the level of pure grammatical knowledge representation and the parsing level. We present an approach to non-constituent coordination within categorial grammars, and reformulate it as a generic rule. This reformulation is context-free parsable and reduces drastically the search space associated to the parsing task for such phenomena.

## 1 Introduction

Grammatical theories always make a distinction between unmarked and marked phenomena. The general case is to use certain mechanisms which are likely to deal with regular behaviour. More complex mechanisms should be available when exceptional behaviour is present. However, this difference is not formally expressed in most grammar accounts of complex phenomena. The need to restrict licensing of additional grammar resources is commonly asserted outside of the formal framework used. As exceptional rules involve a higher computational cost, parsing processes associated to such formalisms become computationally intractable.

Non-constituent coordination treatments show this pattern. In general, a conjunction combines constituents. But there are cases where the conjunction coordinates linguistic elements which do not form a constituent. For instance, consider '*John went to Alabama in summer and to Spain in fall*'. The conjunction '*and*' is coordinating ' *to Alabama in summer*' with ' *to Spain in fall*', which do not form constituents. The solutions displayed in the literature incorporate additional grammatical resources to deal with such data. We would expect a high restriction on the use of the rules associated to this phenomena, but all we have is informal comments about when to apply them. That makes parsing time of the grammar exponential in the length of the string.

Though there is a general concern on the need to make explicit some process of rule licensing, no formal framework has been proposed, to our knowledge, to regulate the interaction between regular and exceptional grammatical resources. In this paper we propose a metagrammatical formalism, *generic rules*, to give a different degree of specificity to each grammatical rule. We present an approach to parse non-constituent coordination within categorial grammars that can be parsed in polynomial time when reformulated as a generic rule.

The essential idea is to introduce a process of *dynamic binding* interfacing the level of pure grammatical representation and the parsing processes. When the parsing process calls the grammar to combine linguistic objects, the dynamic binding process inspects the applicable grammatical resources, considers their different priority and returns an effective rule, or set of

rules, to be applied on such linguistic objects. This approach is inspired in the object-oriented programming paradigm, where polymorphism is exploited in order to get default and specific behaviour.

In order to make dynamic binding a meaningful process, we will view generic and specific behaviour as object-oriented functions that map daughters information into mother features. A set of such functions will be called a *generic rule.* Precedence between them will be established according to the specificity of the type constraints on the daughters.

This paper is structured as follows: first, we outline the main ideas behind our approach. Next, we make a more formal presentation of *generic rules.* Then, we turn to the linguistic problem of non-constituent coordination, and give a linguistic explanation within categorial grammar that uses the *tuple* operation introduced in [Solías, 1993, Morrill and Solias, 1993]. Recognition with that grammar is NP-hard, as it is with most extensions of Lambek Calculus. We present, then, a reformulation of that grammar as a generic rule. The representational facilities of generic rules allow the specification of each grammatical resource at an appropriate level of specificity, drastically constraining the search space for the parsing task.

## 2 Generic Rules

### 2.1 Underlying Ideas

Consider a context-free grammar - possibly augmented with functional restrictions attached to each rule - that includes the hierarchy $\mathtt{VP} \rightarrow \mathtt{VP}_1 \mid \mathtt{VP}_2 \mid \mathtt{VP}_3 \mid \mathtt{VP}_4 \mid \mathtt{VP}_i$ and the rule $\mathtt{S} \rightarrow \mathtt{NP\ VP}$.

If we decide to write down a new rule that specifies a different, more specific behaviour for $\mathtt{VP}_i$ when combining with a NP, we could think of adding a new rule as: $\mathtt{S}_i \rightarrow \mathtt{NP\ VP}_i$ . If our aim is to state this rule as *the* rule to be used when the VP belongs to the more specific subclass $\mathtt{VP}_i$, we want the production $\mathtt{S} \Rightarrow \mathtt{NP\ VP}_i$ to be valid, but not the production $\mathtt{S} \Rightarrow \mathtt{NP\ VP} \Rightarrow \mathtt{NP\ VP}_i$. Just adding $\mathtt{S}_i \rightarrow \mathtt{NP\ VP}_i$ to the CFG does not capture the exceptional sense of the rule; in a CFG, both derivations are equally possible.

We have two options to get the desired behaviour from the CFG. The first one is to keep $\mathtt{S} \rightarrow \mathtt{NP\ VP}$ and rewrite the specific rule $\mathtt{S}_i \rightarrow \mathtt{NP\ VP}_i$ into the following set of rules:

$$\mathtt{S} \rightarrow \mathtt{NP\ VP}_1 \mid \mathtt{NP\ VP}_2 \mid \mathtt{NP\ VP}_3 \mid \mathtt{NP\ VP}_4 \tag{1}$$

The second one is to rewrite the hierarchy, in order to keep the number of rules invariant:

$$\begin{array}{ll} \mathtt{VP} \rightarrow \mathtt{VP}_i \mid \mathtt{VP}_t & \mathtt{S} \rightarrow \mathtt{NP\ VP}_t \\ \mathtt{VP}_t \rightarrow \mathtt{VP}_1 \mid \mathtt{VP}_2 \mid \mathtt{VP}_3 \mid \mathtt{VP}_4 & \mathtt{S}_i \rightarrow \mathtt{NP\ VP}_i \end{array} \tag{2}$$

None of these options is an incremental way of capturing exceptions in a grammar. The introduction of an exception forces the revision of previously described lexical types and grammar rules.

This naive example exemplifies a problem that becomes significant when an operation or rule carries on most of the information-combining task. This is a standard situation in lexicalist approaches to grammar. For instance, [Dowty, 1988] proposes a distinction between a basic categorial grammar, which only includes *forward* and *backward application*, and an extended grammar that also includes *type raising* and *functional composition.* The basic grammar is used for most of the analysis, whereas the extended one should only be used in specific situations such as non-constituent coordination. It is suggested that this rules should be licensed only when functional application fails to parse a string. However, no formalization of the different status of such rules is offered.

We propose a formalism to express and handle default and exceptional rules within a phrase-structure approach that permits this kind of representations. Our formalism describes rules as object-oriented functions that map daughters information into mother features. As in object-oriented programming, precedence between rules is not given statically in the linguistic signature, but it is dynamically established upon the types of the input structures of the compositional process. When combining two constituents, a metagrammatical mechanism establishes

a partial order for the candidate rules and selects the most appropriate rule (or set of rules) to be applied.

Before defining such framework, let us introduce the proposal with a generic rule representation for the naive example above. The generic rule having the desired behaviour for rule $\mathtt{S} \rightarrow \mathtt{NP\ VP}$ and rule $\mathtt{S}_i \rightarrow \mathtt{NP\ VP}_i$ has this aspect:

$$\mathrm{SYN} = \begin{cases} \mathrm{SYN}_{NP \otimes VP}(x, y) = S \\ \mathrm{SYN}_{NP \otimes VP_i}(x, y) = S_i \end{cases} \tag{3}$$

Equation 3 describes a generic rule, SYN, that returns a type for the composition of two linguistic objects. $\mathrm{SYN}_{NP \otimes VP}(x, y) = S$ is a partial rule applicable when the arguments $x, y$ belong to the types $NP, VP$, in that order. $\mathrm{SYN}_{NP \otimes VP_i}(x, y) = S_i$ is applicable when the arguments belong to the types $NP, VP_i$.

A bottom-up parsing process may call SYN when it tries to combine two objects with types $x, y$. The applicable partial rules will be $\{\mathrm{SYN}_{a \otimes b} \mid a \leq x, b \leq y\}$. The natural way to establish precedence between rules is attending to the specificity of the type constraints of their arguments: $\mathrm{SYN}_{a \otimes b}$ will be more specific than $\mathrm{SYN}_{c \otimes d}$ if $a \leq c, b \leq d$.

Some examples of the application of SYN would be:

$$\begin{array}{ll} \mathrm{SYN}(NP, VP) = S & \mathrm{SYN}(NP, VP_2) = S \\ \mathrm{SYN}(VP_2, NP) = \perp & \mathrm{SYN}(NP, VP_i) = S_i \end{array} \tag{4}$$

In each case, the following partial rules have been applied:

$$\begin{array}{ll} \mathrm{SYN}(NP, VP) \Longrightarrow \mathrm{SYN}_{NP \otimes VP} & \mathrm{SYN}(NP, VP_2) \Longleftrightarrow \mathrm{SYN}_{NP \otimes VP} \\ \mathrm{SYN}(VP_2, NP) \Longleftrightarrow \perp \text{ (no applicable rule)} & \mathrm{SYN}(NP, VP_i) \Longleftrightarrow \mathrm{SYN}_{NP \otimes VP} \end{array} \tag{5}$$

In that way, we have the ability to express default rules, and rules with different degrees of specificity in an incremental way, without a need to give them different formal status. Also, only minor changes have to be made to a CFG parsing algorithm in order to work with generic rules.

### 2.2 Definition of a Generic Rule

Consider a partially ordered set (*poset*) $\langle \leq, \mathcal{T} \rangle$ and a domain of linguistic objects $\mathcal{O}$ typed with $\langle \leq, \mathcal{T} \rangle$ (i.e., there is a function type:$\mathcal{O} \rightarrow \mathcal{T}$). Consider also an informational domain $\Phi$ associated to $\mathcal{O}$ by means of a function $\phi$:$\mathcal{O} \rightarrow \Phi$ (perhaps multivaluated) that associates at least one element in $\Phi$ to every object of $\mathcal{O}$.

A *generic rule* G over the tuple $\langle \langle \leq, \mathcal{T} \rangle, \mathcal{O}, \Phi \rangle$ is another tuple $\langle \langle \preceq, \mathcal{T}^* \rangle, \mathcal{F}, \mathcal{G} \rangle$, where:

- $\mathcal{F}$ is a set of functions, that will be called *partial rules*, of the form:

$$\mathrm{f}_{t_1 \otimes t_2} : \Phi \times \Phi \rightarrow \Phi$$

  where $t_1, t_2 \in \mathcal{T}$, and the following condition holds on the set of partial rules:
  $\forall \mathrm{f}_{t_1 \otimes t_2}, \mathrm{f}_{q_1 \otimes q_2} \in \mathcal{F}, t_1 = q_1 \wedge t_2 = q_2 \Longleftrightarrow \mathrm{f}_{t_1 \otimes t_2} = \mathrm{f}_{q_1 \otimes q_2}$.

- $\langle \preceq, \mathcal{T}^* \rangle$ is a partially ordered poset defined over $\mathcal{F}$ as follows:

$$\mathcal{T}^* \equiv \{t_1 \otimes t_2 \mid \mathrm{f}_{t_1 \otimes t_2} \in \mathcal{F}\}$$
$$\forall x_1 \otimes x_2, y_1 \otimes y_2 \in \mathcal{T}^* \quad x_1 \otimes x_2 \preceq y_1 \otimes y_2 \Longleftrightarrow x_1 \leq y_1 \text{ and } x_2 \leq y_2$$

  We call *cartesian poset* $\langle \preceq, \mathcal{T}^* \rangle$ over a poset $\langle \leq, \mathcal{T} \rangle$ to any subset of $\mathcal{T} \times \mathcal{T}$ equipped with the partial ordering relation $\preceq$.

  We call *cartesian type* of a partial rule $\mathrm{f}_{t_1 \otimes t_2}$ to the type specification $t_1 \otimes t_2$.

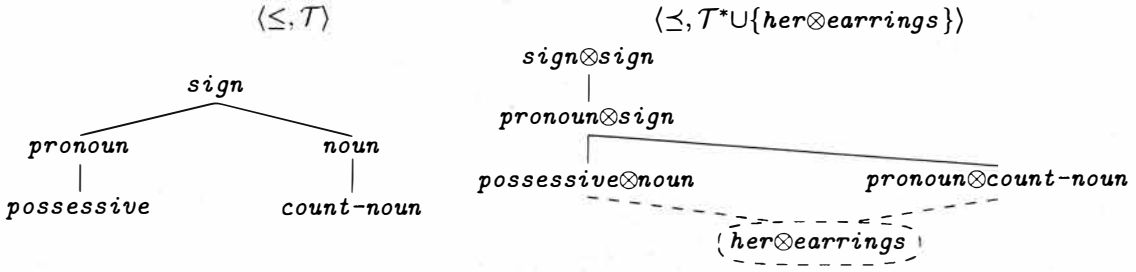



Figure 1: When the generic rule associated to $\mathcal{T}^*$ is invoked to compose *her* and *earrings*, a cartesian type *her⊗earrings* is dynamically considered, but it has more than one direct supertype; therefore, there is not a single most specific rule to be applied on that constituents.

- $\forall x_1, x_2 \in \mathcal{T}, \mathcal{G}(x_1, x_2) \equiv f_{t_1 \otimes t_2}$ such that $t_1 \otimes t_2$ is the lowest upper bound of $x_1 \otimes x_2$ in $\mathcal{T}^* \cup \{x_1 \otimes x_2\}$.

  $\mathcal{G}$ is called the *dynamic binding function* for G. It interfaces parsing processes with the grammar, selecting the most specific partial rule in $\mathcal{F}$ given two arguments of types $x_1, x_2$.

Note that $\mathcal{T}^*$ is a subset of the cartesian product $\mathcal{T} \times \mathcal{T}$, and that the partial ordering relation $\preceq$ is deduced from the relation $\leq$ that holds between the elements of $\mathcal{T}$. By means of the definition of *cartesian poset*, we have managed to axiomatize precedence issues between partial rules, turning the dynamic binding process into a simple order checking over the elements of a poset.

The function $\mathcal{G}$ provides the interface of the generic rule, as a system of object-oriented functions, with the parsing process that calls it. In particular, the dynamic binding function provides the default interpretation for partial rules. Given a pair of arguments of types $x_1, x_2$, a dynamic extension $\mathcal{T}^* \cup \{x_1 \otimes x_2\}$ of the original cartesian poset $\langle \preceq, \mathcal{T}^* \rangle$ that includes the cartesian type $x_1 \otimes x_2$ is considered. In virtue of the definition of the partial ordering relation $\preceq$, the new type $x_1 \otimes x_2$ takes its place in the hierarchy. Its supertypes denote all the applicable rules to compose the pair of arguments, and the partial ordering between this supertypes denotes precedence between them. The action of the generic rule G over a couple of linguistic objects with types $x_1, x_2$ and associated expressions $\phi_1, \phi_2$ is given by $\mathcal{G}(x_1, x_2)(\phi_1, \phi_2)$.

In our definition, precedence is used to keep the most specific rule and override the rest. However, more sophisticated binding processes could be considered. For instance, if the partial rules were unification constraints, some default version of unification could be performed on the applicable rules, taking their relative precedence into account.

We have adopted the restriction of binary branching (binary rules) to take full advantage of the concept of dynamic binding with the minimum effort. However, this is not a serious limitation: most linguistically significant rules are binary, and those which are not can be easily converted in binary rules.

### 2.3 Well-formedness

Given the definition of the preceding section, there can be situations where precedence conflicts cannot be solved. When the cartesian type that represents the arguments of a call to the generic rule has more than one direct supertype, there is not a single "most specific rule" (see the example in Figure 1).

If this indeterminacy is possible for a given generic rule, we say that the Cartesian Poset $\langle \preceq, \mathcal{T}^* \rangle$ is not well-formed:

*A Cartesian Poset* $\langle \preceq, \mathcal{T}^* \rangle$ *is* **well-formed** *iff*
$\forall\ x_1 \otimes x_2\ , y_1 \otimes y_2\ \in \langle \preceq, \mathcal{T}^* \rangle$ *not ordered, such that* $m_1 = x_1 \wedge y_1\ \in \langle \leq, \mathcal{T} \rangle$ *and* $m_2 = x_2 \wedge y_2\ \in \langle \leq, \mathcal{T} \rangle$, *it holds that* $m_1 \otimes m_2\ \in \langle \preceq, \mathcal{T}^* \rangle$ [1]

[1] This definition assumes that the original poset is bounded complete, i.e., each pair of types have at most

This kind of indeterminacy appears also when dealing with multiple default inheritance [Daelemans et al., 1992] or default unification over feature structures with structure-sharing [Bouma, 1992, Carpenter, 1993], and can only be solved adding extra ordering information or forbidding such situations. Our well-formedness condition adopts the first solution. The main reason is that the cartesian hierarchies are not defined by the user, but arranged by the system. Such conflicts are potentially dangerous, as the grammar writer is not necessarily aware of them. An advantage of the definition of well-formedness is that it can detect inconsistencies at compile-time and signal them for correction.

### 2.4 Parsing with Generic Rules

A generic rule may interact with a phrase-structure grammar to perform the composition of some informational field. In [Gonzalo, 1995], for instance, we propose generic rules as a well-suited mechanism to perform categorial semantic interpretation as a modular process that interacts with a phrase-structure grammar. On the other hand, a grammar could be made up exclusively of generic rules, adequately combined to perform parsing. We will present a simple account of each of these possibilities.

We will adopt here the deductive parsing approach described in [Shieber et al., 1994]; it provides a neat account of parsing systems, simplifying our presentation. We will start from a bottom-up shift reduce algorithm as presented in [Shieber et al., 1994]. We will gradually adopt this algorithm to capture generic rule parsing.

Let $\alpha$ be a string of terminals $w_i$. Let $[\alpha \bullet, j]$ stand for $\alpha w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_1 \cdots w_n$. In [Shieber et al., 1994] a shift-reduce bottom-up deductive parsing system is expressed as the following calculus:

**Inference Rules:**

**Axiom:** $[\bullet, 0]$
**Goal:** $[S \bullet, n]$

**Shift** $\dfrac{[\alpha \bullet, j]}{[\alpha w_{j+1} \bullet, j+1]}$

**Reduce** $\dfrac{[\alpha\gamma \bullet, j]}{[\alpha B \bullet, j]} \quad B \rightarrow \gamma$

This parser can be augmented to do semantic interpretation. If each rule has an associated function that related the meanings of the elements in the rhs with the meaning of the lhs element, the basic tuples are $[\alpha \bullet, j, \phi]$. The reduce rule carries on semantic composition:[2]

$$\textbf{Reduce} \quad \frac{[\alpha\gamma \bullet, j, \Phi < \phi_1 \cdots \phi_{|\gamma|} >]}{[\alpha B \bullet, j, \phi < f(\phi_1 \cdots \phi_{|\gamma|}) >]} \quad B \rightarrow \gamma, \text{ f}$$

where f is the semantic composition function associated to the rule $B \rightarrow \gamma$.

We can augment the shift-reduce parser in a similar way to represent the interaction between a generic rule and a phrase-structure grammar. We have to take account of two differences between generic rule parsing and the syntactic-semantic parser above:

1. The action of the generic rule has to be specified through the dynamic binding function. There is not a partial rule associated to each context-free rule.

2. The interpretation of the context-free rules has to be slightly modified to take into account the hierarchization of non-terminals. A hierarchy is equivalent, in context-free terms, to a set of unary rules in which every inmediate-ordering relation is expressed writing a type as a left-hand side of a rule, and its subtype as the right-hand side. Being these rules "compiled" into the hierarchy, a context-free rule as $T \rightarrow T_1 T_2$, has to be interpreted as $T \rightarrow X_1 X_2$ such that $X_1 \leq T_1, X_2 \leq T_2$.

one common subtype.

[2]The other rules are modified trivially.

The basic element in the logic that represents the parsing algorithm is the same as in the preceding case: $[\alpha \bullet, j, \Phi]$. Now, $\Phi$ is an expression that belongs to the informational domain associated to a generic rule G.

The reduce rule that takes account of the interaction of a generic rule with a context-free grammar is

$$\textbf{Reduce}\quad \frac{[\alpha \boldsymbol{x}_1 \boldsymbol{x}_2 \bullet, j, \Phi < \phi_1 \phi_2 >]}{[\alpha A \bullet, j, \Phi < \mathcal{G}(\boldsymbol{x}_1, \boldsymbol{x}_2)(\phi_1, \phi_2) >]} \quad \begin{array}{l} A \rightarrow BC, \boldsymbol{x}_1 \leq B, \boldsymbol{x}_2 \leq C \\ \mathcal{G}(\boldsymbol{x}_1, \boldsymbol{x}_2)(\phi_1, \phi_2) \neq \perp \end{array} \tag{6}$$

The interaction with the generic rule is expressed in the term $\mathcal{G}(\boldsymbol{x}_1, \boldsymbol{x}_2)(\phi_1, \phi_2)$. The left part, $\mathcal{G}(\boldsymbol{x}_1, \boldsymbol{x}_2)$, performs dynamic binding, returning the effective rule that will be applied on $\boldsymbol{x}_1, \boldsymbol{x}_2$.

The side conditions on this rule are: one, that exists an applicable syntactic rule $A \rightarrow BC, \boldsymbol{x}_1 \leq B, \boldsymbol{x}_2 \leq C$. And two, that the generic function, applied over $\phi_1, \phi_2$, returns a positive result. Note that it could be the case that there is an appropriate syntactic rule, but the additional restrictions imposed by the generic rule do not hold [3].

It is interesting to remark that the dynamic binding process depends on the particular objects in the Reduce rule, not only on the conditions of the context-free rule. That makes impossible to pre-attach an "effective partial rule" to each context-free rule at compilation time, such that the binding process could be performed off-line. This fact is reflected in the term $\mathcal{G}(\boldsymbol{x}_1, \boldsymbol{x}_2)$. If it were $\mathcal{G}(B, C)$, binding could be done at compile time. Though this behaviour introduces and additional complexity factor in generic rule parsing, it allows the specification of semantic and syntactic processes as independent mechanisms that interact modularly.

A particular case of generic rule is that in which the combination functions simply return a type, as in our first example. In such a rule, the role of every partial rule is similar to that of a context-free rule, and the generic rule works as a context-free grammar in which rules has a default interpretation. For this kind of generic rules, that combine syntactic information, it is senseless to consider their interaction with a context-free grammar, as they are parsable by themselves. Such parsing does not differ substantially from context-free parsing. The main restriction is that, due to the functional interpretation of partial rules, bottom-up algorithms are best suited that top-down mechanisms, that would require the definition of inverse functions.

The reduce rule that takes account of syntactic parsing with a generic rule can be described as follows:

$$\textbf{Reduce}\quad \frac{[\alpha \boldsymbol{x}_1 \boldsymbol{x}_2 \bullet, j]}{[\alpha \mathcal{SYN}(\boldsymbol{x}_1, \boldsymbol{x}_2)(\boldsymbol{x}_1, \boldsymbol{x}_2) \bullet, j]} \quad \mathcal{SYN}(\boldsymbol{x}_1, \boldsymbol{x}_2)(\boldsymbol{x}_1, \boldsymbol{x}_2) \neq \perp \tag{7}$$

The expression $\mathcal{SYN}(\boldsymbol{x}_1, \boldsymbol{x}_2)(\boldsymbol{x}_1, \boldsymbol{x}_2)$, with both arguments repeated, may seem confusing. Note, however, that each couple of arguments plays very different roles. The first couple are the arguments of the dynamic binding function, from which an effective partial rule is obtained. Such partial rule is then applied to the second set of arguments. They are related to the informational domain associated to the generic rule, which is, in this case, the type information associated to the linguistic objects.

The side condition, in this case, is precisely the one that licenses the syntactic combining operation. If $\mathcal{SYN}(\boldsymbol{x}_1, \boldsymbol{x}_2)(\boldsymbol{x}_1, \boldsymbol{x}_2)$ does not return a positive result, then we cannot build a phrase out of the arguments $\boldsymbol{x}_1, \boldsymbol{x}_2$.

The computational cost of parsing processes associated to this kind of generic rules carrying on syntactic information is obviously the same as the cost of parsing a context-free grammar. The only difference between both processes is the way to select the appropriate rule when the Reduce step is called. In context-free parsing, this step involves looking up the available rules and matching the objects involved with the right-hand side of the rules. In the worst case, this process introduces a factor G in the overall complexity of parsing, being G the size of the

[3] This would be the case of semantic disambiguation.

grammar. For a generic rule, the reduce rule involves introducing a new type in the hierarchy of rules. Again, this implies a factor G in the overall complexity, being G the number of partial rules associated to the generic rule. The dependence with the length of the string is obviously the same, as the surface behaviour of a context-free grammar and a generic rule is exactly the same for bottom-up parsing.

Another interesting case is when we have a generic rule to specify syntactic restrictions, and another one to perform semantic interpretation. It is easy to specify an algorithm to parse such grammars from the preceding cases. Now the elements of the logic have the form $[\alpha \bullet, j, \Phi]$, and the parser includes the following reduce rule:

**Reduce**

$$\frac{[\alpha x_1 x_2 \bullet, j, \Phi\phi_1\phi_2] \quad \mathcal{SYN}(x_1,x_2)(x_1,x_2) \neq \perp}{[\alpha \mathcal{SYN}(x_1,x_2)(x_1,x_2) \bullet, j, \mathcal{SEM}(x_1,x_2)(\phi_1,\phi_2)] \quad \mathcal{SEM}(x_1,x_2)(\phi_1,\phi_2) \neq \perp} \tag{8}$$

Let us see a (linguistically weird ) example. Consider the grammar made up of the following generic rules:

$$\text{SYN} = \begin{cases} \text{SYN}_{NP \otimes VP}(X,Y) = S \\ \text{SYN}_{NP \otimes VP_i}(X,Y) = S_i \end{cases} \tag{9}$$

$$\text{SEM} = \begin{cases} \text{SEM}_{NP \otimes VP}(X,Y) = sem(X)(sem(Y)) \\ \text{SEM}_{Proper\text{-}Noun \otimes VP}(X,Y) = sem(Y)(sem(X)) \wedge (\lambda P.P(\text{Pete}))(sem(Y)) \end{cases} \tag{10}$$

The syntactic rule has already been considered in section 2.1. We assume the same hierarchy of that example, augmented with a new type $proper\text{-}noun \leq NP$. The semantic generic rule takes account of a very special semantic issue, namely that Pete has a weak personality and imitates his friends in everything they do. If we consider the phrases

- $Betty$ : $\lambda P.P(\text{Betty})$ ; $Betty \leq proper\text{-}noun$
- $got\text{+}angry$ : $\lambda x.\text{ANGRY}(x)$ ; $got\text{+}angry \leq VP_i$

the application of the deductive system above to the string '*Betty got angry*' would be as follows:

1. $[\bullet, 0, ]$
2. $[Betty \bullet, 1, \lambda P.P(\text{Betty})]$ (**Shift**)
3. $[Betty \quad got\text{+}angry \bullet, 2, \lambda P.P(\text{Betty}), \lambda x.\text{ANGRY}(x)]$ (**Shift**)
4. $[S_i \bullet, 2, \text{ANGRY(Betty)} \wedge \text{ANGRY(Pete)}]$ (**Reduce**)

In the only application of the reduce rule, the following terms were used:

$\mathcal{SYN}(Betty, got\text{+}angry)(Betty, got\text{+}angry) = \text{SYN}_{NP \otimes VP_i}(Betty, got\text{+}angry) = S_i$

$\mathcal{SEM}(Betty, got\text{+}angry)(\lambda P.P(\text{Betty}, \lambda x.\text{ANGRY}(x)) =$

$\text{SEM}_{Proper\text{-}Noun \otimes VP}(\lambda P.P(\text{Betty}, \lambda x.\text{ANGRY}(x)) = \text{ANGRY(Betty)} \wedge \text{ANGRY(Pete)}$

This example illustrates the point, stated before, that it is not possible to attach a semantic rule to each syntactic rule at compile time. It is, essentially, a dynamic binding process.

A phrase-structure grammar with a one-to-one correspondence between syntactic and semantic rules would need 12 pairs of rules to get the same behaviour as the two generic rules above. The incrementality and modularity of the generic rule approach is evident in this case.

## 3 Non-Constituent Coordination and Categorial Grammar

The general scheme for coordination corresponds to the conjunction of constituents belonging to the same type: '*Nothing is certain, except death and taxes*', '*Take the money and run*', '*the long and winding road*'. Within the categorial grammar framework, such cases are solved using the basic function application rules (corresponding to the non-associative Lambek calculus in a sequent calculus presentation). Nevertheless, there are cases of the so-called *non-constituent coordination*, where the conjoined expressions are not constituents in the classical sense: '*John met Jane yesterday and Chris today*', '*John read a book about linguistics on Monday and a journal about computers on Tuesday*' (Left-node raising) , '*John made and Peter painted a wooden chair* (Right-node raising). The most common solution is to postulate extra-grammatical levels of representation and/or special purpose parsing algorithms.

There have been a number of proposals within the categorial framework, however, that deal with such phenomena at a grammatical level, extending the number of rules or the set of basic operators. The combinatory rules of type raising and functional composition [Steedman, 1985], for instance, introduce associativity in the structural resources of the grammar. [Dowty, 1988] uses these rules to assign a category to the coordinated conjuncts. With the exception of [Wittenburg and Wall, 1990], that proposes a (less intuitive) normal form for such rules that avoid spurious ambiguity, all the proposals suffer from intractability of the parsing task. Significantly, none of them includes, to our knowledge, a formal differentiation between default and exceptional rules.

We will consider here a particularly simple account for non-constituent coordination phenomena based on the *sequence product* operator introduced in [Solías, 1993]. Its reformulation as a generic rule will show the advantages of expressing default and exceptional grammar rules.

In the non-constituent coordination examples above, we may consider '*Jane yesterday*', '*a book about linguistics on Tuesday*', etc, as being tuples of expressions belonging to the tuple product. [4]. In this case the conjunction scheme type $(x \setminus x)/x$ would substitute a sequence product by the variable $x$.

In order to introduce this operator we need to extend the basic string algebra of types by adding a tuple operation. Thus the algebra would be $(L^*, +, \langle ., . \rangle)$, being $+$ the concatenative and associative operation and $\langle ., . \rangle$ the operation of tuple formation.

The model-theoretic definition for the sequence product is as follows:

$$D(\langle A, B \rangle) = \{< x_1, x_2 >: x_1 \in D(A), x_2 \in D(B)\} \tag{11}$$

The sequent rules corresponding to 11 are:

$$\frac{\Gamma \Rightarrow A \quad \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow \langle A, B \rangle} R\langle\rangle \qquad \frac{\Gamma, A, B, \Delta \Rightarrow C}{\Gamma, \langle A, B \rangle, \Delta \Rightarrow C} L\langle\rangle \tag{12}$$

Examples with more than two elements in the sequence product will need a generalization of the tuple operation. This generalization is straightforward using the standard definition of n-tuple: $< x_1, ..., x_n >=<< x_1, ..., x_{n-1} > x_n >$. Now we are able to account for a sentence like '*John read a book about linguistics on Monday and a journal about computers on Tuesday*' by using a 3-tuple $\langle np, pp, (np \setminus s) \setminus (np \setminus s) \rangle$. Right-node raising examples proceed in a similar way.

This approach avoids using type-raising (a rule that can be applied on any category at any time), but still suffers from intractability, as it is available for every combination of types. Again, the problem relies on the exceptional status that should be given to the rules that deal with non-canonical phenomena.

[4] For these simplest cases of Non-constituent coordination we may also use Lambek's associative product. However the introduction of the sequence product brings advantages in more intricate examples of coordination like the ones considered in [Solías, 1993]. Therefore we will use the sequence product in consideration of further extensions of this work.

## 4 A Generic Rule to Parse Non-Constituent Coordination

The framework of generic rules offers a natural way to express the grammar to deal with non-constituent coordination as an arrangement of default rules, where each combination of types is performed according to the most specific rule available.

The essential rules we want to express are:

- *By default, two types are combined using functional application and, if functional application is not applicable, they cannot be combined.*
- *When we try to combine two or more verbal complements and there is a conjunction inmediately preceding them, a sequence product can be formed with them.*
- *When we try to combine a noun phrase and a verb followed by a conjunction, a sequence product can be formed with them.*

"Verbal complement", for our present purposes, stands for a $np$, a $vm$ or, in turn, a sequence product [5].

Such rules would guarantee that a sequence product is formed only in the relevant cases. We only need an additional rule to scan optimally the elements that match the sequence product on the left-side of the coordination.

The formalism of generic rules allows for a direct specification of such set of - still very informal - rules. Once turned into a generic rule, they can be parsed with any bottom-up context-free recognition algorithm (reformulated as the shift-reduce parsing in section 2.4).

The first step is to express the operations related to the grammar as binary rules:

$$\begin{array}{ll} \mathbf{fa} : X/Y \quad Y \rightarrow \quad X & \mathbf{I}_{\langle\ldots\rangle} : conj \quad X \quad Y \rightarrow \quad conj \quad \langle X, Y\rangle \\ \mathbf{ba} : Y \quad Y\backslash X \quad \rightarrow \quad X & \mathbf{D}_{\langle\ldots\rangle} : \langle X_1 \cdots X_n\rangle \rightarrow X_1 \cdots X_n \end{array} \qquad (13)$$

$$\mathbf{scan} : X_n \quad \langle\langle X_1 \cdots X_{n-1}\rangle\, X_n\rangle \backslash \langle X_1 \cdots X_m\rangle \rightarrow \langle X_1 \cdots X_{n-1}\rangle \backslash \langle X_1 \cdots X_m\rangle$$

**fa** and **ba** are the usual forward and backward application rules. $\mathbf{I}_{\langle\ldots\rangle}$ is the rule to introduce a sequence product. As stated, it seems a context-sensitive rule: The formation of a sequence product is only possible when a conjunction is present to the left of the elements that form a sequence product [6]. However, this context-sensitivity does not overcome context-free grammar parsing, as the licensing element is a lexical item, a terminal, and its presence can be checked in constant time. $\mathbf{I}_{\langle\ldots\rangle}$ rule implements the stepped application of rules $L/$ and $R\langle\rangle$ in the cancellation of type $(\langle A,\ B\rangle \setminus \langle A,\ B\rangle)/\langle A,\ B\rangle \quad A,\ B$. **scan** matches elements to the left of the conjunction with the items in the tuple built to the right. This reformulation is intended to a) keep the binary rules arrangement to allow easy formulation of the generic rule, and b) take into account the asymmetry between the formation process for the right and the left coordinated conjuncts [7] . The necessity of bulding a tuple is given by the right conjunct, and the left conjunct is built only to match the right one. **scan** is implementing the consecutive application of $L\backslash$ and $R\langle\rangle$. $\mathbf{D}_{\langle\ldots\rangle}$ is the rule to eliminate the tuple operator and it implements the $L\langle\rangle$ rule.

The crucial point to write a generic rule based upon the rules above is to determine the types of the arguments associated to each operation. By default, any combination of categories is driven by functional application rules: forward application (**fa**) and backward application (**ba**). Therefore, the first partial rule would be licensed on arguments of the most general type $T$. This rule will try to apply **fa** or **ba** , and will return $\perp$ if both fail, forbidding that combination.

[5] We use $vm$ as an abbreviation for $(np\backslash s)\backslash(np\backslash s)$. In this application we are only taken under consideration the possibility of having types $np$ and $vm$ as verbal complements. This set should be extended if some other kind of complementation were considered.

[6] As [Solías, 1992] pointed out, the conjunction licenses type sequences. Therefore, sequence product can appear in coordination environments, whereas other categories do not.

[7] The stepped cancellation of the right and the left conjuncts has well-known linguistic motivation, first stated in [Ross, 1967].

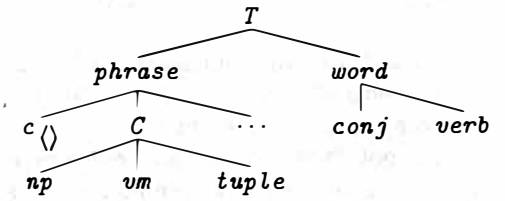
The second partial rule is the rule of tuple formation. The type conditions over the arguments of such a rule arise in a natural way from the informal specification made at the beginning of this section: both tupled elements have to be verb complements (in the case of left-node raising) or an np followed by a verb (in the case of right-node raising). We need two partial rules to establish both type specifications. We will consider a type $C$ that has $np$ and $vm$ as direct subclasses. Another possibility is that one of the tupled members is, in turn, a tuple. Therefore, the tuple has to be introduced as a direct subclass of $C$ as well. That dependency provides the possibility of building n-tuples.

Finally, we need a scan rule to match objects at the left of the conjunction once the right tuple has been combined with the conjunction (by means of simple forward application). Therefore, the type constraint on right elements is type $C$ again. The second one has to be a coordination of tuples missing some elements to its right. We will denote this type as $c_{\langle\rangle}$. This rule has to act in coordination with $\mathbf{D}_{\langle\ldots\rangle}$, which will be applied only after the last conjoined element in the right coordinated conjunct has been cancelled.

To establish the interaction between the rules of 13 and the partial rules, we will use the following operations of composition, disjunction and optionality:

- $r \circ p(x,y) \equiv r(p(x,y))$
- $r \vee p(x,y) \equiv \begin{cases} r(x,y) \text{ if } r(x,y) \neq \bot, p(x,y) = \bot \\ p(x,y) \text{ if } p(x,y) \neq \bot, r(x,y) = \bot \\ \bot \text{ if } p(x,y) = \bot, r(x,y) = \bot \end{cases}$
- $[r](x) \equiv \begin{cases} r(x) \text{ if } r(x) \neq \bot \\ x \text{ if } r(x) = \bot \end{cases}$

That is the fragment of hierarchy that we need for our present purposes:



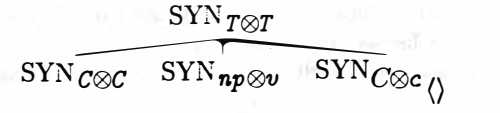

The type $C$ represents verb complements, as introduced above. The type $c_{\langle\rangle}$ represents the combination of a conjunction with a tuple as its right coordinated conjunct. It is needed to specify the scan rule over the appropriate kind of objects. The type $tuple$ is included as a verb complement, to allow formation and coordination of n-tuples.

Given that hierarchy, we propose the following generic rule to parse sentences including non-constituent coordination:

$$\text{SYN} = \begin{cases} \text{SYN}_{T\otimes T}(X,Y) = (\mathbf{fa} \vee \mathbf{ba})(X,Y) \\ \text{SYN}_{C\otimes C}(X,Y) = \mathbf{I}_{\langle\ldots\rangle}(X,Y) \\ \text{SYN}_{np\otimes v}(X,Y) = \mathbf{I}_{\langle\ldots\rangle}(X,Y) \\ \text{SYN}_{C\otimes c_{\langle\rangle}}(X,Y) = (\mathbf{D}_{\langle\ldots\rangle} \circ \mathbf{scan})(X,Y) \end{cases} \tag{14}$$

Note, again, that no precedence has to be defined by the grammar writer to control the interaction of the rules. An easy, natural analysis of the suitable arguments for each rule has implicitly defined a partial ordering between them.

The Hasse diagram of the cartesian poset associated to that generic rule is:



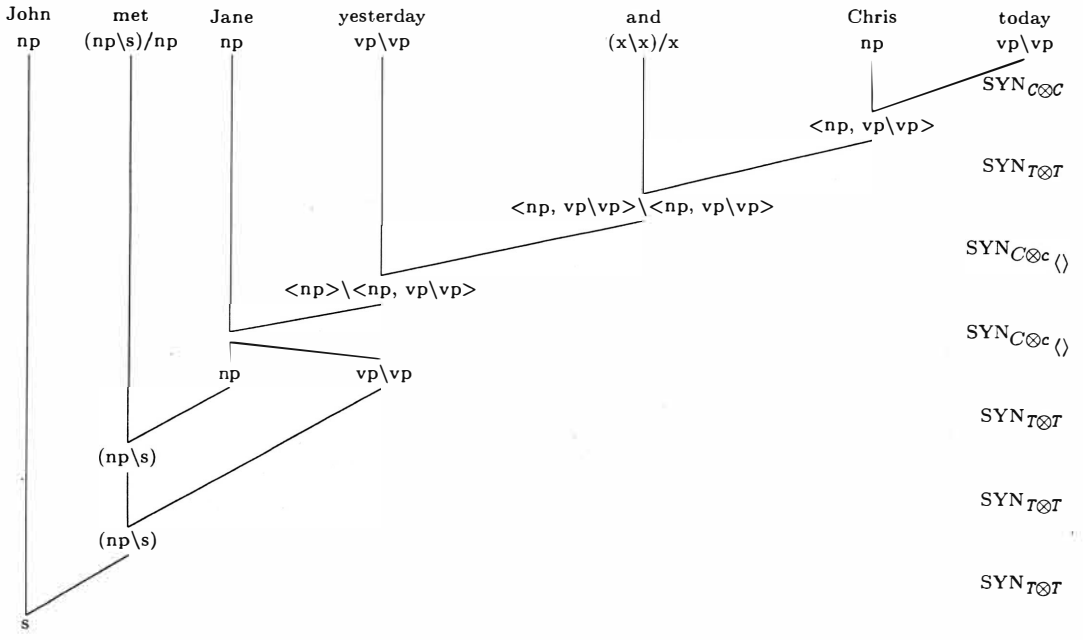



Figure 2: Analysis of "John met Jane yesterday and Chris today" according to SYN.

This rule can be parsed with the algorithm in 7. The only novelty is that some word indexing has to be kept so that the presence of a conjunction inmediately to the right of the arguments can be checked in order to apply $\text{SYN}_{C\otimes C}$ and $\text{SYN}_{np\otimes v}$.

The Figure 2 shows how dynamic binding works to get an analysis of '*John met Jane yesterday and Chris today*'. We have annotated each step of the analysis with the partial rule effectively applied on the constituents being combined. That partial rule is signalled by the dynamic binding function as the most specific to combine the types involved in the combination process.

A context-free parser with the modifications shown in section 2.4 for the **reduce** step can produce this analysis at a context-free cost in time, both in the length of the string and the size of the grammar. Compared with the calculus presented in section 3, the generic rule does not suffer from intractability and can be parsed with well-known, general parsing techniques.

A parser with heuristics or daemons to control coordination processes could achieve a similar efficiency. The clear advantage of a generic rule is that the knowledge that reduces the search space is declaratively introduced at the grammar level, and controlled at an intermediate level between the grammar and the parser (by means of dynamic binding). This enhances linguistic motivation, modularity and incrementality (both to extend the grammar and to control the parsing processes).

## 5 Conclusions

The possibility of formally stating grammar rules with a default interpretation has some advantages from a parsing perspective and from the point of view of knowledge representation. On one side, it provides a declarative and modular way to reduce the search space of parsing processes without altering parsing algorithms with heuristic recipes. On the other side, it provides a linguistically motivated account of exceptional behaviour that is particularly appealing for lexicalized grammar formalisms where the lexicon is the repository of most of the linguistic information, and there are only a few, very general rules that govern linguistic phenomena.

Our account of non-constituent coordination illustrates the advantages of such default arrangements for grammar rules. We have presented a categorial account of non-constituent coordination in which incombinable constituents on both sides of conjunction are treated as tuples of elements by the introduction of a sequence type in the conjunction type. Once we have formed a single tuple constituent, we can combine it with the remaining elements. This approach of non-constituent coordination within Categorial Grammar needs some rules of introduction and elimination of the sequence operator which are not commonly needed for other simpler linguistic phenomena, and that makes the parsing process intractable in its original Lambek-style formulation. When reformulated as a generic rule, the type conditions on the arguments for each rule are used by the dynamic binding process to fire the most appropriate grammar rule at every parsing step. The process turns to be context-free parsable, and exhibits a highly restricted search space.

# References


[Bouma, 1992] Bouma, G. (1992). Feature structures and nonmonotonicity. *Computational Linguistics, 18-2.*

[Carpenter, 1993] Carpenter, R. (1993). Skeptical and credulous default unification with application to templates and inheritance. In *Inheritance, Defaults and the Lexicon.* Cambridge University Press.

[Daelemans et al., 1992] Daelemans, W., De Smedt, K., and Gazdar, G. (1992). Inheritance in natural language processing. *Computational Linguistics, vol 18-2.*

[Dowty, 1988] Dowty, D. (1988). Type raising, functional composition, and non-constituent conjunction. In *Categorial Grammars and Natural Language Structures.* Reidel Publishing Company.

[Gonzalo, 1995] Gonzalo, J. (1995). An object-oriented approach to treat exceptions in grammar. In *Grammar formalisms for NLP.* University of Tuebingen Seminar für Sprachwissenschaft technical report series, to appear.

[Morrill and Solias, 1993] Morrill, G. and Solías, T. (1993). Tuples, discontinuity and gapping in categorial grammar. In *EACL-93.*

[Ross, 1967] Ross, J. (1967). *Constraints on variables in Syntax.* PhD thesis, MIT.

[Shieber et al., 1994] Shieber, S., Schabes, Y., and Pereira, F. (1994). Principles and implementation of deductive parsing. Technical Report TR-11-94, Center for Research in Computing Technology, Harvard University.

[Solías, 1992] Solías, M. (1992). *Gramáticas Categoriales, Coordinación Generalizada y Elisión.* PhD thesis, Universidad Autónoma de Madrid.

[Solías, 1993] Solías, T. (1993). Sequence product, gapping and multiple wrapping. In *Proceedings of the workshop on Linear Logic and Lambek Calculus.*

[Steedman, 1985] Steedman, M. (1985). Dependency an coordination in the grammar of dutch and english. *Language*, 61(3).

[Wittenburg and Wall, 1990] Wittenburg, K. and Wall, R. (1990). Parsing with categorial grammar in predictive normal form. In *Current Issues in Parsing Technologies.* Kluwer Academic Publishers.

# A ROBUST PARSING ALGORITHM FOR LINK GRAMMARS

Dennis Grinberg
John Lafferty
Daniel Sleator
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891, USA
{dennis,lafferty,sleator}@cs.cmu.edu

## Abstract

In this paper we present a robust parsing algorithm based on the link grammar formalism for parsing natural languages. Our algorithm is a natural extension of the original dynamic programming recognition algorithm which recursively counts the number of linkages between two words in the input sentence. The modified algorithm uses the notion of a *null link* in order to allow a connection between any pair of adjacent words, regardless of their dictionary definitions. The algorithm proceeds by making three dynamic programming passes. In the first pass, the input is parsed using the original algorithm which enforces the constraints on links to ensure grammaticality. In the second pass, the total *cost* of each substring of words is computed, where cost is determined by the number of null links necessary to parse the substring. The final pass counts the total number of parses with minimal cost. All of the original pruning techniques have natural counterparts in the robust algorithm. When used together with memoization, these techniques enable the algorithm to run efficiently with cubic worst-case complexity.

We have implemented these ideas and tested them by parsing the Switchboard corpus of conversational English. This corpus is comprised of approximately three million words of text, corresponding to more than 150 hours of transcribed speech collected from telephone conversations restricted to 70 different topics. Although only a small fraction of the sentences in this corpus are "grammatical" by standard criteria, the robust link grammar parser is able to extract relevant structure for a large portion of the sentences. We present the results of our experiments using this system, including the analyses of selected and random sentences from the corpus.

We placed a version of the robust parser on the Word Wide Web for experimentation. It can be reached at URL `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/link/www/robust.html`. In this version there are some limitations such as the maximum length of a sentence in words and the maximum amount of memory the parser can use.

## 1 Introduction

In this paper we present a robust parsing algorithm for the link grammar formalism introduced in [5]. Using a simple extension of the original formalism we develop efficient parsing and pruning algorithms

for extracting structure from unrestricted natural language.

Our approach to robust parsing is purely algorithmic; no modification to the underlying grammar is necessary. We begin by making a generalized definition of what is allowed as a parse. We then assign a non-negative cost to each *generalized parse* in such a way that the cost of a parse which is grammatical with respect to the underlying grammar is zero. The goal of the robust parsing algorithm is then to enumerate all parses of a given sentence that have minimal cost. While this approach to robust parsing is certainly not new, it has a particularly simple and effective realization for link grammar that takes advantage of the formalism's unique properties. In particular, all of the pruning techniques that make link grammar parsing efficient for large natural language grammars either remain unchanged or have natural extensions in the robust parsing algorithm.

The robust algorithm uses the notion of a *null link* to allow a connection between any pair of adjacent words, regardless of their dictionary definitions. The algorithm proceeds by making three dynamic programming passes through the sentence. In the first pass, the input is parsed using the original algorithm which enforces the constraints on links to ensure grammaticality. In the second pass, the total cost of each substring of words is computed, where cost is determined by the number of null links necessary to parse the substring. The final pass counts the total number of parses with minimal cost. Memoization together with pruning techniques enable the algorithm to run efficiently, with theoretic time complexity of $O(n^3)$ for an input of $n$ words.

We have implemented these ideas and tested them by parsing the *Switchboard* corpus of conversational English [1]. This corpus is comprised of approximately three million words of text, corresponding to more than 150 hours of transcribed speech collected from telephone conversations restricted to 70 different topics. Although only a small fraction of the sentences in this corpus are "grammatical" by standard criteria, the robust link grammar parser is able to extract relevant structure for a large portion of the sentences. We present here the results of our experiments using this system, including the analyses of selected and random sentences from the corpus, and the statistics of a typical parsing session.

There is a wide body of literature related to robust parsing. This can be seen immediately by the 200(!) references given by S. Abney in a tutorial titled "Partial Parsing" at the 4th Conference on Applied Natural Language Processing in 1994. While some robust parsing methods depend on specific domains and use semantics as their guide, in this paper we investigate a purely syntactic technique. The advantages of using syntactic techniques include the ability to use the robust parser in varied domains and applications, the ability to use existing grammars with little or no change, and the ability to base techniques on parsing technologies whose characteristics are well understood.

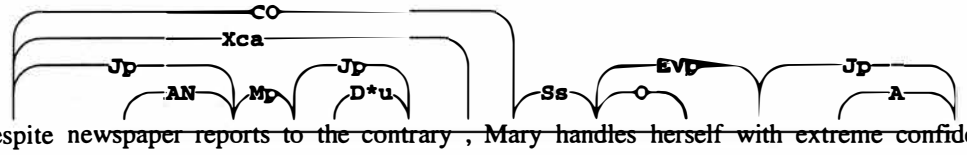
There has been work by a number of researchers on *least-errors recognition* for context-free grammars. In 1974, Lyon [4] proposed a dynamic programming algorithm for finding the least number of mutations, insertions, and deletions of terminal symbols necessary to parse a sentence. Recently, Lee *et al.* [3] extended this work by allowing errors in non-terminals. In [2] Lavie and Tomita describe a modification of the Generalized LR Parser. Their GLR* algorithm is designed to determine the maximal subsets of the input that are parsable by skipping words. While not guaranteed to have cubic running time, a beam search is used to prune parsing options that are unlikely to produce a maximal parse. With pruning, the system is no longer guaranteed to produce the best solution, but the authors report that the beam search works well in practice.

In the following section we briefly review the relevant concepts from link grammar that are necessary for presenting the robust parsing algorithm. In Section 3 we give two definitions of cost. The first is given in terms of the edit distance of a generalized parse and it is analogous to least-errors recognition. The second definition introduces the concept of a null link. Using this definition our approach bears

similarity to the work in [2]. While the edit distance is perhaps a more general and intuitive concept, it does not lead us to efficient parsing algorithms. In Section 4 we give the details of the robust parsing algorithm using a cost function defined in terms of null links. In Section 5 we explain how the pruning techniques of [5] can be extended to accommodate null links. Finally, in Section 6 we present the results of our experiments with the robust parser on the Switchboard corpus.

## 2 Link Grammar Concepts and Notation

In this section we briefly summarize the relevant concepts and notation of link grammar. The figure below represents one of the parses produced by a link grammar parser on the input sentence "Despite newspaper reports to the contrary, Mary handles herself with extreme confidence." The labelled arcs connecting words to other words on their left or right are called *links*. A valid parse is called a *linkage*.




In this figure we see that `confidence` is linked on the left to the adjective `extreme` with a link labelled `A`, denoting an adjective. (We can also say that `extreme` is linked on the right to `confidence`.) Words can have multiple links; for example, The word `handles` has a singular noun (`Ss`) link to its left and object (`O`) and prepositional phrase (`EVp`) links to its right.

A link grammar is defined by a dictionary comprising a vocabulary and definitions of the vocabulary words. These definitions describe how the words can be used in linkages. A word's definition can be thought of as a list of *disjuncts*. Each disjunct $d$ is represented by two ordered lists, written as

$$d = ( (l_1, l_2, \ldots, l_m) \ (r_n, r_{n-1}, \ldots, r_1) )$$

where $l_i$ are *left connectors* and $r_j$ are *right connectors*. (Connector names typically begin with one or more upper case letters followed by sequences of lower case letters and *'s.) For example, from the above linkage we can see that one of the disjuncts in the dictionary definition of `handles` must be ( (`Ss`) (`EVp`, `O`) ).

A word $W$ with disjunct $d = ( (l_1, l_2, \ldots, l_m) \ (r_n, r_{n-1}, \ldots, r_1) )$ can be linked to other words by connecting each $l_i$ to right connectors of words on $W$'s left and also connecting each $r_j$ to left connectors of words on $W$'s right. Links are only permitted between *matching* connectors.

A linkage is specified by choosing a disjunct for each word in the sentence from the word's definition and linking every connector in the disjunct with a connector of a different word's disjunct. A linkage is valid if and only if it meets the following criteria:

- Connectivity: The graph of links and words is connected.
- Planarity: When drawn above the sentence, the links do not cross.
- Ordering: For disjunct $d = ( (l_1, l_2, \ldots, l_m) \ (r_n, r_{n-1}, \ldots, r_1) )$ of word $W$, $l_i$ and $r_j$ are linked to words whose distances from $W$ are monotonically increasing in $i$ and $j$.
- Exclusion: No two links connect the same pair of words.

While the use of disjuncts simplifies the mathematical analysis and parsing algorithms for link grammars, it is cumbersome to express actual grammars in these terms. For this reason the linking

requirements of a word are often expressed as a logical formula involving connectors and the operators & and or. As a simple example, the formula

```
(D- & (O- or S+))
```

represents the two disjuncts ( (D) (S) ) and ( (D, O) () ). The links attached to each word in a linkage must *satisfy* that word's formula. That is, the connectors must make the logical expression represented by the formula true, with the or operator understood to be exclusive.

A convenient data structure for storing and manipulating a link grammar dictionary represents a formula as an *expression tree*, with each node in the tree being either an or-node, an &-node, or a leaf representing a connector. To prepare for parsing a sentence, the expression tree for each word in the sentence is first pruned to eliminate connectors that cannot possibly participate in any linkage. The pruned expression tree is then expanded into a list of disjuncts before *power pruning* is carried out to eliminate disjuncts that necessarily violate one or more of the structural constraints that a valid linkage must obey. After pruning, the parsing algorithm itself is carried out. In Sections 4 and 5 we give the details on the parsing and pruning algorithms that are needed to explain the robust algorithm. In [5] the relationship between link grammar and other grammatical formalisms is discussed.

## 3 The Cost of a Linkage

The approach that we take to robust parsing uses the notion of *cost*. Informally, the idea is to define a set of generalized parse trees with respect to the grammar and the assignment of a number $cost(P \mid S) \geq 0$ to each generalized parse $P$ of a sentence $S$. The task of the robust parsing algorithm is to find all generalized parses having minimum cost. This, of course, is not a new idea; many approaches to robust parsing can be viewed in these terms. Some algorithms minimize the number of ungrammatical "islands," while statistical parsing algorithms for unrestricted text generally use the cost function $cost(P \mid S) = -\log \Pr(P \mid S)$ where $\Pr(P \mid S)$ is given by a probabilistic model. It is important to note that the objective of minimizing a cost function is only a heuristic. For any cost function there are likely to be examples for which "optimal" parses have non-minimal cost.

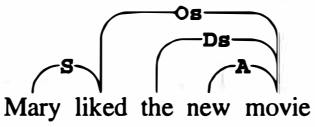
In this section we discuss two definitions of cost for link grammars. The first is called edit distance. The second, which introduces null links, is more restricted in the generalized parses it allows. We have been unable to devise a practical robust parsing algorithm for the edit distance, and thus we only mention it briefly as motivation. We have, however, been successful in designing an efficient algorithm using the more restricted definition of cost.

### 3.1 Edit Distance

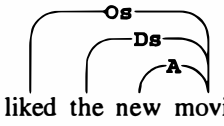
To define a cost function, we first need to define the set of generalized parses that are allowed. The definition of a linkage given in Section 2 requires that the links obey the planarity, connectivity, ordering, and exclusion properties. A set of links that obeys all of these conditions will be called *structurally sound*. If, in addition, the links satisfy the formula of each word in the sentence, the linkage will be called *legal*. We will assign a cost to any structurally sound linkage.

The *edit distance* of a structurally sound linkage is the minimum number of links and words that need to be added, deleted, or renamed in order to create a collection of one or more legal linkages. The edit distance of a string of words is defined to be the smallest edit distance of all structurally sound linkages connecting those words.

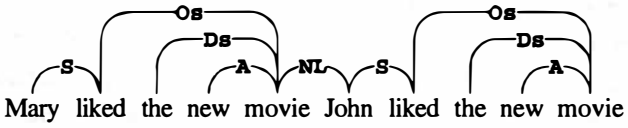
To illustrate these definitions, suppose that the following linkage is legal:




Then the edit distance of the string "liked the new movie" is no more than two, since the linkage above can be constructed from the following structurally sound linkage by adding a word and a link:




Similarly, the string "Mary liked the Fellini movie" has edit distance no greater than one if the word "Fellini" is not in the dictionary. The string "Mary liked the new movie John liked the new movie" also has edit distance no greater than one since the structurally sound linkage

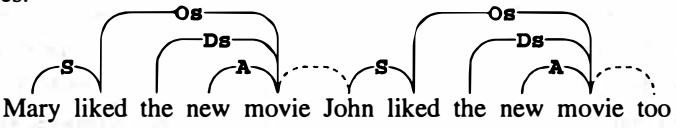



can be reduced to two legal linkages by removing a single link. In general, the edit distance of a string of $n$ words is no more than $2n - 1$. This is because the linkage obtained by linking only adjacent words can be disassembled into the empty linkage by deleting $n - 1$ links and all $n$ words.

Using the original link grammar parsing methods, it is not too difficult to design an algorithm that takes $O(n^3)$ steps to calculate the edit distance of a string of $n$ words. However, we have been unable to find such an algorithm that will run efficiently for a significantly large grammar. The primary reason is that the pruning methods that allow the standard link grammar parsing algorithm to run efficiently do not have natural counterparts to prune the space of structurally sound linkages. However, by using a more restricted notion of edit distance we can design an efficient robust parsing algorithm.

### 3.2 Null Links

A *null link* is an unlabeled link connecting adjacent words. The following linkage has two null links, drawn as dashed arcs.




If both of these null links are deleted, then we obtain two legal linkages together with the disconnected word "too." If a structurally sound linkage has the property that removing its null links results in a collection of one or more legal linkages together with zero or more isolated words, we say that the linkage is *chained.* We define a cost to each chained linkage equal to the number of null links it contains. The cost of a string of words is defined to be the minimum cost of all chained linkages connecting those words. By this measure, the cost of a string of $n$ words can be no greater than $n - 1$, and it is zero only if the string is grammatical. (We do not include a dependence on the number of isolated words since there is no *a priori* reason to favor one chained linkage over another, if they have the same number of null links, simply because it has fewer isolated words when disassembled.)

The set of chained linkages is the same as the set of legal linkages in an extended grammar. To construct this grammar we create a special connector name NL that does not appear in the original

dictionary. For each disjunct

$$d = ((l_1, l_2, \ldots, l_m)\ (r_n, r_{n-1}, \ldots, r_1))$$

in the original dictionary $\mathcal{D}$ we construct the following three disjuncts in the extended dictionary:

$$\begin{aligned} d' &= ((l_1, l_2, \ldots, l_m)\ (r_n, r_{n-1}, \ldots, r_1, \mathtt{NL})) \\ d'' &= ((\mathtt{NL}, l_1, l_2, \ldots, l_m)\ (r_n, r_{n-1}, \ldots, r_1)) \\ d''' &= ((\mathtt{NL}, l_1, l_2, \ldots, l_m)\ (r_n, r_{n-1}, \ldots, r_1, \mathtt{NL})). \end{aligned}$$

In addition, we add the following three disjuncts to the definitions of each word:

$$((\mathtt{NL})\ ()), \quad (()\ (\mathtt{NL})), \quad \text{and} \quad ((\mathtt{NL})\ (\mathtt{NL})).$$

This defines the extended dictionary $\mathcal{D}'$. Each link labeled NL in a linkage of the extended grammar corresponds to a null link. The removal of such links results in one or more legal linkages in the original grammar, together with zero or more disconnected words.

There are several advantages to using the restricted definition of cost described above. As we will show in the next two sections, there are natural extensions of the original parsing and pruning algorithms that accommodate null links. These extensions are easily implemented, and they lead to a robust parsing algorithm that is efficient and practical. Moreover, there is no need to explicitly construct the extended grammar just described; in fact, the grammar does not need to be modified in any way. Thus, null links represent virtual rather than physical links. Most importantly, as we indicate in Section 6, experimentation with the robust parser using null links has shown that it is capable of extracting relevant grammatical structure even in ungrammatical text such as transcriptions of spontaneous speech.

## 4 The Robust Parsing Algorithm

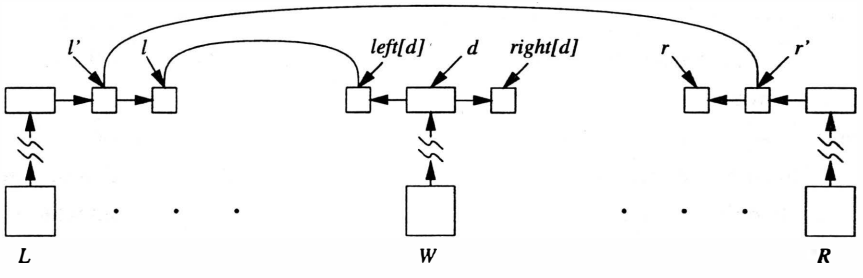
In this section we describe an algorithm for calculating the cost of a sentence. The algorithm determines the minimum number of null links necessary to parse the sentence by making three dynamic programming passes. In the first pass the input is parsed using the original algorithm which counts the number of legal linkages. If the sentence is grammatical then this is the only pass that is carried out. If not, a second pass is made to calculate the minimum number of null links necessary to parse each substring. Since the results of the first pass are memoized, only regions of the input sentence that are not grammatical need to be explored in this pass. The final pass counts the total number of parses with minimal cost.

The robust parsing algorithm is most easily understood by first considering the problem of counting *all* chained linkages, not just those with the fewest possible null links. To explain how this calculation is carried out we need to review the original link grammar parsing algorithm, which proceeds by recursively counting parses in a top-down fashion. Consider the situation after a link has been proposed between a connector $l'$ on word $L$ and a connector $r'$ on word $R$. For convenience, we define $l$ and $r$ to be $next[l']$ and $next[r']$ respectively.

In order to attach the words of the region $(L, \ldots, R)$ strictly between $L$ and $R$ to the rest of the sentence, there must be at least one link either from $L$ to some word in this region, or from $R$ to some word in this region (since no word in this region can link to a word outside of the $[L, \ldots, R]$ range, and something must connect these words to the rest of the sentence). Since the connector $l'$ has already been used in the solution being constructed, this solution must use the rest of the connectors of the disjunct in which $l'$ resides. The same holds for $r'$. The only connectors of these disjuncts that can be involved in the $(L, \ldots, R)$ region are those in the lists beginning with $l$ and $r$. (The use of any other connector on

these disjuncts in this region would violate the ordering requirement.) In fact, all of the connectors of these lists must be used in this region in order to have a satisfactory solution.

Suppose that $l$ is not NIL. We know that this connector must link to some disjunct on some word in the region $(L, \ldots, R)$. The algorithm tries all possible words and disjuncts. Suppose it finds a word $W$ and a disjunct $d$ on $W$ such that the connector $l$ matches $left[d]$. We can now add this link to our partial solution. The situation is shown in the following diagram.




Here the square box above $L$ represents the data structure node corresponding to the word $L$. This points to a list of disjuncts, each of which is shown as a rectangular box above the word. The disjunct in turn points to two linked lists of connectors, shown here as small square boxes.

The algorithm determines if this partial solution can be extended to a full solution by solving two problems similar to the original problem. In particular, the word range $(L, \ldots, W)$ is extended using the connector lists beginning with $next[l]$ and $next[left[d]]$. In addition, the word range $(W, \ldots, R)$ is extended using the connector lists beginning with $right[d]$ and $r$. The case where $l$ = NIL is similar.

To explain the robust algorithm it will be helpful to repeat a fragment of the pseudocode representing the original algorithm as given in [5]. The function COUNT takes as input indices of two words $L$ and $R$, where the words are numbered from 0 to $N - 1$, and pair of connector lists $l$ and $r$. COUNT returns the number of different ways to draw links among the connectors on the words strictly between $L$ and $R$, and among the connectors in lists $l$ and $r$.

```
PARSE
1  t ← 0
2  for each disjunct d of word 0
3      do if left[d] = NIL
4          then t ← t + COUNT(0, N, right[d], NIL)
5  return t

COUNT(L, R, l, r)
1  if R = L + 1
2    then if l = NIL and r = NIL
3           then return 1
4           else return 0
5  else total ← 0
6    for W ← L + 1 to R − 1
7        do for each disjunct d of word W
8           ...
```

The algorithm counts all linkages by recursively counting smaller and smaller regions. Lines 1–4 of the COUNT procedure handle the boundary conditions. If word $R$ is adjacent to $L$ then there is no solution unless $l = r$ = NIL. This ensures that all of the connectors are used for every disjunct that participates

in a linkage. If $L$ and $R$ are not neighboring words then the algorithm examines each intervening word $W$, beginning at line 6.

To explain how null links are introduced we need to first closely examine the situation where $l = \text{NIL}$ and $r \neq \text{NIL}$. In this case, to extend the linkage into the region $(L, \ldots, R)$, a word $W$ must be chosen to link with $r$. This leads to a test of whether the region $(L, \ldots, W)$ can be extended using the connector lists NIL for $L$ and $left[d]$ for some disjunct $d$ of $W$. Assuming a leftmost derivation, this process continues until either $left[d] = \text{NIL}$ or $W = L + 1$. If $left[d] = \text{NIL}$ then there is no solution in the original algorithm except when the boundary condition $W = L + 1$ holds. In the robust algorithm, however, there is always a way of continuing.

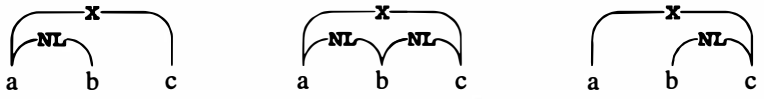
Note that in case $l = \text{NIL}$ when parsing with respect to the original dictionary $\mathcal{D}$, we can always replace $l$ by NL for the extended grammar $\mathcal{D}'$. Since NL connectors can only link adjacent words, this means that $L$ could only be connected to $W = L + 1$ with a disjunct $d$ whose connector $left[d]$ is NL in the extended grammar. The robust algorithm makes such links "virtually," without actually parsing the extended grammar. If $R > L+1$ and $l = r = \text{NIL}$, the algorithm considers each disjunct $d$ of $W = L+1$ having the property that $left[d] = \text{NIL}$. For each such disjunct, a null link is made between $L$ and $W$ and then the region $(W, \ldots, R)$ is parsed using the connector list beginning with $right[d]$ for $W$. Another way of proceeding is to make a null link between $L$ and $W$ and parse the region $(W, \ldots, R)$ using the connector NIL for $W$. This corresponds to using the disjunct ( (NL) (NL) ) for $W$ in the extended grammar.

The counting algorithm just outlined is given more formally in the pseudocode below. The code after line 12 is identical to that of the original algorithm.

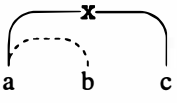
```
PARSE
 1  t ← 0
 2  for each disjunct d of word 0
 3      do if left[d] = NIL
 4          then t ← t + COUNT(0, N, right[d], NIL)
 5  t ← t + COUNT(0, N, NIL, NIL)
 6  return t

COUNT(L, R, l, r)
 1  if R = L + 1
 2      then if l = NIL and r = NIL
 3              then return 1
 4              else return 0
 5  elseif l = NIL and r = NIL
 6      then total ← 0
 7              for each disjunct d of word L + 1
 8                  do if left[d] = NIL
 9                      then total ← total + COUNT(L + 1, R, right[d], NIL)
10              total ← total + COUNT(L + 1, R, NIL, NIL)
11              return total
12  else ...
```

Note that the boundary conditions in lines 1–4 are identical to those of the original algorithm. In the case where $l = r = \text{NIL}$ and $R > L + 1$ the original counting procedure returns 0. This is where null links are introduced in the robust algorithm.

This algorithm returns a number that is, in general, less than the number of legal linkages in the extended grammar; that is, it only counts a subset of all chained linkages. The reason is that the extended grammar introduces NL links symmetrically, and this symmetry is not "broken" by the parsing algorithm. For example, if one of the following linkages is legal in the extended grammar then all three of them are:




In the algorithm given above, however, a null link can be introduced in this case only as follows:




This inconsistency is easily remedied. The algorithms will return the same number of chained linkages if the original parsing algorithm for the extended grammar is modified so that in the case where $l = \texttt{NL}$ and $r = \texttt{NL}$, a connection is only allowed to $l$.

Knowing the total number of chained linkages is of limited use. This collection of parses includes many that have little or no structural information, such as the linkage that joins each word with its immediate neighbors by null links. Using the ideas just described, however, we can count the number of linkages that have the minimal number of null links. We first modify the algorithm to compute the cost of each span. To do this, the increment operations are replaced by "min" operations. The following pseudocode demonstrates how this is carried out.

```
CALCCOST
1  c ← ∞
2  for each disjunct d of word 0
3      do if left[d] = NIL
4          then c ← MIN(c, COST(0, N, right[d], NIL))
5  c ← MIN(c, COST(0, N, NIL, NIL))
6  return t

COST(L, R, l, r)
 1  if R = L + 1
 2     then if l = NIL and r = NIL
 3             then return 0
 4             else return ∞
 5  elseif l = NIL and r = NIL
 6     then cost ← ∞
 7          for each disjunct d of word L + 1
 8              do if left[d] = NIL
 9                 then cost ← MIN(cost, COST(L + 1, R, right[d], NIL) + 1)
10          cost ← MIN(cost, COST(L + 1, R, NIL, NIL) + 1)
11          return cost
12  else . . .
```

Note that while the cost is initialized to be infinite on line 1 of CALCCOST and line 6 of COST, the cost of a string of $n$ words can be no greater than $n - 1$. The code after line 12 is modified from the original

code in the obvious way. If a region $(L, \ldots, R)$ with connector lists $l$ and $r$ is grammatical, this fact will have been recorded (memoized) during the first pass which carries out the standard algorithm. In this case the cost is zero, and there is no need to carry out the second pass on the span. Thus, the second pass only needs to explore those regions that are ungrammatical.

In the third pass, the number of chained linkages with minimal cost is calculated in a manner similar to the way in which the second pass is carried out. However, rather using than a "min" operation, a count is incremented if a given span has minimum cost. The details are omitted.

## 5 Pruning Techniques

The original link grammar parsing algorithm is made practical by means of several pruning techniques, including expression pruning, the fast-match data structure, power pruning, and conjunction pruning. This section shows how each of these techniques has a simple and natural extension to the robust parsing algorithm.

### 5.1 Pruning

Suppose that a word $W$ has a disjunct $d$ with a connector `C` in its list of right connectors. In order for $d$ to be used for $W$ in a linkage of a given sentence, some word to the right of $W$ must have a disjunct with a connector that matches `C` in its left list. This simple observation is the basis for an algorithm to prune the set of disjuncts that are considered by the parsing algorithm.

The pruning algorithm alternately makes sequential left-to-right and right-to-left passes through the words in the sentence. In a left-to-right pass, a set of right connectors is maintained, and is initially empty. The pruning step for word $W$ consists of examining each disjunct $d$ of $W$ that has survived previous pruning passes. If a left connector of $d$ does not match any connector in the set, then $d$ is discarded. Otherwise, the connectors in the right list of $d$ are added to the set. After all disjuncts of $W$ have been processed in this way, the algorithm advances to the next word. The right-to-left pass is analogous.

Pruning can be carried out more efficiently when applied to the entire expression tree representing the formula of a word. In a left-to-right expression pruning pass, each left connector in a word's expression tree is checked to see if it has a match in the set of right connectors. If not, the connector and those nodes that depend on it through & relations are pruned from the tree.

When parsing with null links, the pruning algorithm can be used without modification. Viewing null links in terms of the extended grammar, it can be seen that a disjunct will never be pruned because of the lack of a matching `NL` connector (excepting boundary cases where there is no word to the left or right). In other words, applying pruning to the original grammar and then extending the surviving disjuncts with `NL` connectors results in the same set of disjuncts as applying the pruning algorithm to the extended grammar.

### 5.2 The Fast-Match Data Structure

The fast-match data structure is used in the parsing algorithm to quickly determine which disjuncts of a word might possibly match a given left or right connector. The data structure uses two hash tables for each word in the sentence. The disjuncts that survive pruning are hashed, and disjuncts that hash to the same location are maintained in a linked list. The left table for a word uses a hash function that depends only on the first connector on the left list of the disjunct. The right table has an analogous hash function. When parsing a span $(L, \ldots, R)$ by attempting to link the connector lists $l$ and $r$ to an intermediate word $W$, the fast-match data structure allows a list of candidate disjuncts for $W$ to be quickly obtained by

forming the union of the lists obtained from the left table of $W$ by hashing $l$ and the right table of $W$ by hashing $r$.

A simple modification to the fast-match data structure can be made to accommodate null links. As discussed in the previous section, when null links are constructed "virtually" rather than by explicitly parsing the extended grammar, a null link is only made in the case where the region $(L, \ldots, R)$ is extended using connector lists $l$ and $r$ which are both NIL. In this case, the only disjuncts of word $W = L + 1$ that are considered are those with empty left connector lists. Thus, to modify the fast-match data structure it is only necessary to maintain, for each word, a list of disjuncts $d$ for which $left[d] =$ NIL.

### 5.3 Power Pruning

After expression pruning is carried out, each word's pruned expression tree is expanded into a list of disjuncts. At this point power pruning is carried out in an attempt to enforce the structural constraints imposed on valid linkages, including the ordering, connectivity, planarity, and exclusion properties. As with pruning, power pruning is carried out by making alternating left-to-right and right-to-left passes through the sentence; the details are given in [5].

The power pruning algorithm eliminates some disjuncts by observing that in order for a linkage to be connected, it is necessary that any two connectors between *non-neighboring* words cannot both be the last connectors of their respective lists. This condition is relaxed when parsing with null links. In fact, a null link is *only* formed after two non-neighboring words are connected using the last connectors in their lists. All of the other conditions that power pruning checks remain valid. Thus, it is a simple matter to modify the power pruning algorithm to allow for the possibility of null links.

In the case where the sentence contains a conjunction, a variation of the power pruning algorithm enables pruning of disjuncts before *fat connectors* are built [5]. This variation uses the notion of a *deletable region.* A substring of words is said to be a deletable region if its removal would result in a grammatical sentence. For example, in the sentence *the dog and cat ran*, both *dog and* and *and cat* are deletable regions. In essence, conjunction pruning proceeds by parsing the sentence but allowing for the excision of one or more deletable regions. Each disjunct that participates in such a parse is marked; all unmarked disjuncts can be pruned away. This strategy is again easily extended to the situation where we allow null links. This is done by simply using the robust parsing algorithm when marking disjuncts, and allowing any region of words to be deletable.

## 6 Experimental Results

The Switchboard corpus was created to allow standard evaluations and provide a stable research environment for large vocabulary continuous speech recognition. The corpus contains an abundance of phenomena associated with conversational language: non-standard words, false starts, stutters, interruptions by other speakers, partial words, grammatical errors, changes in grammatical structure mid-sentence, hesitation sounds, and non-speech sounds.

This section describes the results of an experiment in which we applied our robust parser to a randomly chosen subset of the Switchboard corpus. Our purpose in doing this experiment was twofold. First, we wanted to determine whether the robust parsing algorithm, built upon a grammar that attempts to describe "correct" English, would be able to glean useful structure from such "incorrect" text. Secondly, we wanted to measure the time efficiency of the parser. It was not our purpose to give an objective measure of the quality of the grammar, or to make a direct comparison with other robust parsing methods.

| | | |
|---|---|---|
| sentences | # not skipped | 909 (56%) |
| | # skipped (too short) | 706 (44%) |
| | # grammatical | 202 (22%) |
| | # with unknown words | 115 (13%) |
| words | # not skipped | 12047 (89%) |
| | # skipped (too short) | 1452 (11%) |
| linkages | Average # per sentence | 114 |
| | Average # of null links | 1.99 |
| time | Total wall time (seconds) | 3143 |
| | Maximum for one sentence | 273 |

Other than the entries listing the number of sentences and words skipped, skipped sentences do not participate in the calculation of the other entries. The total wall time is the elapsed time needed to compute and sort all linkages for each sentence on a DEC AXP 3000/600 Alpha workstation.

Table 1: The results of the robust parser on Switchboard data

Before parsing the text, we removed all punctuation and case information. We also combined some commonly occuring word pairs (such as "you know") into single words; however, the dictionary was not modified to account for these new words. These changes make the problem of parsing the Switchboard corpus closer to that of analyzing speech.

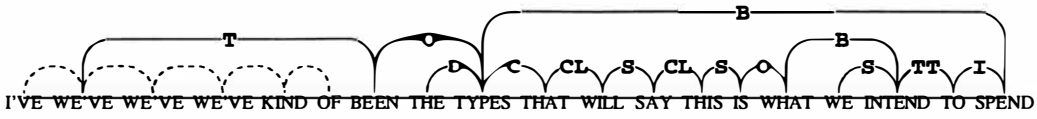
One problem that emerged in trying to apply the parser to this corpus was the lack of sentence boundaries. The parser expects its input to be broken into "sentence size" blocks. A natural approach is to use changes of speaker identity to partition the input. We will call such a block of text an *utterance*. Some utterances are very long, and require a lot of time and memory to parse if fed to the parser as a single sentence. We explored two ways to deal with this problem. The first (reported below) is merely to split long utterances into shorter groups of words (that we will call sentences) that can be parsed using reasonable resources. This loses some grammatical information. But with even an unsophisticated splitting algorithm the parser still seems to discover useful structure. Another method to alleviate the high cost of parsing long sentences is to limit the length of the longest link permitted.

In the experiment reported here, we chose 1500 utterances at random from the Switchboard corpus. Each utterance was split into pieces no longer than 25 tokens each. (The number of words in an sentence after the splitting can still be greater than 25 because the parser considers some tokens, such as possessives, to be more than one word.) We also skipped all sentences with fewer than four words. The results of applying the robust parser to this data are summarized in Table 1.

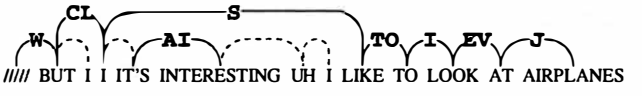
Somewhat surprisingly, given that only 22% of the corpus's sentences were found to be grammatical, only two null links were needed on average to parse the corpus. In order to better understand this statistic we randomly permuted the 13,499 words in the 1617 sentences we previously used, preserving the lengths of the sentences. Of these randomly generated sentences, only 8 (0.89%) were grammatical, and on average, 6 null links were needed to parse them.

Perhaps the most conspicuous feature that distinguishes the utterances in the Switchboard data from more grammatical text is the frequent occurrence of repeated words. The following example contains two repeats of the word WE'VE, in addition to a false start. (A description of the connectors used is given in an appendix.) The robust parser successfully ignores the false start by enforcing the constraint of subject-verb agreement. The three occurrences of WE'VE are interchangeable, and this results in three

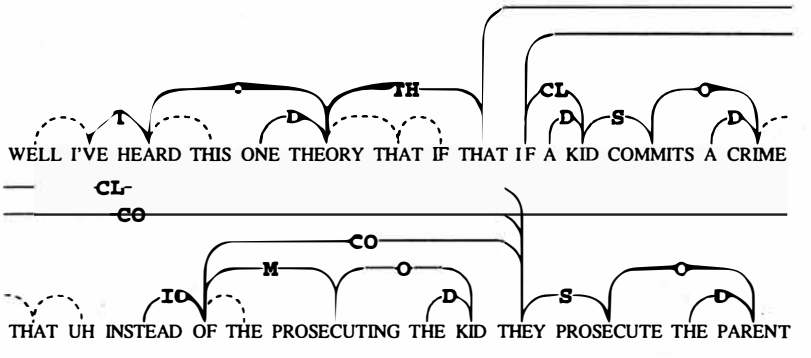
equivalent parses for every use of this word.




It is also common in this data for an utterance to include an embedded phrase that is grammatical in isolation but not with respect to the larger utterance. As the following example demonstrates, the robust parser can handle this by connecting the embedded phrase to the larger utterance with a null link. There were 5 other linkages of this utterance that are essentially equivalent to the one shown below.




In many cases the utterance is ungrammatical, yet by minimizing the number of null links the parser is able to recover the relevant structure. The following example fails to be grammatical because of false starts and the sequence of words INSTEAD OF THE PROSECUTING.

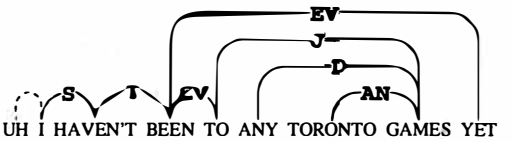



### 6.1 Random Sentences

We now present parses of ten randomly chosen sentences from the corpus.

1. UH I HAVEN'T BEEN TO ANY TORONTO GAMES YET

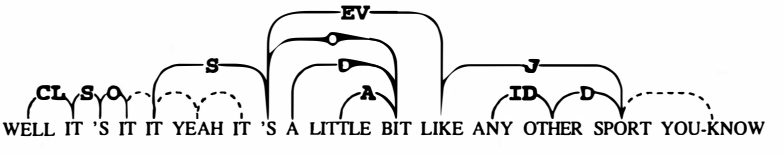
There are 6 parses for this sentence having a single null link.




The word TORONTO is not in the dictionary. When a word is not defined, the parser assigns a default list of disjuncts to the word. In this case the context is used to correctly determine that the unknown word functions as an adjective.

2. WELL IT'S IT IT YEAH IT'S A LITTLE BIT LIKE ANY OTHER SPORT YOU_KNOW

This sentence has six linkages with four null links.

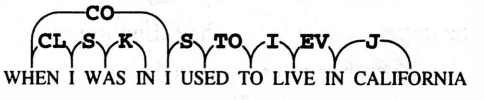



3. UH ARE YOU PRESENTLY LOOKING UH FOR A USED CAR

One linkage with two null links.

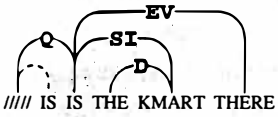

4. WHEN I WAS IN I USED TO LIVE IN CALIFORNIA

One linkage with no null links.




5. IS IS THE KMART THERE

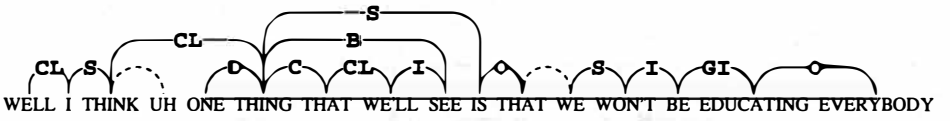
Two linkages differing only in which instance of IS is used.




The word KMART is not in the dictionary, but the unknown word mechanism takes advantage of the context to choose an appropriate disjunct.

6. WELL I THINK UH ONE THING THAT WE'LL SEE IS THAT WE WON'T BE EDUCATING EVERYBODY

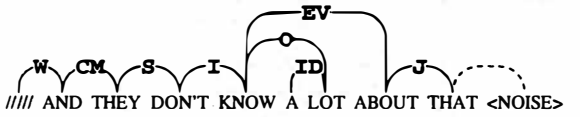
One linkage with two null links.




The parser incorrectly splits this sentence into WELL I THINK ONE THING THAT WE'LL SEE IS THAT and WE WON'T BE EDUCATING EVERYBODY.

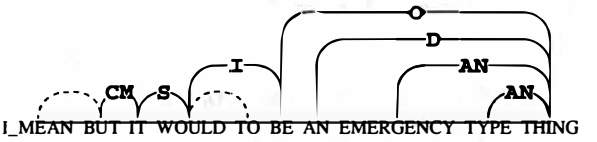
7. AND THEY DON'T KNOW A LOT ABOUT THAT <NOISE>

Five linkages with one null link. They differ in whether ABOUT modifies A LOT or KNOW, whether A LOT is an object or an adverb and whether or not to treat A LOT as an idiom. The correct linkage is shown below.



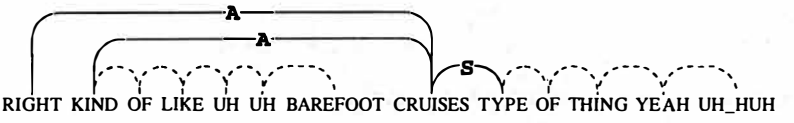

8. I_MEAN BUT IT WOULD TO BE AN EMERGENCY TYPE THING

One linkage with two null links.



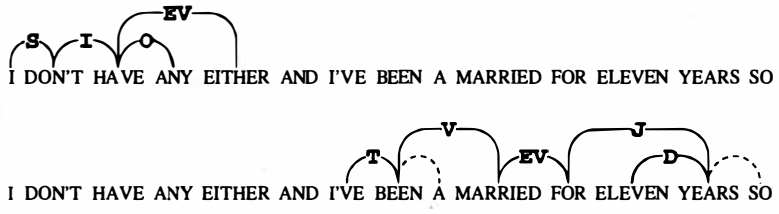

9. RIGHT KIND OF LIKE UH UH BAREFOOT CRUISES TYPE OF THING YEAH UH_HUH

Five similar linkages with nine null links.



10. I DON'T HAVE ANY EITHER AND I'VE BEEN A MARRIED FOR ELEVEN YEARS SO

Two linkages with two null links. The following linkage treats AND as a conjunction.




### 6.2 Analysis

The mixed-case version of the parser treats words beginning with uppercase characters as proper nouns. However, since there is no case information available in our data most proper nouns must be treated as unknown words. The use of a generic definition for unknown words often interprets these words correctly, taking clues from the surrounding context.

Minimizing the number of null links in a linkage, while a fairly good heuristic to produce good parses, is not foolproof. For example, we might interpret example 4 as a false start of WHEN I WAS IN followed by I USED TO LIVE IN CALIFORNIA. The parser cannot mimic this interpretation because to do so would require more than the minimum number of null links. Example 6 demonstrates a deficiency of the grammar, and suggests that a learning algorithm would be useful to infer new word usages and grammatical relations. Similarly, example 8 shows that the parser can not infer missing words. We would like the parser to realize that HAVE should be inserted between WOULD and TO. As we see from example 9, the parser can utterly fail at producing useful structure.

The robust parser has limitations on the input it can parse. For the Switchboard domain, splitting long sentences into shorter ones does not seem to cause significant problems but on other domains it might. Limiting the length of the longest link permitted can allow the system to parse long utterances in reasonable time, with realistic memory limitations. While the resulting linkages might not be as good as ones produced without any limitation, the tradeoff of time and space versus accuracy may be worthwhile for many applications.

## References


[1] J. Godfrey, E. Holliman, and J. McDaniel. Switchboard: Telephone speech corpus for research development. In *Proc. ICASSP-92*, pages I–517–520, 1992.

[2] A. Lavie and M. Tomita. Parsing unrestricted sentences using a generalized LR parser. In *Proceedings of the Third International Workshop on Parsing Technologies*, pages 123–134, 1993.

[3] K. J. Lee, C. J. Kweon, J. Seo, and G. C. Kim. A robust parser based on syntactic information. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, 1995.

[4] G. Lyon. Syntax-directed least-errors analysis for context-free languages: A practical approach. *Communications of the ACM*, 17(1):3–14, Jan 1974.

[5] D. D. K. Sleator and D. Temperley. Parsing English with a link grammar. Technical Report CMU-CS-91-196, School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, 1991.

# AN IMPLEMENTATION OF SYNTACTIC ANALYSIS OF CZECH *

Tomáš Holan[1],Vladislav Kuboň[2], Martin Plátek[3]

[1] Department of Software and Computer Science Education
[2] Institute of Formal and Applied Linguistics
[3] Department of Theoretical Computer Science
Faculty of Mathematics and Physics
Charles University, Prague
Czech Republic
holan@ksvi.ms.mff.cuni.cz
vk@ufal.ms.mff.cuni.cz
platek@kki.ms.mff.cuni.cz

**Abstract.** This paper describes current results achieved during the work on parsing of a free-word-order natural language (Czech). It contains theoretical base for a new class of grammars — CFG extended for dependecies and non-projectivities — and also the description of the implementation of a parser and grammar-checker. The paper also describes some typical problems of parsing of free-word-order languages and their solutions (or discusssion of those problems), which are still subject of investigation.
The implementation described here serves currently as a testing tool for the development of a large scale grammar of Czech. Some of the quantitative data from a processing of test sentences are also included.

## 1 Introduction

All practically oriented linguistically based systems of natural language processing have to solve, to a certain extent, the problem of complex, linguistically adequate, correct, robust, fast and effective parsing of a particular language. Some of these requirements are contradictive — the more complex, correct, robust and linguistically adequate the system, the slower and ineffective it is.

It is quite clear that most of the interpreters of current linguistic formalisms are not effective enough to be used in a practical application, which requires "real time" processing of large quantities of texts. On the other hand, the means used in the parsing of formal languages (especially context-free languages) are effective enough, but are not prepared to deal with certain features of natural languages adequately, as for example long distance dependencies, syntactic and morphemic ambiguities etc.

During our work on the pivot implementation of a grammar checker of Czech we have tried to develop a formalism linguistically adequate and also effective enough. Our task was complicated by the nature of the language under consideration — Czech is a language with highly free word order.

* The work on this paper is a part of the joint research project PECO 2824 "Language Technologies for Slavic Languages" supported by EC and of the project "Automatic checking of grammatical errors in Czech" supported by a grant of the Grant Agency of the Czech Republic No.0475/94.

The authors are also grateful to Karel Oliva and Vladimír Petkevič for several stimuli given to us in the course of our work on the issues elaborated in the paper.

## 2 Non-projective Context-free Dependency Grammars

The standard CF grammars, as used for the description of formal languages, can not describe some constructions containing relations between non-neighbouring symbols. In the framework of the dependency theory these constructions are called non-projective constructions. In this chapter we introduce a class of formal grammars capable to describe these syntactic constructions.

### 2.1 Definition of NCFDG

In the following definition we have chosen a certain normal form of the grammar. The reason for this is the shape of the input of our parser. Two special symbols, called sentinels, are added to every original sentence, one from the left and the other from the right. Let us note that in our application we consider the categories computed by the morpholexical analysis as terminals.

**Definition 1.** Non-projective context-free dependency grammar (NCFDG) *is a quadruple* $(N, T, S, P)$, *where* $N, T$ *are sets of nonterminals and terminals,* $S \in N$ *is a starting symbol and* $P$ *is a set of rewriting rules of the form* $A \rightarrow_L BC$ *or* $A \rightarrow_R BC, A \in N, B, C \in V$ *where* $V = N \cup T$.

*The relation of immediate derivation* $\Rightarrow$ *is defined as:*
$rAst \Rightarrow rBsCt$, *if* $(A \rightarrow_L BC) \in P$
$rsAt \Rightarrow rBsCt$, *if* $(A \rightarrow_R BC) \in P$,
*where*

$$A \in N, B, C \in V, r, s, t \in V^*$$

*The relation of derivation is the transitive and reflexive closure of the relation* $\Rightarrow$.

*NCFD grammar* $G$ *defines language* $L(G)$ *as a set of all words* $t \in T^*$ *that can be derived from the starting symbol* $S$. *We say that* $L(G)$ *is recognized (generated) by* $G$.

**Remark 2.** *We can impose certain limitations on the defined language by minor changes of the definition of the relation* $\Rightarrow$. *For example, the condition* $s = EmptyString$ *reduces the relation* $\Rightarrow$ *to derivation without nonprojectivities — i.e. the same as in the standard CFGs.*

We work with fixed restrictions on the form of the rules. Variations of the type of languages are then obtained only through different types of derivation.

**Definition 3.** *A Tree of a word* $a_1 a_2 \ldots a_n \in T^*$ *dominated by the symbol* $X \in V$, *created by NCFDG* $G$ *is a binary branching tree* $Tr$ *fullfilling the following conditions*

*a) a node of* $Tr$ *is a triad* $[A, i, j]$, *where* $A \in V$ *and* $i, j \in 1 \ldots = n$. *The number* $i$ *is the horizontal index of the node and* $j$ *is the vertical index.*
*b) a node* $[A, i, m]$ *of* $Tr$ *has daughters* $[B, j, p]$, $[C, k, r]$ *only if*
  *1)* $j < k$, $m = p + 1$, $j = i$ *and* $(A \rightarrow_L BC) \in P$ *or*
  *2)* $j < k$, $m = r + 1$, $k = i$ *and* $(A \rightarrow_R BC) \in P$
*c) a root of* $Tr$ *is such a node of* $Tr$ *which has no mother.*
  *We can see that the root has a form* $[X, i, m]$ *for some* $i, m \in 1 \ldots n$.
*d) leaves of* $Tr$ *are exactly all nodes* $[a_i, i, 1], i \in 1 \ldots n$.

**Remark 4.** *For the sake of simplicity we are going to use in the following text the simple term Tree instead of the term Tree dominated by the symbol* $S$ *(where* $S$ *is a starting symbol).*

There are two differences between *Tree* and a standard parsing (derivation) tree of CFG. The first one is that a (complete) subtree of *Tree* may cover non-continuous subset of the input sentence. Second difference is that *Trees* contain enough information for building dependency trees. The basic type of the output of our parser is the dependency tree. Some constraints for our grammar-checker are introduced with the help of trees.

**Definition 5.** *Let $Tr$ be a Tree of a word $a_1 a_2 \dots a_n \in T^*$ created by a NCFDG $G$. The* dependency tree *$Dep(Tr)$* to *$Tr$ is defined as follows: The set of nodes of $Dep(Tr)$ is the set*

$$\{[a, i]; \textit{there is a leaf of } Tr \textit{ of the form} [a, i, 1]\}.$$

*The set of edges of $Dep(Tr)$ is the set of pairs*

$$([a, i], [b, j]),$$

*so that $[a, i, 1], [b, j, 1]$ are two different leaves of $Tr$ and there is an edge $([A, i, p], [B, j, r])$ of $Tr$ for some $A, B, p, r$.*

**Observation 6.** *The language*

$$L = \{w \in (a, b, c)^*; \textit{the number of the symbols } a, b, c \textit{ contained in } w \textit{ is equal}\}$$

*may be recognized by a NCFDG $G_1$ and is not context-free. $G_1 = (N, T, S, P)$, $T = \{a, b, c\}$, $N = \{T, S\}$, $P = \{S \to_L aT|Ta|SS, T \to_L bc|cb\}$*

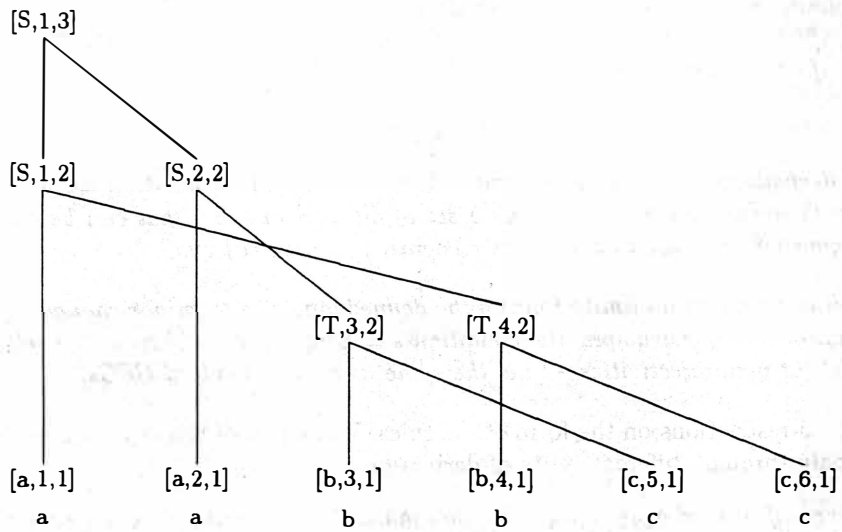



**Fig. 1.** A tree generated by the grammar $G_1$

## 2.2 Algorithms of recognition and parsing

From our point of view it is interesting to find out how it would be possible to compute for given words $a_1 a_2 \dots a_n$ one or all *Trees* constructed according to a given NCFDG $G$.

The algorithms described in the following paragraphs are based on a similar process of construction of items as for example the CYK algorithm. Those items are represented by six-tuples

$$[symbol, position, coverage, ls, rs, rule]$$

containing enough information necessary for both the reconstruction of the parsing process and for the creation of a *Trees* representing the structures of the parsed sentence.

The decision which information will be taken into account by the algorithm is then influenced by the fact whether we are trying to do only the recognition or a full parsing of a particular sentence.

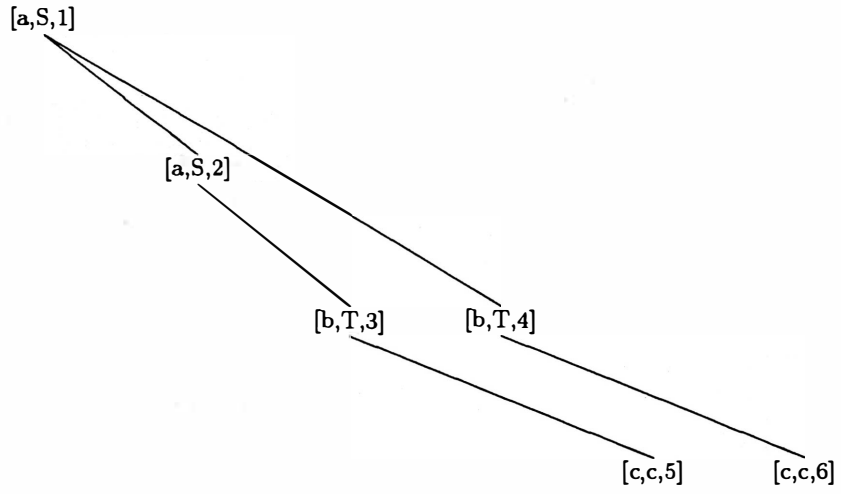



**Fig. 2.** Dependency tree corresponding to the tree from Fig 1.

**Algorithm 7 (parsing).** This algorithm works with a list $D$ of items

$$D_i = [symbol, position, coverage, ls, rs, rule]$$

where every item corresponds to some derivations of a subsequence of the word $a_1 a_2 \ldots a_n$ according to a grammar $G$ starting with the symbol *symbol*.

*symbol* represents the root symbol of the *Tree* corresponding to the derivation,

*position* is its horizontal index,

*coverage* represents the set of horizontal indices of its leaves (yield),

*ls, rs* are indices of items $D_{ls}$, $D_{rs}$ containing the left and right daughter of the root of the particular *Tree* and

*rule* means the serial number of the rule according to which the item was created.

The basic idea of the algorithm is to add new items to the list as long as possible.

The list $D$ is initialized so that for each word $a_i$ (the word from the input sentence) we create an item

$$[a_i, i, \{i\}, 0, 0, 0]$$

Let $|D|$ be the leghth of the list $D$, $D[i]$ denotes the i-th item of $D$.

The derivation is then performed by gradual comparison of all pairs $i, j$, $1 <= i, j <= |D|$ and by investigating whether it would be possible to apply some rule $A \rightarrow_X BC$ ($_X$ is either $L$ or $R$), which would derive the left-hand side from the right-hand side. If this is possible, a new item is created, which inherits the position either from $D[i]$ or from $D[j]$, according to whether $X$ was L or R. The new coverage is a union of both coverages.

Two items may create a third one only if their coverages are disjoint. The difference between this algorithm and CYK is in the way how the coverage is handled by both algorithms. In CYK, the coverage is a continuous interval from-to and since the new, derived item also has to have a continuous coverage, the rule is applied only in case that j's "from" follows i's "to".

Our approach, which allows to deal with non-projective constructions, replaces this constraint by the constraint on the disjointness of both coverages. The coverage of the result is the same in both cases — the union of original individual coverages.

Let us suppose that a[1..n] is the array containing the analyzed sentence.

Step 1. (initialization)

```
for i:=1 to n do
    D[i]:=[a[i], i, {i}, 0,0,0];
NumberOfItems:=n;
```

Step 2. (derivation)

```
i:=2;
while i <= NumberOfItems do
   for j:=1 to i-1 do
     if D[i].coverage * D[j].coverage = {} then
       for rule:=1 to NumberOfRules do
       begin
         Try( D[i], D[j], rule )
         Try( D[j], D[i], rule )
       end;
```

where the procedure Try( Item1, Item2, NoOfRule ) tries to apply a rule with the number NoOfRule to items Item1 and Item2.

**Algorithm 8 (recognition).** This algorithm is the same as Algorithm 7 with only one difference: adding of a derived item to the list D is limited only to those cases, in which D does not yet contain an item equal to the new item in the fields $[symbol, -, coverage, -, -, -, -]$.

### 2.3 One gap

The algorithm described above has, despite of its similarity to CYK, greater computational complexity because it is extended to non-projective constructions. We introduce a contraint which may be imposed on non-projectivity and therefore may improve the complexity of the algorithm.

We say that a relation $\Rightarrow$ is *derivation with one gap* if for every item *Item* derived by $\Rightarrow$ there are some

$$i, j, k, l \in 1 \ldots n, i \leq i \leq j \leq k \leq l \leq n$$

so that

$$Item.coverage = \{i \ldots j\} \cup \{k \ldots l\}$$

Unfortunately upper bound of the complexity of recognition by Algorithm 2 according to a general NCFDG is exponential. On the other hand, time and space complexity of the recognition with one gap is polynomial to the length of the sentence and to the size of the grammar, too.

In the sequel, the derivation will always mean the derivation with one gap.

## 3 The Treatment of Ambiguities of Word Forms

We have already mentioned one important property of natural languages — the ambiguity of word forms. It means that the same word form can belong to two or more different syntactic categories.

Our algorithm, in the same vein as CYK and other tree-based algorithms, allows to solve this problem in a strikingly simple way, changing only the initialization step of the above described algorithm from

Step 1. (initialization)

```
for i:=1 to n do
    D[i]:=[a[i], i, {i}, 0,0,0];
NumberOfItems:=n;
```

to

Step 1. (initialization)

```
NumberOfItems:=0;
for i:=1 to n do
   for j:=1 to NoOfAmbiguity[i] do
   begin
      Inc( NumberOfItems );
      D[NumberOfItems]:=[a[i,j], i,{i}, 0,0,0];
   end;
```

where a[i,j] is the j-th syntactic meaning of the word a[i].

After this first step the algorithm continues as described above, without changes, as if all input words were unambiguous.

## 4 Data Structures

The data structures used in our implemetation are similar to a substantially simplified form of feature structures. Our data structure is a set of attributes and their values — the value of an attribute may be only an atomic value or a list of data structures containing only attributes with atomic values. This means that the recursive use of data structures as values of attributes is limited to one recursion only.

This form of data is suitable as a intermediate step between the feature structure based formalism, in which the data are described in the main morphosyntactic lexicon of the system and the inner representation the of data.

## 5 Rules

The rules of the formal grammar in our system are the rules of NCFDG. If we look at the rules of our grammar from the strictly theoretical point of view, we may notice that, compared to grammar rules of a grammar of a typical programing language, there is one substantial difference between these two. The difference lies in the size of the alphabet. While the programming languages typically do not exceed the size of hunderds of symbols in the alphabet, formal description of natural languages must count with hundreds of thousands.

On the other hand lots of words of a particular natural language have very similar syntactic properties. Therefore it would be natural to use for the description of natural language grammar certain meta-rules, which will describe the syntactic properties and interactions of whole classes of words.

For example a meta-rule describing a modification of a noun by an adjective standing immediatelly on the left hand side of the noun is applicable for those word forms, where the first (left) one has a value of an attribute *syntactic class* — *syntcl* equal to *adjective* — *adj* and second (right) one has the value of the same attribute equal to *noun* — *n*.

This simplified meta-rule is obviously applicable to a whole set of pairs of word-forms and therefore it may substitute all rules for every corresponding pair.

In contrast to other formalisms, the form of the description of meta-rules does not only declare constraints on the applicability of a particular rule (symbol $A$ in case of $A \rightarrow_X BC$ rule), but also explicitly defines the order of applying the constraints together with forming the resulting data structure (symbol $A$) from $B$ or from $C$ (with respect to whether the symbol $_X$ is equal to $_L$ or to $_R$).

### 5.1 A formalism of meta-rules

For the description of meta-rules, we have developed a simple formalism and interpreter of this formalism. Both were designed in order to simplify the process of development of a large scale grammar of Czech for the purpose of grammar checking of Czech texts. The formalism and the

interpreter serve for the pivot implementation of the grammar checker. They can also serve as tools for testing and debugging a grammar in the process of its development.

The current version of the interpreter works with three kinds of objects:

- Input items: data structures A,B
- Temporary item: data structure P
- Output item: data structure X

Every meta-rule is described by means of a sequence of elementary instructions; some of the instructions are used in the following example:

Example:

```
;  Rule 20 - Noun in genitive
;            filling the valency frame of a preceding noun
;
   PROJECT
   IF A.SYNTCL = n THEN
      IF B.SYNTCL = n THEN
      IF B.CASE = gen THEN
                ELSE  FAIL ENDIF
              ELSE  FAIL ENDIF
      ELSE FAIL ENDIF
   ENDIF
   A.RIGHTGEN = yes
   B.CYCLE  ? yes       unfilled_frame
   P in A.FRAMESET
   P.PREP = 0
   B.CASE ? P.CASE case_disagr_in_the_frame
   X:=A
   \ P from X.FRAMESET
   IF |X.FRAMESET|=0 THEN  X.CYCLE:=yes  ELSE ENDIF
   X.rightgen := no
   OK
   END_P
```

This rule is applied to a combination of two nouns, where the second noun is in genitive. The set of constraints (starting with A.RIGHTGEN = yes and ending with IF .. ENDIF) contains two different kinds of constraints - soft constraints (operator ?) and hard constraints (operator =). In case that it is necessary to apply the constraint relaxation technique (part of an extended grammar), soft constraints are those constraints which may be relaxed. If all constraints are fulfilled, the rule causes a change of a value of the attribute RIGHTGEN in the resulting structure from "yes" to "no".

### 5.2 Head of a meta-rule

Any of the two parameters PROJECT and NEGATIVE may precede all commands and constraints in a particular meta-rule.

PROJECT means that the rule may be applied only in a projective way — it is in principle same as a rule of a standard CFG.

NEGATIVE means that the rule is taken into account only in an extended grammar (the explanation follows).

The handling of meta-rules is more or less procedural — they are performed from top to bottom with respect to the order of constraints and commands. A typical meta-rule of our formalism is introduced in the following example:

## 6 The Architecture of the Grammar Checker

Our parser is being developed for the purpose of a pivot implementation of a grammar checker for Czech. The acceptable result of the work of a grammar checker is not only the message whether the sentence of a particular language belongs to the set of well-formed sentences of that language. It is also supposed that it should find and mark the location of syntactic errors.

The location of some errors may however be determined only on the base of extralinguistic knowledge. For this reason we will talk about a grammar checker as about the program able to locate syntactic inconsistencies rather than errors. The error is usually considered as an attribute of a particular word, the syntactic inconsistency is a relation between two or more words.

Our grammar consists of two different sets of rules, a set of positive rules (without constraint relaxation) and an extended set, which contains the rules with relaxed constraints and also some special error-handling rules [6]. These rules are called negative rules. They describe typical (known) incorrect constructions and also some general rules, which are supposed to be able to handle even some unknown types of errors and therefore to make the whole system robust. The extended grammar is a proper superset of the positive grammar.

Those two sets of rules may be applied in the parser with the following three types of results:

- The sentence is recognized using positive grammar — it is correct
- The sentence was recognized using at least one negative rule — the sentence contains at least one syntactic incosistency
- The sentence was not recognized — the grammar is too weak; the system does not know anything about the sentence

In order to increase the reliability of messages about syntactic inconsistencies and also to increase the performance of the system we may apply additional constraints to the application of negative rules. For example, in our implementation we impose a constraint that *Tree* cannot contain two immediately following negative edges (this corresponds to the application of two negative rules immediately following each other).

The flow of the grammar checking is the following: The program first tries to recognize the input sentence with the positive grammar without non-projectivities. When the recognition fails, the NCFDG recognition with the positive grammar or CFG analysis with the extended grammar (projective or non-projective) may be tried.

We speak about the recognition, because the recognition is sufficient for marking the sentence as being correct or incorrect. On the other hand, if a particular sentence is not recognized with the positive grammar, it is necessary to provide full parsing with the extended grammar in order to obtain all possible syntactic inconsistencies.

Let us mention one very important remark. Even after the full parsing using the extended grammar it is not possible to provide the user with an error message. The message for the user should be created by a separate module following the parser, which will have an access to all results of parsing in case that the parser runs more than once.

At the moment our system offers four main variants of recognition or parsing:

1. recognition (parsing), positive, projective
2. recognition (parsing), positive, non-projective
3. parsing, extended, projective
4. parsing, extended, non-projective

## 7 Implementation

The implementation of the parser described in this section was created in 1994 and was presented to the public in February 1995 at the review meeting of the PECO 2824 JEP in Prague. It consists of app. 4000 lines of source code written in Borland Pascal.

### 7.1 Sample results

It seems that the derivation does not create too many new items during the processing. It means that in most derivations by rules $A \rightarrow_L BC$ or $A \rightarrow_R BC$, $A$ is described by the same data structure as item $B$ or $C$. For example, a noun modified by an adjective has usually the same syntactic data as the noun which is not modified (this holds of course only for the unambiguous structures; the morphological ambiguity of nouns is on the other hand very often solved if the noun is modified by an adjective). This results in the fact that the final number of all derivable items does not substantially exceed the number of items describing the original input sentence. The storage requirements of all items grow linearly with the length of the sentence.

For the space requirements, the number of derived items (the six-tuples from the description of the algorithms) is more important. In this section we present some examples of recognition and analysis of some sentences which may illustrate the performance of the intepreter with respect to the correctness, projectivity and length of the sentences.

Examples show number of initial items / derived items / filtered out items / Trees and memory (in bytes) / time (in seconds) requirements.

The parser was tested on IBM PC 486 DX/2 66MHz with the grammar of twenty six meta-rules.

*1) Které děvčata chtěla koupit ovoce?*
(Lit.: Which girls wanted [to] buy fruits?)

This sentence allows also two readings, depending whether the pronoun "které" (which) depends on "girls" or on "fruits". In the first case there is a syntactic inconsistency in the sentence (the pronoun and the subject (girls) do not agree in case), in the second case there is a long-distance dependency between the pronoun and the object (fruits). This sentence clearly illustrates the advantages of our multi-way approach: it is very difficult, if not impossible, to provide both results in a single-way mode. In the single-way mode the system usually prefers one of the possible solutions. This sentence and its two possible readings also provide an evidence for our claim that the parser has to be followed by an evaluating module, which will create the message for the user.

| | Items | Space | Time |
|---|---|---|---|
| positive projective recognition | 45/29/6/0 | 3.201 bytes | 0,33s |
| negative projective parsing | 45/467/270/1 | 9.080 bytes | 2,15s |
| positive non-projective parsing | 45/146/40/1 | 5.982 bytes | 3,18s |

*2) KDS nepředpokládá spolupráci se stranou pana Sládka a není pravdou, že předseda křesťanských demokratů pan Benda prosadil v telefonickém rozhovoru s Petrem Pithartem ing. Dejmala do funkce ministra životního prostředí.*

(Lit.: CDP [does] not suppose cooperation with party [of] mister Sládek and it isn't true, that chairman [of] Christian democrats mister Benda enforced in telephonic discussion with Petr Pithart ing.Dejmal to function [of] minister [of] environment.)

This is a real, correct and projective sentence from newspapers — it serves as an example that the greatest problem of the current version of the interpreter are complex sentences. If no constraints are imposed on the interaction of words from different clauses in one meta-rule, the number of derived items grows dramatically. The investigations concerning the type of constraints necessary are in progress.

| | Items | Space | Time |
|---|---|---|---|
| positive projective recognition | 218/1998/1248/1 | 34.816 bytes | 19,66s |
| positive projective parsing | 218/4207/2125/16 | 147.284 bytes | 53,06s |

## 8 Conclusion

The main goal of this implementation of the parser was to test two basic ideas:

whether it is possible to substitute depth-first search (backtracking)[3] with the width-first search, which may, thanks to the pruning, give better results with respect to time, but usually has much bigger memory requirements, and,

whether it is possible to solve the unambiguity of word forms by creating a larger number of unambiguous items from the original ambiguous ones.

The results of the testing of our parser document that both ideas are acceptable and that much bigger role in the overall effectivity of the parsing process is played by the size of the grammar.

In the future, we are going to concetrate on the modification of our approach in order to handle efficiently complicated sentences containing many clauses, including some changes in the formalism for the description of meta rules and also in the program itself.

## References


1. G. Görz : Einführung in die Künstliche Intelligenz, Addison - Wesley, 1993
2. M. A. Harrison : Introduction to Formal Language Theory, Addison - Wesley, 1978
3. J. Hric: Implementation of the Formalism for a Grammar Checker, in: Practical Application of Prolog, eds.: L. Sterling, Al Roth, 1994, Royal Society of Arts, London, UK, pp. 271-280
4. V. Kuboň, V. Petkevič, M. Plátek : Formalism for Shallow Error Checking, JRP PECO 2824, Final Research Report of the Task Adaptation and Transfer of Description Formalisms, Saarbruecken, 1993
5. V. Kuboň, M. Plátek : Robust Parsing and Grammar Checking of free Word Order Languages, in Natural Language Parsing: Methods and Formalism eds. K. Sikkel, A. Nijholt, TWLT 6, December 1993, pp. 157 - 161
6. V. Kuboň, M. Plátek : A Grammar Based Approach to Grammar Checking of Free Word Order Languages, In: Proceedings COLING 94, Vol.II, Kyoto, August, 1994, pp. 906 - 910
7. M. Plátek : The Architecture of a Grammar Checker, In: Proceedings SOFSEM '94, Milovy, 1994, pp. 85 - 90
8. N. Sikkel: Parsing Schemata, Proefschrift, Enschede,ISBN 90-9006688-8, 1993
9. T. Holan, V. Kuboň, M. Plátek: An implementation of a syntactic analysis of Czech, Technical Report No 113. of KKI MFF UK Prague
10. Z. Kirschner: CZECKER - a Maquette Grammar-Checker for Czech, The Prague Bulletin of Mathematical Linguistics 62, MFF UK Prague, 1994, pp. 5–30

# ANALYZING COORDINATE STRUCTURES INCLUDING PUNCTUATION IN ENGLISH

Sadao Kurohashi*
Section of Electronics and Communication, Kyoto University
Yoshida-honmachi, Sakyo, Kyoto, 606-01 Japan
kuro@kuee.kyoto-u.ac.jp

**Abstruct**

We present a method of identifying coordinate structure scopes and determining usages of commas in sentences at the same time. All possible interpretations concerning comma usages and coordinate structure scopes are ranked by taking advantage of parallelism between conjoined phrases/clauses/sentences and calculating their similarity scores. We evaluated this method through experiments on held-out test sentences and obtained promising results: both the success ratio of interpreting commas and that of detecting CS scopes were about 80%.

## 1 Introduction

It is commonly recognized that coordinate structures (CSs) present both linguistic and practical difficulties [Steedman, 1990, Agarwal and Boggess, 1992, Okumura and Muraki, 1994]. The practical difficulties are especially serious because not only do CSs themselves have complex scope ambiguity, but also sentences with CSs tend to be long and the combination of scope ambiguity and the intrinsic syntactic ambiguity in long sentences can easily cause combinatorial explosion.

We first noticed the importance of solving this problem in the analysis of Japanese sentences, and developed a unique, efficient method of resolving CS scope ambiguities [Kurohashi and Nagao, 1992]. The underlying assumption of this method is that conjoined phrases/clauses/sentences exhibit parallelism, that is, they have a certain similarity in the sequence of words and their grammatical structures as a whole. Based on this assumption, we devised an algorithm which determines conjuncts by finding the two most similar word-sequences on the left and the right of a conjunction (this method is called *similarity-based CS analysis*). By incorporating this method into a conventional syntactic parser, we developed a reasonably good parsing system which can work on real world texts [Kurohashi and Nagao, 1994].

Since we expected that the analysis based on this assumption would also work well for English CSs, we slightly modified our system to handle English sentences and tested it against a certain amount of text. We found that it works well on simple cases, like sentences containing only one conjunction, but it does not always work well on real sentences. In Japanese, it is morphologically clear what kind of expressions indicate the existence of CSs and whether a CS consists of noun phrases or verb phrases. Therefore, even if two or more CSs exist in a Japanese sentence, it is not so hard to order their scopes and to get a totally consistent structure. In

*This work was done at Institute for Research in Cognitive Science, University of Pennsylvania. We would like to thank Prof. Aravind K. Joshi, Prof. Mitchell P. Marcus, B Srinivas, and Dania Egedi for helpful comments and advice.

English, however, it is not morphologically clear and so the analysis of many CSs in an English sentence is much harder. In particular, commas cause serious problems since they have a variety of usages: some of them indicate CSs, some surround parenthetical expressions, and others just indicate constituent boundaries.

To solve this problem, we have developed the following method:

1. enumerate all possible interpretations concerning usages of commas and hierarchical structures between CSs (this interpretation is called *general structure* (GS)),
2. for each GS, detect the most plausible scopes of the CSs using the similarity-based CS analysis (here, each CS is given a score expressing the similarity between its conjuncts),
3. rank the GSs roughly according to the sum of the similarity scores for the CSs in each GS. This results in the final analysis.

Another possible solution is to parse a sentence and then check parallelism of parsed possible CSs. This method has advantages in that the structural information in the CSs is available and checking syntactically impossible scopes can be avoided. In practice, however, it is very difficult to employ this method because sentences containing CSs tend to be long and an unacceptably large number of parses are often produced for such long sentences.[1] Our method presented here also produces an unacceptably large number of GSs for some sentences, however, the combinatorial explosion of GSs is much slower than that of straight parsing, so most sentences containing CSs can be handled.

Although some of the recent work on English CSs has taken their parallelism into account [Okumura and Muraki, 1994], there is no work that we are aware of which resolves the inherent problems of handling CSs — the dual problems of handling many CSs in a sentence and of interpreting commas.

We start by introducing the similarity-based CS analysis in Section 2. Then we clarify the necessity of enumerating possible GSs, and present our algorithm in Section 3. Finally we discuss the experimental evaluation in Section 4.

## 2 Similarity-Based CS Analysis

First of all, we describe the core engine of our method, similarity-based CS analysis, which takes advantage of parallelism between conjoined phrases/clauses/sentences. This method tries to find the two most similar word-sequences on the left and the right of a conjunction, and then considers them conjuncts.

The following explains the definition of a similarity measure between two word sequences and how to calculate it (Figure 3 to 5 are concrete examples, and the adjusted parameters used in our experiment are shown in the Appendix).

- An input sentence should be in the following form:

  $$w_0/POS_0 \;\; k_1 \;\; w_1/POS_1 \;\; k_2 \;\; w_2/POS_2 \;\; \ldots \;\; k_l \;\; w_l/POS_l$$

  where $w_i$ is a word other than *and*, *or*, *but*, comma(,), $POS_i$ is one or more possible parts-of-speech (POS) of $w_i$, $k_i$ is a sequence of zero or more *and*, *or*, *but*, comma(,). Hereafter, the possible values of $k_i$, such as ",", "*and*", ", *and*", are called *key expressions*.

[1] Furthermore, part-of-speech tagging near conjunctions is not very reliable and CSs sometimes contain gaps, so it is intrinsically difficult to parse sentences containing CSs.

- A triangular matrix, $A = (a(i,j))$, is composed for an input sentence, where the i-th diagonal element is $k_i w_i$, and another element $a(i,j)$ $(i < j)$ is the similarity value between words $w_i$ and $w_j$. A similarity value between two words is calculated according to POS matching, exact word matching, and their closeness in a thesaurus tree (WordNet is currently being used [Beckwith *et al.*, 1991]).
- In detecting a CS scope concerning $k_n$, a *path* is defined as follows, which basically shows correspondences between words in candidate pre- and post-conjuncts:

  $$\text{path} ::= (a(p_1, q_1), a(p_2, q_2), \ldots, a(p_r, q_r))$$

  where $q_1 = n$, $p_i < p_{i+1}$, $q_i < q_{i+1}$, $p_r = n - 1$.
- A *path score* is defined by the following five criteria which indicate the similarity between two word sequences on the left side of the path and under the path.
  - sum of each element's points on the path,
  - penalty points when the path extends non-diagonally (which causes unbalanced lengths of conjuncts),
  - penalty points if the path contains symbols indicating a constituent boundary, such as a comma,
  - bonus points on expressions signaling the beginning or ending of a CS, such as *both*,
  - normalizing the total score of the above four criteria by the n-th power $(0 \leq n \leq 1)$ of the number of words covered by the path $(w_{p_1} \ldots w_{q_r})$.
- When the path with the maximum path score concerning the key expression, $k_n$, is detected, two word sequences on the left side of the path and under the path are considered to be the two most similar word sequences, that is, to be the conjuncts concerning $k_n$. Calculation of path scores and selection of the path with the maximum path score are done using a dynamic programming method (see [Kurohashi and Nagao, 1992] for the details).

## 3 Enhanced Similarity-Based CS Analysis

### 3.1 Necessity of Enumerating Possible General Structures

The similarity-based CS analysis just described can identify a CS scope for two conjuncts on the left and the right of a key expression. However, in real sentences there often exist CSs consisting of three or more conjuncts, such as "*A, B, and C*", and some sentences may contain two or more CSs (which can be hierarchical). In order to handle these cases, the original method for Japanese sentences consisted of the following steps:

1. for all key expressions in a sentence, detect their CS scopes,
2. by checking overlapping relations in all pairs of detected CS scopes,
   - combine CSs into one CS whenever they consecutively overlap each other (i.e., "*A, B, and C*" is first detected as "*A, B*" and "*B and C*"),
   - recognize a hierarchical relation between two CSs whenever one of them includes the other entirely.

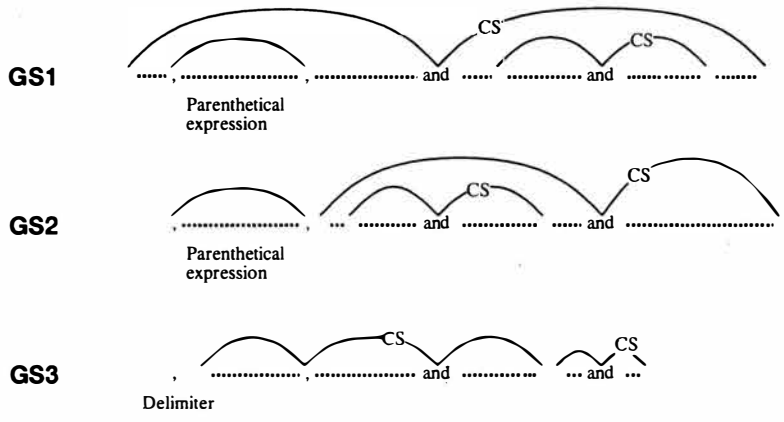



Figure 1: Examples of general structures.

In English, however, this method of processing complex CSs does not always work well. For example, a sentence like " ... , ... , ... *and* ... *and* ..." has a number of possible general structures (GSs), that is, a number of possible interpretations concerning usages of commas and hierarchical structures between the CSs. Some of these are shown in Figure 1. Suppose the first GS in Figure 1 is correct. When the similarity scores between two word sequences around the first *and* are calculated on the original sentence, it is possible that correct conjuncts do not have the maximum similarity score. Instead, an incorrect scope, like the second GS in Figure 1, may have a larger similarity score and thus be selected.

In order to give a reasonably large similarity score to the correct conjuncts, the following steps are necessary: 1) postulate that the first GS is correct, that is, the two commas surround a parenthetical expression and the CS concerning the first *and* contains the CS concerning the second *and*, 2) assume a reduced sentence in which the parenthetical expression and either conjunct of the latter CS are removed, and 3) perform the similarity-based CS analysis on the reduced sentence. Of course, since we do not know which GS is correct, this test must be performed on all possible GSs.

### 3.2 Enhanced Similarity-Based CS Analysis

Based on the considerations detailed in the previous section, we have developed an enhanced CS analysis method to process English sentences with complex CSs. The steps of this enhanced method are presented in detail below.

#### Step 1. Enumerating Possible GSs

The main problem in CS analysis is the ambiguity of commas. A comma may be used in any of the following ways.

*Strong connector* — followed by *and* or *or*, resulting in a CS with three or more conjuncts,

*Weak connector* — not followed by *and* or *or* but resulting a CS,

*Parenthesizor* — surrounding a parenthetical expression with another comma,

*Delimiter* — indicating a certain constituent boundary.

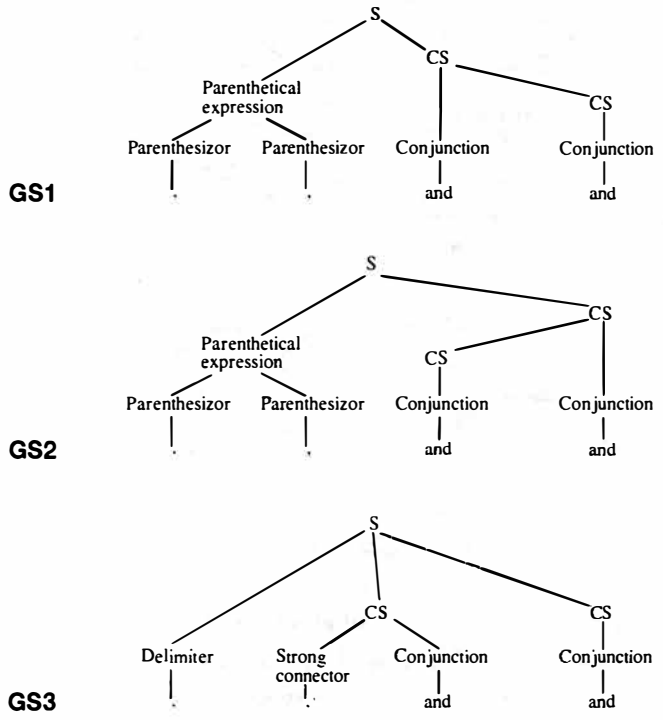



Figure 2: Examples of parse trees of key expressions.

Possible GSs are created by combining these interpretations of commas with patterns of hierarchical structures among CSs and parenthetical expressions. We constructed a set of phrase structure rules which regards key expressions[2] as terminal symbols and translates a sequence of key expressions to possible GSs. Some parse trees resulting from the sequence of key expressions, "... , ... . ... *and* ... *and* ...", and corresponding to the GSs in Figure 1 are shown in Figure 2. This sequence of key expressions has 135 possible parses (GSs) in total.

### Step 2. Detecting Plausible CS Scopes for each GS

Then, for each GS, CSs are analyzed, starting with the innermost CS in the GS and working outward. For each CS, its possible scope is limited to retain the GS, and the scope with the maximum similarity score within the limits is detected by the similarity-based CS analysis. For example, in the case of the first GS in Figure 1, a CS scope for the right *and* is first detected on the condition that its starting point is restricted not to go over the left *and*. Then a CS scope for the left *and* is detected within the range of its post-conjunct including the left entire CS.

Once the scope of a CS is detected, its conjuncts other than the first one are removed. In the case of a parenthetical expression, its content is removed after analyzing all CSs within it. By this process of reducing a sentence, the similarity-based CS analysis for an upper level CS in a GS which contains CSs and/or parenthetical expressions can be done more precisely and its proper scope is likely to be detected.

[2]While handling only commas as a punctuation symbol now, we intend to extend our system to handle other symbols, such as hyphen(-), colon(:), semicolon(;).

**Step 3. Ranking Possible GSs**

Scores are given to key expressions according to their usage. For conjunctions and connectors (commas) of CSs, the scores are based on the similarity score between their conjuncts, while for delimiters and parenthesizors, they are based on the expressions around them. As a final result, we rank all the possible GSs according to the sum of scores for all key expressions in each GS.

To give decreasing priority to the evaluation of a CS indicated by a conjunction, a strong connector, and a weak connector, the similarity scores of their CSs are weighted by $w_{conjunction}$, $w_{s\_connector}$, $w_{w\_connector}$ respectively, where $w_{conjunction} > w_{s\_connector} > w_{w\_connector}$.

When a key expression is a comma interpreted as a delimiter or a parenthesizor, a larger score is given if it appears with an expression being considered to be parenthetical or calling for a delimiter. If not, a smaller score is given. For example, when the commas in a sentence, "... , followed by ... , ...", are interpreted as a parenthesizor, they are given larger scores.

## 4 Experiments and Discussion

We report the experimental results to illustrate the effectiveness of our present method. From the beginning of our work on English CSs, we have used an article from Brown Corpus to test the original method, to develop the enhanced similarity-based CS analysis method, and to adjust the various parameters in the method (the adjusted parameters are shown in the Appendix).[3] Then we randomly chose six more articles from Brown Corpus, and analyzed them using the method with the adjusted parameters.

### 4.1 Experimental Evaluation

Table 1 shows the statistics of sentences with key expressions and the average number of their possible GSs. Out of 709 sentences of the six test articles, 424 sentences contain commas and/or conjunctions and need this kind of analysis. Among them, 14 sentences which contained six or more key expressions and would produce an unacceptably large number of GSs (underlined in Table 1) were excluded from the experimentation.[4]

An input to our method is a POS tagged sentence. The following four types of test sets were prepared:

Type 1 : manually tagged (revised) sentences found in Penn Treebank [Marcus *et al.*, 1993],

Type 2 : one-best tagged sentences by Brill's tagger [Brill, 1994],

Type 3 : n-best tagged sentences by Brill's tagger,

Type 4 : combination of the one-best and n-best tagged sentences (n-best tags were given only to words adjoining key expressions).

[3]The parameters used in this paper were adjusted manually through experimentation. As for automatic parameter tuning, we are currently working on a project, getting promising results. This issue will be published in the near future.

[4]In sentences containing six or more key expressions, some of the key expressions often compose very simple CSs, like "... houses, hotels, clubs, ships, and theaters ..." or "... red, green, or blue ... ". By constructing heuristic rules for such simple CSs and applying them to input sentences first, the number of possible GSs would be reduced so that our method can handle sentences with many key expressions. Implementation of such mechanism will be a target of our future work.

Table 1: Statistics of key expressions and GSs.

| # of *and*, *or*, *but* | # of commas | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 285 (–) | 59 (2) | 29 (8) | 6 (44) | 3 (284) | 2 (2004) | 1 (?) | 0 (–) |
| 1 | 137 (1) | 64 (5) | 25 (29) | 5 (182) | 1 (1529) | 2 (?) | 0 (–) | 1 (?) |
| 2 | 32 (3) | 18 (18) | 9 (119) | 3 (932) | 3 (?) | 0 (–) | 0 (–) | 0 (–) |
| 3 | 5 (12) | 6 (82) | 4 (574) | 1 (?) | 1 (?) | 1 (?) | 0 (–) | 1 (?) |
| 4 | 0 (–) | 2 (416) | 1 (?) | 0 (–) | 0 (–) | 0 (–) | 0 (–) | 0 (–) |
| 5 | 0 (–) | 0 (–) | 2 (?) | 0 (–) | 0 (–) | 0 (–) | 0 (–) | 0 (–) |

a (b)

a : # of sentences occurring in the test articles,
b : average # of GSs.

Table 2: Analysis results of interpreting commas and detecting CS scopes.

| | Comma interpretation | CS scope detection |
|---|---|---|
| Type 1 (with Penn Treebank tags) | 83.8% | 81.0% |
| Type 2 (with one-best tags) | 82.6% | 79.0% |
| Type 3 (with n-best tags) | 72.9% | 67.7% |
| Type 4 (with combined tags) | 83.2% | 77.6% |

Table 3: Analysis results of interpreting commas.

| Correct interpretation | Detected interpretation | | |
|---|---|---|---|
| | Strong connectors | Weak connectors | Parenthesizors or delimiters |
| Strong connectors | 35 | 11 | 8 |
| Weak connectors | 5 | 28 | 6 |
| Parenthesizors or delimiters | 6 | 23 | 229 |

The total success ratio was 83.2% ((35+28+229)/(35+11+8+5+28+6+6+23+229)).

Table 4: Analysis results of detecting CS scopes.

| Types of key expressions | , | , , | , , , | *and* *or* *but* | , *and* , *or* | , , *and* , , *or* | , , , *and* , , , *or* | Total |
|---|---|---|---|---|---|---|---|---|
| # of correct analysis | 17 | 2 | 2 | 286 | 26 | 3 | 1 | 337 (77.6%) |
| # of occurrence | 22 | 4 | 3 | 356 | 40 | 8 | 1 | 434 |

We analyzed these test sets by our method and checked the results as follows:

Interpretation of commas

We checked whether commas were interpreted correctly or not (the left-hand side of Table 2).

Detection of CS scopes

We also checked whether the detection of a CS scope was correct or not by automatically comparing the analysis results with the parsed sentences in the Penn Treebank[5] (the right-hand side of Table 2). Since the aim of our method is to get rough information on CS scopes (mentioned in the next section in detail), we regarded a detected scope as correct when the detected CS scope contains at least the head words of all conjuncts and, if any, at most one extra constituent of determiner or preposition.

The success ratios both for the comma interpretation and for the CS scope detection were around 80%, for the test sets of type 1, 2 and 4. We concluded that they were fairly good, comparing the results of previous work [Agarwal and Boggess, 1992, Okumura and Muraki, 1994] and considering that our method handles the comma interpretation as well as the CS scope detection. Table 3 and 4 show the details of our evaluation for the combined-tag test set (type 4).

### 4.2 Discussion

#### Goal of Our Method

Here, we would like to make the aim of our method clear. Our goal is to get a general information on sentences containing commas and/or CSs, including the interpretation of comma usages and the scopes of CSs. Although the CS scopes obtained by our method are only rough approximations, to get such rough approximations is the first and most critical task. Later stages can be used to determine their exact scopes.

For example, suppose there is an input sentence whose POS sequence is "$N_1\ N_2\ V_1\ N_3\ N_4$ *and* $N_5\ V_2\ N_6\ N_7$". The aim of our method is to determine whether the *and* conjoins two sentences or two noun phrases. If our method estimates that the two sentences are conjoined, the detected scope might not contain border nouns, such as $N_1$ and $N_7$. But they would be easily included in the CS scope at a subsequent syntactic analysis stage. On the other hand, when our method estimates the noun phrase coordination, it might not return the precise scope, either. The two possibilities, "$(N_3\ (N_4$ and $N_5))$" or "$((N_3\ N_4)$ and $N_5)$", would need to be distinguished at a semantic analysis stage.

#### POS Tagging and CS Analysis

Since POS tagging near conjunctions is not very reliable, n-best tagged sentences, rather than one-best tagged sentences, are considered to be a better input to a CS analysis system, simulating its real application. The current n-best tagger, however, assigns too many tags to the words. As a result, it causes a troublesome side-effect, giving inappropriate similarity scores to many pairs of words, and the success ratio of CS scope detection for n-best tagged sentences becomes very low as shown in Table 2.

[5]Since CSs by weak connectors were not explicitly indicated in the Penn Treebank, they were checked manually.

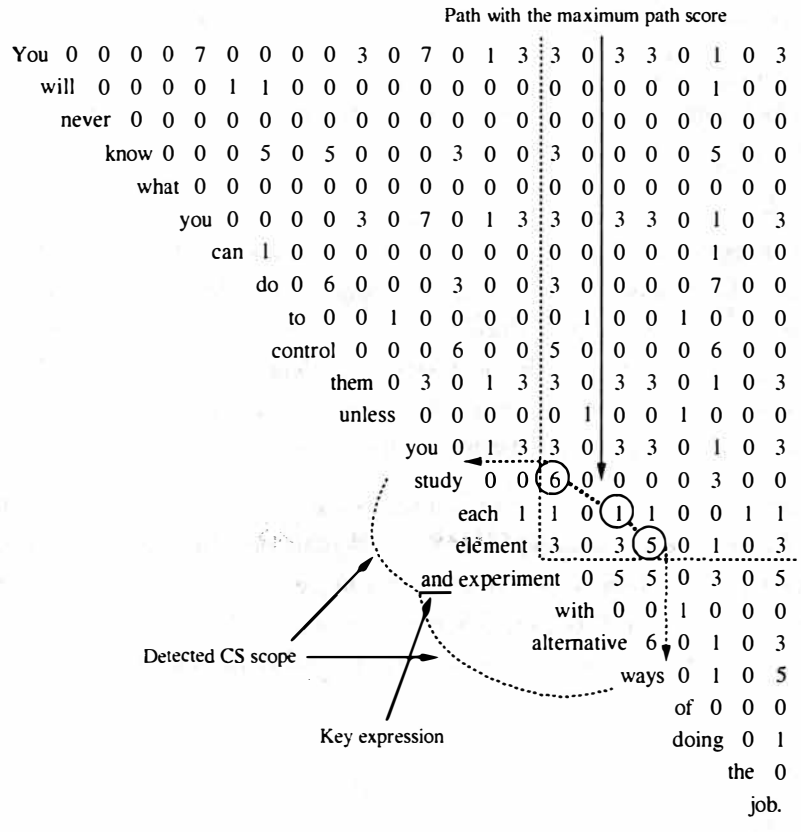



Figure 3: An example of detecting CS scopes and interpreting comma usages (1).

One compromise we have done is that we combined the results by the one-best tagger and that by the n-best tagger, as in our test set, type 4. For this test set, the success ratio was as good as the one-best tagged set, and some CSs which could not be detected correctly because of the errors of the one-best tagger, were detected correctly by using the combined tags (Figure 3 shows an example of such CSs).

This result is very promising since it shows that our method can handle two very difficult problems: POS ambiguity and CS scope ambiguity, simultaneously in a sense. (Note that getting the correct CS scope directly leads to the resolution of POS ambiguity around the key expression.)

### Examples of Correct Analysis

In the case of the example in Figure 3, the verb phrase coordination was analyzed correctly. The detected scope of the CS was regarded as correct since the two head verbs are included, although the ending words are not. In this example, when the one-best tagger was used, the incorrect tag, **NN** (Noun), was given to the head verb of the pre-conjunct ("experiment") and so the CS could not be analyzed correctly. On the other hand, the use of the n-best tags assignment to the word, **NN** (Noun), **VB** (Verb, base form), and **VBP** (Verb, non-3rd person, singular, present) led to the correct analysis, as shown in Figure 3.

In the case of the example sentence in Figure 4, the two commas were correctly interpreted as surrounding a parenthetical expression, and the verb phrase coordination was analyzed correctly.

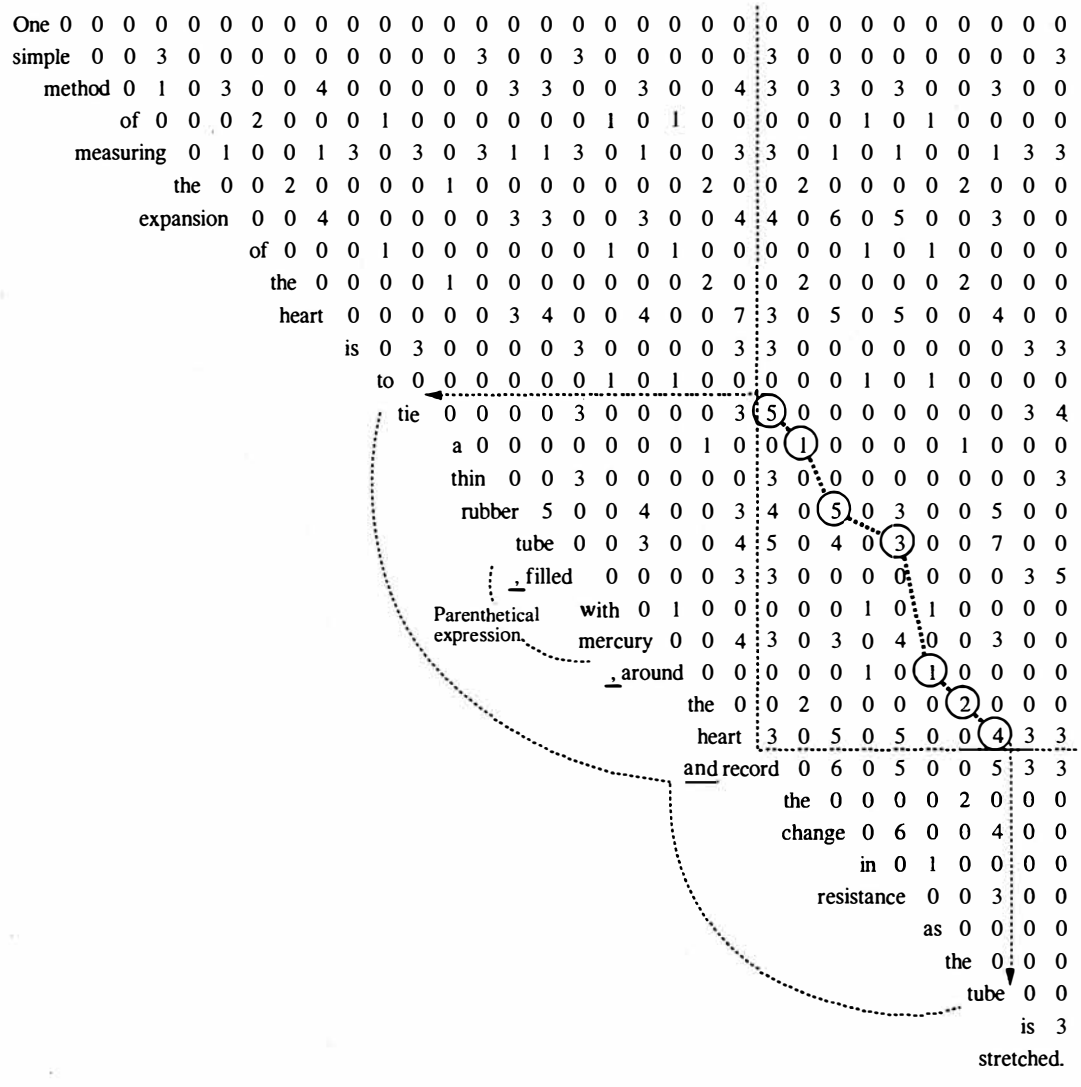



Figure 4: An example of detecting CS scopes and interpreting comma usages (2).

In the example in Figure 5, the sentential coordination containing a gap and indicated by a weak connector was analyzed correctly. The former comma is correctly interpreted as a delimiter; the latter as a weak connector.

### Types of Incorrect Analysis

Finally, we can classify the types of incorrect analyses produced by our method.

- Some types of CSs are extremely hard to handle. The following CS from a test sentence consists of a prepositional phrase and wh-phrase, and it cannot be detected on a parallel basis (angle brackets are inserted to indicate the correct CS scope):

  ... but anyone ⟨ with a wide acquaintance in both groups ⟩ and ⟨ who has sat through the many round tables ⟩ will recognize ...

  Such expressions are, however, very rare in actual texts.

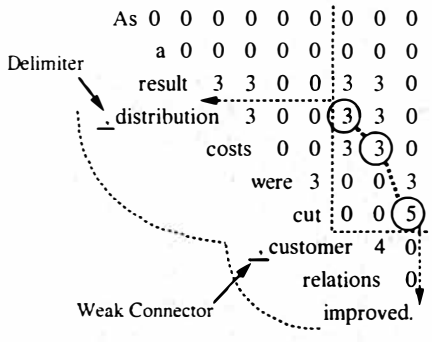



Figure 5: An example of detecting CS scopes and interpreting comma usages (3).

- Some idiomatic expressions were not analyzed correctly, for example, "*may and more likely may not*", "*... or nothing at all*". There seems to be no solution but to prepare a set of specific rules for such expressions. Since parsed corpora are now widely available, we believe that it will not be difficult to collect such expressions.
- As for the interpretation of commas, it was difficult to discriminate weak connectors from others. The comma in the following sentence was incorrectly interpreted as a weak connector (angle brackets indicate the incorrectly detected CS scope):

  > Outputs of the two systems are ⟨ measured by a pulse-timing circuit and a resistance bridge ⟩ , ⟨ followed by a simple analogue computer which feeds a multichannel recorder. ⟩

  We need to establish more precise detection methods for weak connectors.

## 5 Conclusion

We have proposed a method of detecting valuable information from real sentences which contain commas and/or CSs, including the interpretation of comma usages and the scopes of CSs. The core engine of our method is a way of identifying conjuncts by checking their parallelism. Using this engine, we have developed a method of enumerating possible general structures of a sentence and ranking them based on similarity scores for their CSs.

The information obtained by this method can be used to guide a syntactic parser to obtain parses for actual, long sentences. Integrating this method with an existing parsing system will be our next target.

## References

[Agarwal and Boggess, 1992] Rajeev Agarwal and Lois Boggess. A simple but useful approach to conjunct identification. In *30th Annual Meeting of the Association for Computational Linguistics*, pages 15–21, 1992.

[Beckwith *et al.*, 1991] Richard Beckwith, Christiane Fellbaum, Derek Gross, and George Miller. WordNet: A lexical database organized on psycholinguistic principles. In Uri Zernik, editors, *Lexical Acquisition: Exploiting On-Line Resources to Build a Lexicon*, pages 211–232, Erlbaum, 1991.

[Brill, 1994] Eric Brill. Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, 1994.

[Kurohashi and Nagao, 1992] Sadao Kurohashi and Makoto Nagao. Dynamic programming method for analyzing conjunctive structures in Japanese. In *Proceedings of 14th COLING*, Vol.1, pages 170–176, Nantes, 1992.

[Kurohashi and Nagao, 1994] Sadao Kurohashi and Makoto Nagao. A syntactic analysis method of long Japanese sentences based on the detection of conjunctive structures. *Computational Linguistics*, 20(4):507–534, 1994.

[Marcus *et al.*, 1993] Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[Okumura and Muraki, 1994] Akitoshi Okumura and Kazunori Muraki. Symmetric pattern matching analysis for English coordinate structures. In *Proceedings of the 4th Conference on Applied Natural Language Processing*, pages 41–46, Stuttgart, 1994.

[Steedman, 1990] Mark J. Steedman. Gapping as constituent coordination. *Linguistics and Philosophy*, 13:207–263, 1990.

## Appendix : Parameters

| **Similarity value between two words** | |
|---|---|
| POS matching of content words | 3 |
| POS matching of functional words | 1 |
| Exact word matching of content words | 4 |
| Exact word matching of functional words | 1 |
| Semantic similarity by WordNet | (8 − two words' distance in WordNet)/2 |
| **Path Score** | |
| penalty on non-diagonal path extension | |
| causing to skip a content word | −2 |
| causing to skip a functional word | −1 |
| Penalty on containing a comma being a delimiter | −4 |
| Bonus point on CS starting expressions | |
| *between*, *both* (← *and*) | 3 |
| *either*, *neither whether*(← *or*) | 3 |
| *not only* (← *but also*) | 10 |
| The power of normalization | 2/3 |
| **GS ranking** | |
| Weights for key expressions of CSs | |
| $w_{conjunction}$ | 1.5 |
| $w_{s\_connector}$ | 1.0 |
| $w_{w\_connector}$ | 0.5 |
| Score for a parenthesizor | 1.8 or 0.6 (depending on neighbor expressions) |
| Score for a delimiter | 1.79 or 0.59 (depending on neighbor expressions) |

# ON PARSING CONTROL FOR EFFICIENT TEXT ANALYSIS

Fabio Ciravegna and Alberto Lavelli
Istituto per la Ricerca Scientifica e Tecnologica
I-38050 Loc. Pantè di Povo, Trento, Italy
e-mail: {cirave|lavelli}@irst.itc.it

## 1. Introduction

Experience has shown that traditional models of language analysis as used in interfaces are not suitable to analyze texts, as in many cases they tend to visit the entire search space, pursuing every possible solution; moreover when a failure occurs they tend to hypothesize some exotic linguistic phenomena instead of a lack in their own coverage. As a consequence when confronted with complex inputs such as texts, a standard analyzer has to face a huge search space, with consequent excessive waste of computational resources, as well as reduced coverage. In particular coverage has shown to be the crucial aspect: on one hand it is well-known that parsers can not be expected to find full parses for every sentence; on the other hand coverage can increase the combinatorics of parsing and the likelihood of incorrect results. A linguistic analyzer has therefore to be controlled to pursue efficiency and robustness, especially when extensive linguistic resources are provided to reach broad coverage. Efficiency means visiting the search space looking for the most probable solutions, delaying or pruning other possibilities. Robustness means being able to gracefully cope with gaps in the system's knowledge.

In this paper we propose a control strategy for reducing the inefficiency of a broad coverage parser. It uses scores coming from extra-linguistic criteria, e.g text segmentation information, domain-specific heuristics, and the frequency of linguistic phenomena. The analyzer is also able to produce partial results when it is not possible to derive a global analysis, or its search becomes too expensive.

## 2. Parsing Control Strategies

The main component of the system's linguistic analyzer is an *agenda-based bottom-up bidirectional chart parser* [4], coupled with an agenda management mechanism which sorts tasks and rules to be applied, so to focus the analysis on the most promising solutions.

The keystone of the adopted control mechanism is a preliminary preprocessing phase based on shallow linguistic techniques. This phase is divided in two steps: text segmentation and text classification. Sentence segmentation is a typical technique used in text analysis [3]: in our approach segments represent basic constituents (such as simple nominal, verbal, prepositional phrases, etc.) and are detected via pattern matching. During text classification the current text is assigned to classes through statistical hierarchical pattern matching; the result of text classification is a set of templates to be filled in by the linguistic processor. During linguistic analysis the agenda management uses the preprocessor results to control the parser: the segmentation results are used to split the sentence analysis in two steps (i.e. segment parsing and segment combination), whereas information about the templates is used, among others, as heuristics to sort the tasks in the agenda [1].

During segment parsing the segments produced by the preprocessor are analyzed, producing basic constituents such as simple NPs, PPs, and so on. This means that the combination of edges crossing the boundaries of segments is prevented (i.e. delayed until segment combination) by an appropriate setting of the control strategy. At this level no other control mechanism is applied, as the number of edges generated by such basic constituents is quite small.

During segment combination the syntactic analysis is completed considering all the tasks in the agenda (included those that cross the boundaries of segments). This is the very moment when the control mechanism is necessary. The analyzer is controlled through the sorting of the tasks in the agenda; four different criteria are employed to contribute the scores used in task ranking:

- *Segmentation*: the tasks generated by edges spanning a whole segment are given an extra score. This criterion introduces some kind of top-down control on the parser actions. It is

useful for example to avoid some phenomena similar to garden paths that are easily followed by an uncontrolled parser.

- *Rule Score:* more frequent rules are applied first; this is particularly important in a free word order language like Italian, where the number of rules seldom applied tends to be very high.
- *Template Filling:* solutions providing information for target template(s) are given an extra score, favoring the most interesting solutions from an applicative point of view.
- *Width:* tasks combining edges spanning wider parts of the input are preferred.

Each criterion produces a numerical score; these four scores are weighted and summed up. The effectiveness of the control strategy relies therefore on the weights given to the different scores. We are currently experimenting different criteria of composition [2]. Some preliminary results seem to indicate that segmentation is to be given the maximum weight, followed by template filling, rule score and width. This strategy seems to be appealing from an intuitive point of view too, as the main role is played by the top-down control for avoiding garden paths, followed by the goal-driven strategy (i.e. template filling); the rule score weight is still to be determined precisely.

At the end of parsing the (possible) different (complete or partial) solutions produced by the linguistic analyzer can be evaluated according to the same four criteria (i.e. segmentation, template filling, rule score and width). In this case the most important one will be of course the template filling criterion.

The strategy mentioned above allows the parser to focus on the most promising solutions. Nevertheless focusing is not enough for efficiency, unless coupled with a process of reduction of the search space; in our case that reduction means pruning some tasks from the agenda. Unfortunately, pruning is dangerous, as it can prevent the parser from finding the correct solution. In our approach the agenda is pruned of some low score tasks when at least a "satisfying" solution has been found. Some experiments have been carried out to determine a criterion for considering a solution as "satisfying"; currently the following minimal criterion is used: when an edge spanning all the sentence has been found that has maximum segmentation score and an *acceptable* template score, the tasks with lower segmentation scores are pruned. This strategy also allows to overcome possible segmentation errors [2].

## 3. Parser at Work

The modules described so far have been integrated in a system for text understanding currently under development at IRST [1]. The system has been implemented in Common Lisp. As for the linguistic modules of the system, both syntactic and semantic information are encoded using a formalism based on Typed Feature Structures.

We have been experimenting on a corpus composed of short news (average 70 words) taken from the Italian financial newspaper "IL SOLE 24 Ore". Up to now, we have carried on only some preliminary experiments that seem to show a certain reduction in the amount of edges built by the parser, i.e. between 10% and 40%. The computational cost of the application of the control strategies is negligible.

In the near future we plan to do more extensive experiments to find the most effective way of combining the four criteria and to consider additional criteria. An evolution of the agenda pruning strategy using also rule scores is under study.

## References


[1] Fabio Ciravegna and Nicola Cancedda. Integrating shallow and linguistic techniques for information extraction from text. In *Proceedings of the Fourth Congress of the Italian Association for Artificial Intelligence*, Firenze, Italy, October 1995. To be published in Lecture Notes in Artificial Intelligence, Springer-Verlag.

[2] Fabio Ciravegna and Alberto Lavelli. Controlling bidirectional parsing for efficient text analysis. Technical Report 9508-02, IRST, July 1995.

[3] Paul S. Jacobs. To parse or not to parse: Relation-driven text skimming. In *Proceedings of the Thirteenth International Conference on Computational Linguistics*, pages 194–198, Helsinki, Finland, 1990.

[4] G. Satta and O. Stock. Formal properties and implementation of bidirectional charts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1480–1485, Detroit, MI, 1989.

# A PRACTICAL DEPENDENCY PARSER

Vincenzo Lombardo
Leonardo Lesmo

Dipartimento di Informatica - Università di Torino - c.so Svizzera, 185 - 10149 Torino - Italy
Centro di Scienze Cognitive - Università di Torino - via Lagrange, 3 - 10123 Torino - Italy
e-mail: {vincenzo, lesmo}@di.unito.it

The working assumption is that cognitive modeling of NLP and engineering solutions to free text parsing can converge to optimal parsing. The claim of the paper is that the methodology to achieve such a result is to develop a concrete environment with a flexible parser, that allows the testing of various psycholinguistic strategies on real texts. In this paper we outline a flexible parser based on a dependency grammar.

Cognitive modeling of human parsing and engineering solutions to text parsing are usually far apart. The goal of this research is to test whether the two fields can converge to optimal parsing. Efficiency in NLP is mostly obtained by separating the two phases of parsing and interpretation: an active chart parser can produce an annotated compact representation of the exponential number of syntactic trees in cubic time; the interpreter reads out the syntactic trees and builds a semantic representation for each valid one. The annotation corresponds to associate a disjunction of tuples with an edge. The interpretation process is theoretically exponential, and various authors have tried to reduce the computation time by means of devices that extend the disjoint representation to feature constraints application (Maxwell, Kaplan 1993) and to the semantic interpretation (Rim et al. 1990), or annotate the nodes of a shared-packed forest with the calls to the interpretation routines, that are "lazily" executed only when needed (Harper 1994).

Interleaving syntax and semantics has been proposed mostly in cognitive approaches (Schubert 1984) (Crain, Steedman 1985) (Hirst 1987). Since it is unreasonable to think that the human being carries on the exponential number of alternative paths that arise during the parsing process, many authors have proposed that many ambiguities are solved with the contribution of other sources, like semantics and context. Incremental interpretation, as this approach is usually referred to, involves an interleaving of the modules that can be implemented at various degrees. Interleaving requires the distinction of the several paths during the parsing process: only a limited number of paths is allowed for continuation (active paths), while many others are abandoned (inactive paths). The whole process is theoretically exponential, and we cannot isolate a part that can be executed in a polynomial time. The approach followed in this paper is to give up the polynomiality of parsing, by providing a method to factorize the paths in an efficient way and to easily switch between active and inactive paths when a recovery phase is required. We have implemented a flexible parser for testing various psycholinguistic hypotheses on the human parsing mechanism and we have equipped the parser with a recovery mechanism that allows an intelligent backtracking (Lombardo 1995). Currently we are testing the practical validity of a number of heuristics on an Italian corpus.

The syntactic representation is a *dependency graph*, that compacts all the dependency trees associated with a sentence. The dependency graph in the upper part fig. 1 is associated with the sentence "I saw a tall old man in the park with a telescope".

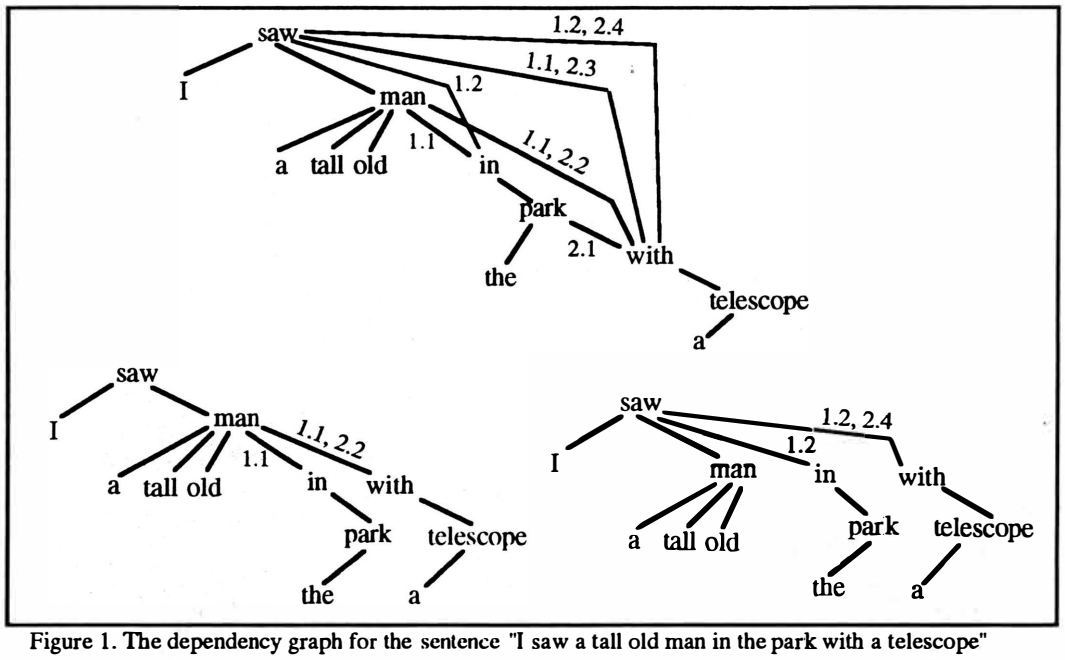



Figure 1. The dependency graph for the sentence "I saw a tall old man in the park with a telescope"

Various paths are identified by labelling some parts of the dependency graph with *indices* during the parsing process. An index is a pair of natural numbers h.k. Two indices, $h_1.k_1$ and $h_2.k_2$, are *compatible* iff either $h_1 \neq h_2$ or $h_1 = h_2$ & $k_1 = k_2$. A *path* is a set of indices compatible with each other. For each dependency tree T of a sentence there exists one and only one path P in G such that the tree consisting of all the vertices and edges reachable through P from a root vertex is equal to T. The bottom of fig. 1 shows the two dependency trees associated with the paths {1.1, 2.2}, {1.2, 2.4}.

The parser combines top-down predictions with bottom-up filtering. In case of ambiguity the parser explores in parallel the several paths for an input fragment. The dependency graph built in this phase keeps the paths distinct by means of the indices introduced above. The parser associates an integer h with each point of ambiguity and an integer h.k with each solution for the ambiguity h. Then the parser chooses the best path according to some heuristic. This becomes the *active path*, that is used for continuation; the others are the *inactive paths*. The parser supports an incremental interpretation at a fine degree of interleaving because the syntactic structure is always connected and each operation of the parser modifies the structure. The path representation is the communication medium between the parser and the recovery mechanism when something goes wrong.

## References

Crain S., Steedman M., *On not Being Led Up the Garden Path: The Use of Context by the Psychological Syntax Processor*, in Dowty, Karttunen, Zwicky (eds.): **Natural Language Parsing**, Cambridge Univ. Press, 1985, pp. 320-358.

Harper M. P., *Storing Logical Form in a Shared-Packed Forest*, **Computational Linguistics 20**, 1994, pp. 649-660.

Hirst G., **Semantic Interpretation and the Resolution of Ambiguity**, Cambridge Univ. Press, 1987.

Lombardo V., *Parsing and Recovery*, Proc. 17th Annual Conference of the Cognitive Science Society, Pittsburgh, 1995. pp.648-653.

Maxwell J. T., Kaplan R. M.: The Interface between Phrasal and Functional Constraints, **Computational Linguistics 19**, 1993, pp. 571-590.

Rim H. C., Seo J., Simmons R. F., *Transforming syntactic graphs into semantic graphs*, Proc. of the 28th Annual Meeting of the ACL, Pittsburgh (PA), 1990, pp. 47-53.

Schubert L. K., *On Parsing Preferences*, Proc. COLING 84, Stanford, 1984, pp. 247-250.

# A Labelled Analytic Theorem Proving Environment for Categorial Grammar

Saturnino F. Luz-Filho[†]
Patrick Sturt[‡]
Centre for Cognitive Science
2 Buccleuch Place, Edinburgh EH8 9LW, Scotland, UK
{luz,sturt}@cogsci.ed.ac.uk

**Abstract**

We present a system for the investigation of computational properties of categorial grammar parsing based on a labelled analytic tableaux theorem prover. This proof method allows us to take a modular approach, in which the basic grammar can be kept constant, while a range of categorial calculi can be captured by assigning different properties to the labelling algebra. The theorem proving strategy is particularly well suited to the treatment of categorial grammar, because it allows us to distribute the computational cost between the algorithm which deals with the grammatical types and the algebraic checker which constrains the derivation.

## 1 Background

A current trend in logic is to attempt to incorporate semantic information into the domain of deduction, [11], [5]. An area for which this strategy is particularly useful is the problem of categorial grammar parsing. The categorial grammar research programme requires the use of a range of logical calculi for linguistic description. Some researchers have considered labelled deduction as a tool for implementing categorial parsers [17], [19], and this paper can be seen as a new contribution to this field.

In this paper we aim for a modular approach, in which the basic grammar is kept constant, while different calculi can be implemented and experimented with by constraining the derivations produced by the theorem prover. At present, our system covers the classical Lambek Calculus, L, as well as the non-associative Lambek calculus NL, [13], and variants such as Van Benthem's [6] LP, LPC, LPE and LPCE, and their non-associative counterparts. The system is based on labelled analytic deduction, particularly on the LKE method, developed by D'Agostino and Gabbay [7]. LKE is similar to a Smullyan-style tableau system, in which the derivations obey the sub-formula principle, but it improves on efficiency by restricting the number of branching rules to just one. Different categorial logics are handled by assigning different properties to the labelling algebra, while the basic syntactic apparatus remains the same. This allows the user to experiment with various linguistic properties without having in principle to modify the grammar itself.

The basic structure of the paper is as follows. In section 1.1, we introduce the family of categorial calculi, and discuss some of the linguistic arguments which have been put forward in the literature with regard to these calculi. In section 2, we introduce the logical apparatus on which the system is based, describe the algorithm and prove some of the properties mentioned in section 1.1 within this framework. We also show how different grammars can be characterised and present a worked example. In section 2.3 the system is compared with other strategies for dealing with multiple categorial logics, such as hybrid formalisms and unification-based Gentzen-style deduction. In this section, we also suggest some ways to improve the efficiency of the system, and strategies for dealing with the complexity of labelled unification.

### 1.1 A Family of Categorial Calculi and Their Linguistic Applications

Categorial Grammars can be formalised in terms of a hierarchy of well understood and mathematically transparent logics, which yield as theorems a range of combinatorial operations. However

[†]Supported by CNPq Brazilian Research Council Research Studentship No. 200210/93-9
[‡]Supported by ESRC Research Studentship No. R00429334338

the precise nature of the combinatorial power required for an adequate characterisation of natural language is still very much a matter of debate. For this reason, it is desirable to have a means of systematically testing the linguistic consequences of adopting various calculi. In this section we give an overview of the linguistic applications of some of the calculi in the hierarchy, with a view towards motivating the usefulness of a generic categorial theorem prover as a tool for linguistic study.

The combinatorial possibilities of expressions in general can be characterised in terms of *reduction laws*. In *R1-R6* below, we give some reduction laws discussed in [16], which have been found to be linguistically useful.[1].

| | | | | | |
|---|---|---|---|---|---|
| *R1:* | *Application* | *R2:* | *Composition* | *R3:* | *Associativity* |
| | X/Y, Y ⊢ X | | X/Y, Y/Z ⊢ X/Z | | (Z\X)/Y ⊢ Z\(X/Y) |
| | Y, Y\X ⊢ X | | Z\Y, Y\X ⊢ Z\X | | Z\(X/Y) ⊢ (Z\X)/Y |
| *R4:* | *Lifting* | *R5:* | *Division (main functor)* | *R6:* | *Division (sub. functor)* |
| | X ⊢ Y/(X\Y) | | X/Y ⊢ (X/Z)/(Y/Z) | | X/Y ⊢ (Z/X)\(Z/Y) |
| | X ⊢ (Y/X)\Y | | Y\X ⊢ (Z\Y)\(Z\X) | | Y\X ⊢ (Y\Z)/(X\Z) |

It is possible to define a hierarchy of logical calculi, each of which admits one or more of *R1-R6* as theorems; from the purely applicative calculus AB, of Ajdukiewicz and Bar-Hillel, [2], which supports only *R1*, to the full Lambek calculus L, which supports all the above laws. Calculi intermediate in power between AB and L have been explored (e.g. Dependency Categorial Grammar [20]), as well as stronger calculi which extend the power of L through the addition of structural rules.

Much of the interest in using categorial grammars for linguistic research derives from the possibilities they offer for characterizing a flexible notion of constituency. This has been found particularly useful in the development of theories of coordination, and incremental interpretation. For example, assuming standard lexical type assignments, the following right node raised sentence cannot be derived in AB, but does receive a derivation in a system which includes *R3*, with each conjunct assigned the type indicated.

(1) [John resents $_{S/NP}$] and [Peter envies $_{S/NP}$] Mary

A calculus which includes composition, *R2*, will allow a function to apply to an unsaturated argument, and it is this property which allows Ades and Steedman [1] to treat long distance dependencies, and motivates much of Steedman's later work on incremental interpretation.

Dowty [9] uses the combination of composition, *R2* and lifting, *R4*, to derive examples of non-constituent coordination such as *John gave mary a book and Susan a record.*

We can increase the power of L by adding the structural transformations *Permutation*, *Contraction* and *Expansion*, to derive the calculi LP, LPC, LPE and LPCE. The structural transformation *Permutation*, which removes the restrictions on the linear order of types, allows us to go beyond the purely concatenative derivations of L. This allows us to deal with sentences exhibiting non-standard constituent order. For example, Moortgat suggests using permutation for dealing with *heavy NP-shift* in examples similar to the following [16]:

(2) John gave [to his nephew $_{PP}$] [all the old comic books which he'd collected in his troubled adolescence $_{NP}$].

In (2), the bracketed constituents can be "rearranged" via permutation so that a derivation is possible that employs the standard type ((NP\ S)/PP)/NP for the ditransitive verb *gave.*

In L, while it is possible to specify a type missing an argument on its left or right periphery, it is not possible to specify a type missing an argument "somewhere in the middle", making it impossible to deal with non-peripheral extraction. However, as Morrill et al show, permutation provides the additional power necessary to account for this phenomenon [12].

[1] We adopt the Lambek notation, in which X/Y is a function which "takes" a Y to its right to yield an X, and Y\X is a function which "takes" a Y to its left to yield an X.

In addition to permutation, there are also linguistic examples which motivate contraction (e.g. gapping, [16]) and expansion (e.g. right dislocation, [16]). However it is universally recognized that a system employing the unrestricted use of structural transformations would be far too powerful for any useful linguistic application, since it would allow arbitrary word order variation, copying and deletion. For this reason, a goal of current research is to build a system in which the resource freedom of the more powerful calculi can be exploited when required, while the basic resource sensitivity of L is retained in the general case. One such approach is to employ structural modalities [12], which are operators that explicitly mark those types which are permitted to be manipulated by specific structural transformations.[2]

## 2 A framework for Categorial Deduction

In this section we describe the theorem proving framework for categorial deduction. We start by setting up basic ideas of categorial logic, giving formal definitions of the core logical language. Then we move on to the theorem proving strategy, introducing the LKE approach [7] and the algebraic apparatus used to characterise different calculi.

### 2.1 The core syntax

We assume that there is a finite set of atomic grammatical categories which will be represented by special symbols: NP for noun phrases, S for sentences, etc. So, the set of well-formed categories can be defined as below.

Definition 1 *The set of well-formed categories, $C$ is the smallest set which contains every basic category and which is closed under the following rule:*

*(i) If $X \in C$ and $Y \in C$ , then $X/Y$, $X\backslash Y$ and $X \bullet Y \in C$*

Our purpose in this section is to define a prcedure which will enable us to verify, given an entailment relation $\vdash$ , whether or not such a relation holds for the logic being considered.

Many proof procedures for classical logic have been proposed: natural deduction, Gentzen's sequents, analytic (Smullyan style) tableaux, etc. Among these, methods which conform to the sub-formula principle are particularly interesting, as far as automation is concerned. See [10] for a survey. Most of these methods, along with proof methods developed for resource logics, such as Girard's proof nets (a variant of Bibel's connection method), can be used for categorial logic. Leslie [14] presents and compares some categorial versions of these procedures for the standard Lambek calculus L, taking into account complexity and proof presentation issues. Although tableau systems are not discussed in [14], a close relative, the cut-free sequent calculus is presented as being the one which represents the best compromise between implementability and display of the proof.

Smullyan style tableau systems, however, have been shown to be inherently inefficient [8]. They cannot even simulate truth-tables in polynomial time. The main reason for this is the fact that many of the Smullyan tableau expansion rules cause the proof tree to branch, thus increasing the complexity of the search. Moreover, keeping track of the structure of the derivations represents an extra source of complexity, which in most categorial parsers [17, 19] is reflected in expensive unification algorithms employed for dealing with substructural implication. In order to cope with efficiency and generality, we have chosen the LKE system [7] as the proof theoretic basis of our approach[3]. LKE is an analytic (its derivations exhibit the sub-formula property) method of proof by refutation[4] which has only one branching rule. In addition, its formulae are labelled according to a labelling algebra which will determine the closure conditions for the proof trees[5]. In what follows, we shall concentrate on explaining our version of the system, the heuristics that

[2]Systems which allow the selective use of structural transformations may be implemented in the general framework presented here, although we do not address this issue.

[3]For standard propositional logic, it has been shown that LKE can simulate standard tableau in polynomial time, but the converse is not true.

[4]A formula is proved by building a counter-model for its negation.

[5]See section 2.3 for a discussion of how LKE, unlike proof nets or standard tableaux, enables us to reduce the computational cost of label unification.

we have found useful for dealing with particularities of the calculi covered, and the relevant results for these calculi. The usual completeness and soundness results (with respect to the algebraic semantic provided) are already given in [7], so we will not discuss them here.

We have mentioned that the condition for a branch to be considered closed in a standard tableau is that both a formula and its negation occur on it. The calculus defined above presents no negation, though. So, we have to appeal to some extrinsic mechanism to express contradiction. In Smullyan's original formulation, the formulae occurring in a derivation were all preceded by *signs*: T or F. For instance, assume that we want to prove A $\Rightarrow$ A in classical logic. We start by saying that the formula is false, prefixing it by F, and try to find a refutation for F A $\Rightarrow$ A. For this to be the case both T A (the antecedent) and F A (the consequent) have to be the case, yielding a contradiction. In classical logic we can interpret T and F as assertion and denial respectively, and so we can incorporate F into the language as negation, obtaining uniform notation by eliminating the need for signed formulae. In our approach, since negation is not defined in the language, we shall make use of signed formulae as proof theoretic devices. T and F will be used to indicate whether or not a certain string available for combination to produce a new one.

### 2.2 The generalised parsing strategy

If we had restricted the system to dealing with signed formulae, we would have a proof procedure for an implicational fragment of standard propositional logic enriched with backwards implication and conjunction. However, we have seen that the Lambek calculus does not exhibit any of the structural properties of standard logic, and that different calculi may be obtained by varying structural transformations. Therefore, we need a mechanism for keeping track of the structure of our proofs. This mechanism is provided by labelling each formula in the derivation with *information tokens*[6].

Labels will act not only as mechanisms for encoding the structure of the proof, from a proof-theoretic perspective, but will also serve as means to propagate semantic information through the derivation. A label can be seen as an information token supporting the information conveyed by the signalled formula it labels. Tokens may convey different degrees of informativeness, so we shall assume that they are ordered by an anti-symmetric, reflexive and transitive relation, $\sqsubseteq$ , so that an expression like x $\sqsubseteq$ y asserts that y is at least as informative as x (i.e. verifies at least as many sentences as x). We also assume that this semantic relation, "verifies", is closed under deductibility.

It is natural to suppose that, as well as categories, information tokens can be composed. We have seen that a type S/NP can combine with a type NP to produce an S. If we assume that there are tokens x and y verifying respectively S/NP and NP, how would we represent the token that verifies S? Firstly, we define a token composition operation $\circ$ . Then, we assume that, a priori, the order in which the categories appear in the string matters. So, a minimal information token verifying S would be x $\circ$ y. As we shall see below, the constraints we impose on $\circ$ will ultimately determine which inferences will be valid. For instance, if we assume that the order in which the types occur is not relevant, then we may allow permutation on the operands, so that x $\circ$ y $\sqsubseteq$ y $\circ$ x; if we assume that contraction is a structural property of the calculus then the string [S/NP, NP, NP] will also yield an S, since y $\circ$ y $\sqsubseteq$ y, etc. Let's formalise these notions by defining an algebraic structure, called *Information frame.*

Definition 2 *An Information Frame is a structures* $\mathcal{L} = \langle \mathcal{P}, \circ, 1, \sqsubseteq \rangle$, *where (i)* $\mathcal{P}$ *is a non-empty set of information tokens; (ii)* $\mathcal{P}$ *is a complete lattice under* $\sqsubseteq$ *; (iii)* $\circ$ *is an order-preserving, binary operation on* $\mathcal{P}$ *which satisfies continuity, i.e., for* every *directed family* $\{z_i\}$, $\bigsqcup \{z_i \circ x\} = \bigsqcup \{z_i\} \circ x$ and $\bigsqcup \{x \circ z_i\} = x \circ \bigsqcup\{z_i\}$; *and (iv)* 1 *is an identity element in* $\mathcal{P}$ .

Combinations of types are accounted for in the labelling algebra by the composition operator. Now, we need to define an algebraic counterpart for syntactic composition, • , itself. When a formula like S/NP • NP is verified by a token x, this is because its components were available for combination, and consequently were verified by some other tokens. Now, suppose S/NP was verified by a token, say a. What would be the appropriate token for NP, such that S/NP

[6]See [11] for a proof theoretic motivated, LDS approach, and [5] for an approach based on a finer-grained, semantically motivated information structure.

combined with NP would be verified by x? It certainly would not be more informative than x. Moreover, if the expression S/NP • NP were to stand for the composition of the (informational) meanings of its components, then the label for NP would have to verify, when combined with a, at most as much information as x. In order to express this, we define the label for NP as being *the greatest y s.t. x is at least as informative as a combined with y.* This token will be represented by $x \swarrow a$. In general, $x \swarrow y \stackrel{def}{=} \bigsqcup \{z |\ y \circ z \sqsubseteq x\}$. An analogous operation, $\nearrow$, can be defined to cope with cases in which it is necessary to find the appropriate label for the first operand by reversing the order of the tokens in the definition above. Some properties of $\swarrow$[7]:

$$y \circ (x \swarrow y) \sqsubseteq x \quad (3) \qquad 1 \sqsubseteq x \swarrow x \quad (4)$$

$$(x \swarrow y) \circ z \sqsubseteq (x \circ z) \swarrow y \quad (5) \qquad (x \swarrow y) \swarrow z \sqsubseteq x \swarrow (y \circ z) \quad (6)$$

Having set the basic elements of our proof-theoretic apparatus, we are now able to define the components of a derivation as follows:

Definition 3 *Signed labelled formulae (SLF) are expressions of the form S Cat : L, where* $S \in \{T.F\}$. $Cat \in \mathcal{C}$ *and* $L \in \mathcal{L}$

A derivation, or proof will be a tree structure built according to certain syntactic rules. These rules will be called *expansion* rules, since their application will invariably expand the tree structure. There are three sorts of expansion rules: those which expand the tree by generating two formulae from a single one occurring previously in the derivation, those which expand the tree by combining two formulae into a third one which is then added to the tree, and the branching rule. The first kind of rule corresponds to what is called $\alpha$-rule in Smullyan tableaux; these rules will be called $\alpha$-rules here as well. The second and third kinds have no equivalents in standard tableau systems. We shall refer to the second kind as $\sigma$-rules, and to the branching rule as $\beta$-rule – after Smullyan's, even though his branching rules are different. *Figure 1* summarises the expansion rules to be employed by the system. A deduction bar says that if the formula(e) appearing above it occurs in the tree, then the formula(e) below it should be added to the tableau. The rules are easily interpreted according to the intuitions assigned above to signs, formulae and information tokens. A rule like $\alpha_{(i)}$, for example, says that if A\B is not available for combination and x verifies such information, then this is because there is an A available at some token a, but the combination of a and x (notice that the order is relevant) does not produce B. Given the expansion rules,

| **$\alpha$-rules** | (i) | (ii) | (iii) | **$\beta$-rule** | | |
|---|---|---|---|---|---|---|
| $(\alpha_1)$ | F A\B : x | F A/B : x | T A • B : x | $\dfrac{}{(\beta_1)\ TA : x \mid (\beta_2)\ FA : x}$ | | |
| $(\alpha_2)$ | T A : a* | T B : a* | T A : a* | | | |
| $(\alpha_3)$ | F B : a ∘ x | F A : x ∘ a | T B : x ↙ a | | | |
| **$\sigma$-rules** | (i) | (ii) | (iii) | (iv) | (v) | (vi) |
| $(\sigma_1)$ | T A\B : x | T A\B : x | T A/B : x | T A/B : x | F A • B : x | F A • B : x |
| $(\sigma_2)$ | T A : y | F B : y ∘ x | F A: x ∘ y | T B : y | T A : y | T B : y |
| $(\sigma_3)$ | TB : y ∘ x | F A : y | F B : y | T A : x ∘ y | F B : x ↙ y | F A : x ↗ y |

**: a new label* a *(not occurring previously in the derivation) must be introduced.*

**Figure 1:** Tableau expansion rules

the definition of the main data structure to be manipulated by the theorem proving (parsing) algorithm is straightforward: a derivation tree, $\mathcal{T}$, is simply a binary tree built from a set of given formulae by applying the rules. The next step is to define the conditions for a tree to be regarded as complete. Completion along with inconsistency are the notions upon which the algorithm's termination depends. It can be readily seen on *Figure 1* that for a finite set of formulae, the number of times $\alpha$ and $\sigma$ rules can be applied increasing the number of SLFs (nodes) in $\mathcal{T}$ is finite. Unbounded application of $\beta$, however, might expand the tree indefinitely. In order to assure

termination, applications of $\beta$ will be restricted to sub-formulae of formulae in $\mathcal{T}$. These notions are formalised in *Definition 4*.

Definition 4 *(Tree Completion) Given $\mathcal{T}$, a tree for a set of SLFs S, we say that a binary tree $\mathcal{T}^*$ is a tableau for S if $\mathcal{T}^*$ results from $\mathcal{T}$ through the application of an expansion rule. A tableau $\mathcal{T}^*$ is* linearly complete *if it satisfies the following conditions: (i) if $\alpha_1 \in \mathcal{T}$, then $\alpha_2$ and $\alpha_3 \in \mathcal{T}$; (ii) if $\sigma_1$ and $\sigma_2 \in \mathcal{T}$, then $\sigma_3 \in \mathcal{T}$. A tree $\mathcal{T}^*$ is* complete *iff for every $A \in \mathcal{T}^*$ and every sub-formula A' of A, both F A' : x and T A' : x have been added to $\mathcal{T}^*$ by an application of the $\beta$-rule.*

Now, the first step towards building a counter-model for the denial of the formula to be proved is the search for a tree containing *potential* contradictions. Whether or not a potentially inconsistent tree is a counter-model for the formula will depend ultimately upon the constraints on the labelling algebra. This form of inconsistency is defined below.

Definition 5 *(Branch and Tree Inconsistency) A branch is* inconsistent *iff for some type X both T X and F X, labelled by any information token, occur in the branch. A tree is inconsistent iff its branches are all inconsistent.*

Given the definitions above, we are ready to define an algorithm for expanding linearly the derivation tree. For efficiency reasons non-branching rules will be exhaustively applied before we move on to employing $\beta$-rules. *Definition 6* presents the basic procedure for generating linear expansion for a branch.[7] The complete LKE algorithm, *Definition 8*, which uses the procedure below, will be presented after we have discussed tableau closure from the information frame perspective.

Definition 6 *(Algorithm: Linear Completion) Given $\mathcal{T}$, a LKE-tableau structure, we define the procedure:*

Linear-Completion($\mathcal{T}$)
1 do $\mathcal{T} \Leftarrow \alpha$-completion($\mathcal{T}$)
2 formula $\Leftarrow$ head[$\mathcal{T}$]
3 while ( $\neg$completed($\mathcal{T}$) or consistent($\mathcal{T}$) )
4 do if $\sigma_1$-type(formula)
5 then do formula$_{aux}$ $\Leftarrow$ search($\mathcal{T}$,$\sigma_2$) ▷ formula$_{aux}$ is a set of $\sigma_2$-type slf's
6 if formula$_{aux} \neq \emptyset$ ▷ $\sigma_3$-set results from combining $\sigma_1$ to each $\sigma_2$
7 then do $\sigma_3$-set $\Leftarrow$ combine-labels($\sigma_1$,formula$_{aux}$)
8 $\sigma_3$-expansion $\Leftarrow$ $\alpha$-completion($\sigma_3$-set)
9 $\mathcal{T} \Leftarrow$ append($\mathcal{T}$,$\sigma_3$-expansion)
10 do formula $\Leftarrow$ next[$\mathcal{T}$]
11 return $\mathcal{T}$

We have seen above that the labels are means to propagate information about the formulae through the derivation tree. From a semantic viewpoint, the calculi addressed in this paper are obtained by varying the structure assigned to the set of formulae in the derivation[8]. Therefore, in order to verify whether a branch is closed for a calculus one has to verify whether the information frame satisfies the constraints which characterise the calculus. For instance, the standard Lambek calculus L does not allow any sort of structural manipulation of formulae apart from associativity; LP allows formulae to be permuted; LPE allows permutations and expansion (i.e. if B can be proved from the sequent $\Delta$, A, $\Gamma$, then B can be proved from $\Delta$, A, A, $\Gamma$); LPC allows permutation and contraction; etc. The definition below sets the algebraic counterparts of these properties.

Definition 7 *An information frame is: (i)* associative *if $x \circ (y \circ z) \sqsubseteq (x \circ y) \circ z$ and $(x \circ y) \circ z \sqsubseteq x \circ (y \circ z)$; (ii)* commutative *if $x \circ y \sqsubseteq y \circ x$; (iii)* contractive *if $x \circ x \sqsubseteq x$; (iv)* expansive *if $x \sqsubseteq x \circ x$; (v)* monotonic *if $x \sqsubseteq x \circ y$, for all $x, y, z \in \mathcal{P}$ .*

[7] The reader will notice that if we had allowed $\sigma_2$ formulae to search for $\sigma_1$ types for combination, in the same way that $\sigma_1$s search for $\sigma_2$s, then linear expansion would not terminate for some cases. Consider for example the infinite sequence of $\sigma$ applications: T A / B: x, T B / A : y, T A : z, T B : y ∘ z, T A : x ∘ (y ∘ z), T B : y ∘ (x ∘ (y ∘ z)),... A strategy to allow unrestricted $\sigma$-application without running into non-terminating procedures, as well as other practical and computational issues is discussed in [15].

[8] For instance, resource sensitive logics such as linear logic are frequently characterised in terms of multisets to keep track of the "use" of formulae throughout the derivation.

Now, we say that a branch is *closed* with respect to the labelling algebra if it contains SLFs of the form T X : x and F X : y, where x $\sqsubseteq$ y. Likewise, a tree is closed if it contains only closed branches. Checking for label closure will depend on the calculus being used, and consists basically of reducing information token expressions to a *normal form*, via properties (3)–(6), and then matching tokens and/or variables that might have been introduced by applications of the $\beta$-rule according to the properties or combination of properties (*Definition 7*) that characterise the calculus considered. It should be noticed that, in addition to the basic algorithm, heuristics might be employed to account for specific linguistic aspects. Some examples: (a) it could be assumed that all the bracketing for the strings is to the right thus favouring an incremental approach; (b) type reuse could be blocked at the level of the formulae, reducing the the computational cost of searches for label closure, since most of the calculi in the family covered by the system are resource sensitive; (c) priority could be given to juxtaposed strings for $\sigma$-rule application, etc. *Definition 8* gives the general procedure for tableau expansion, abstracted from the heuristics mentioned above.

Definition 8 *(Algorithm: LKE-completion) The complete tableau expansion for a LKE-tree $\mathcal{T}$ is given by the following procedure:*

```
expansion(T)
1  do closure-flag ⇐ no
2  while ¬( completed(T) or closed-T= yes)
3        do T⇐ linear-completion(T)
4           if ¬consistent(T) and label-closure(T)
5             then do closure-flag ⇐ yes
6             else do subformula ⇐ select-subformula(T)
7                     subformula_T ⇐ assign-label-T(subformula)
8                     subformula_F ⇐ assign-label-F(subformula)
9                     T_1 ⇐ append(T,{subformula_T})
10                    T_2 ⇐ append(T,{subformula_F})
11                    if ( expansion(T_1) = yes and expansion(T_2) = yes )
12                      then do closure-flag ⇐ yes
13 return closure-flag
```

As it is, the algorithm defined above constitutes a semi-decision procedure. This is due to the fact that even though the search space for signed formulae is finite, the search space for the labels is infinite. The labels introduced via $\beta$-rules are in fact universally quantified variables which must be instantiated during the label unification step. This represents no problem if we are dealing with theorems, i.e. trees which actually close. However, for completed trees with an open branch, the task might not terminate. In order to overcome this problem and bind the unification procedure we restrict label (variable) substitutions to the set of tokens occurring in the derivation — similarly to the way parameter instantiation is dealt with by liberalized quantification rules for first-order logic tableaux. In practice, the strategy adopted to reduce label complexity also employs the following refinements: (i) the tableau is linearly expanded keeping track of the choices made when $\sigma$-rules are applied (the options are kept in a stack); (ii) once this first step is finished, if the tableau is still open, then backtrack is performed until either the choices left over are exhausted or closure is achieved; (iii) only then is the $\beta$-rule applied. This explains the role played by the heuristics mentioned above.[9] We are now able to establish some results regarding the reduction laws mentioned in section 1.1.

Proposition 1 (Reduction Laws) *Let X, Y and Z be types, and $\mathcal{L}$ an information frame. The properties R1–R5 hold:*

Proof 1 *The proofs are obtained by straightforward application of* Definition 6 *and* Definition 8. *Below we illustrate the method by proving (R1) and (R2):*

*(R1) To prove right application we start by assuming that it is verified by the identity token* 1. *From this we have: 1- T X/Y • Y : m, 2- F X : 1 ∘ m = m. Then, we apply $\alpha_{(iii)}$ to 1*

[9] Furthermore, as we will show in section 2.3, if associativity is allowed at the syntactic level then it is possible to eliminate the branching rule for the class of calculi discussed here.

*obtaining 3- T X/Y : n and 4- T Y : m ↙ n. The next step is to combine 3 and 4 via $\sigma_{(iv)}$ getting 5- T X : n ∘ (m ↙ n). Now we have a potential closure caused by 5 and 2. If we apply property (3) to the label for 5 we find that n ∘ (m ↙ n) ⊑ m, which satisfies the closure condition thus closing the tableau.*

*(R2) Let's prove left composition. As we did above, we start with: 1- T Z\Y • Y\X : m and 2- F Z\X : 1 ∘ m. Applying $\alpha_{(iii)}$ to 1 we get: 3- T Z\Y : a and 4- T Y\X : m ↙ a. Now, we may apply $\alpha_{(i)}$ to 2 and get: 5- T Z : b and 6- F X : b ∘ m. Then, combining 3 and 5 via $\sigma_{(i)}$: 7- T Y : b ∘ a. And finally 4 and 7 through the same rule: 8- T X : (b ∘ a) ∘ (m ↙ a). The closure condition for 8 and 6 is achieved as follows:*

$$\begin{array}{lll} (b \circ a) \circ (m \swarrow a) & \sqsubseteq\ b \circ (a \circ (m \swarrow a)) & \textit{by associativity} \\ & \sqsubseteq\ b \circ m & \textit{by (3) and } \circ \textit{ being order-preserving} \end{array}$$

■

Even though L does not enjoy finite axiomatizability, the results above suggest that the calculus finds a natural characterization in LKE for associative information frames. In particular, the Division Rule (*R6*) can be regarded as L's characteristic theorem, since it is not derivable in weaker calculi such as AB, NL, and F. If we do not allow associative frames, we get NL. Stronger calculi such as LP, LPE, LPC and LPCE [18] can be obtained for the same general framework by assigning further properties to ∘ in the labelling algebra. Frames exhibiting combinations of monotonicity, expansivity, commutativity and contraction allow us to characterise these substructural calculi. Algebras that are both associative and commutative describe LP. Adding expansivity (weakening) to LP results in LPE. Associativity, commutativity and contraction describe LPC frames. LPCE is obtained by combining the properties of LPC and LPE algebras.

We end this section with a simple example requiring associativity: show, in L, that an NP (John), combined with a type (NP\S)/NP (likes) yields S/NP, i.e a type which combined with a NP will result in a sentence (Proof 2).

Proof 2 *Let's assume the following type–string correspondence: NP for* John, *(NP\S)/NP for* likes. *The expression we want to find a counter-model for is: 1- F NP • (NP\S)/NP ⊢$_L$ S/NP. Therefore, the following has to be proved: 2- T NP • (NP\S)/NP : m and 3- F S/NP : m. We proceed by breaking 2 and 3 down via $\alpha_{(iii)}$, obtaining: 4- T NP : a, 5- T (NP\S)/NP : m↙a, 6- T NP : b, and 7- F S : (m ∘ b).*

*Now we start applying σ-rules (annotated on the right-hand side of each line):*

*8- T NP\S : (m ↙ a) ∘ b 5,6 $\sigma_{(i)}$*
*9- T S : a ∘ ((m ↙ a) ∘ b) 4,9 $\sigma_{(i)}$*

*We have derived a potential inconsistency between 7 and 9. Turning our attention to the information tokens, we verify closure for* L *as follows:*

$$\begin{array}{lll} a \circ ((m \swarrow a) \circ b) & \sqsubseteq & (a \circ (m \swarrow a)) \circ b \quad \textit{by associativity} \\ & \sqsubseteq & m \circ a \quad \textit{by property (3)} \end{array}$$

■

### 2.3 Comparison with Existing Approaches

Early implementations of CG parsing relied on cut-free Gentzen sequents implemented via backward chaining mechanisms [16]. Apart from the fact that it lacks generality, since implementing more powerful calculi would involve modifying the code in order to accommodate new structural rules, this approach presents several sources of inefficiency. The main ones are: the generate-and-test strategy employed to cope with associativity, the non-determinism in the branching rules and in rule application itself. The impact of the latter form of non-determinism over efficiency can be reduced by testing branches for *count invariance* prior to their expansion and by performing sequent proof normalisation. However, non-determinism due to splitting in the proof structure still remains. As we move on to stronger logics and incorporate structural modalities such problems tend to get even harder.

An improved attempt to deal uniformly with multiple calculi is presented in [17]. In that paper, the theorem prover employed is based on proof nets, and the characterisation of different calculi is taken care of by labelling the formulae. For substructural calculi stronger than L, much

of the complexity (perhaps too much) is shifted to the label unification procedures. A strategy for improving such procedures by compiling labels into higher-order logic programming clauses is presented in [19] for NL and L. However, a comprehensive solution to the problem of binding label unification, a problem which arises as we move from sequents to labelled proof nets, has not been presented yet. Moreover, as discussed in [14], if we consider that the system is to be used as a parser, as a tool for linguistic study, the proof net style of derivation does not provide the clearest or most intuitive display of the proofs.[10]

In our approach, the burden of parsing is not so concentrated in label unification but is more evenly divided between the theorem prover and the algebraic checker. This is mainly due to the fact that the system allows for a controlled degree of non-determinism, present in the $\sigma$-rules, which enables us to reduce the introduction of variables in the labelling expressions to a minimum. We believe this represents an improvement on previous attempts. Besides this, controlling composition via bounded backtrack opens the possibility of implementing heuristics reflecting linguistic and contextual knowledge. In fact, we verify that, under the appropriate application of rules, we are able to eliminate the $\beta$-rule for a class of theorems.

Proposition 2 (Elimination Theorem) *All closed LKE-trees derivable by the application of the set of rules* $\mathcal{R} = \{\alpha_{(i)},...,\alpha_{(iii)},\sigma_{(i)},...,\sigma_{(vi)},\beta\}$ *can be also derived from* $\mathcal{R} - \{\beta\} + \{$ *assoc* $\}$.

The proof of this proposition can be done by defining an *abstract Gentzen relation*, proving a substitution lemma with respect to the labelling algebra (as in [7]), and showing that our consequence relation is closed under the relevant Gentzen conditions even if no $\beta$ rule is employed. The proof appeals to the fact that no formula signed by F can occur in the sequents on the left-hand side of the entailment relation, since the calculi presented here do not have negation.[11] We believe that this result shows that, even though LKE label unification might be computationally expensive for substructural logics in general, the system seems to be well suited for categorial logics. We refer the reader to [15] for a more comprehensive discussion of these issues.

## 3 Conclusions and Further Work

We have described a framework for the study of categorial logics with different degrees of expressivity on a uniform basis, providing a tool for testing the adequacy of different CGs to a variety of linguistic phenomena. From a practical point of view, we have investigated the effectiveness and generality issues of a parsing strategy for CG opening an avenue for future developments. Moreover, we have pointed out some strategies for improving on efficiency and for dealing with more expressive languages, including structural modalities.

The architecture proposed seems promising. Its flexibility with respect to the variety of logics it deals with, and its modularity suggest some natural extensions to the present work. Among them: implementing a semantic module based on Curry-Howard correspondence between type deduction and $\lambda$-terms, adding local control of structural transformations (structural modalities) to the language, increasing expressivity in the information frames for covering calculi weaker than L (e.g. Dependency Categorial Grammar [20]), exploiting the derivational structure encoded in the labels to define heuristics for models of human attachment preferences etc. Problems for further investigation might include: the treatment of polymorphic types (by incorporating rules for dealing with quantification analogous to Smullyan's $\delta$ and $\gamma$ rules [10] [21]), and complexity issues regarding how the general architecture proposed here would behave under more standard theorem proving methods.

## References

[1] A. Ades and M. Steedman. On the order of words. *Linguistics and Philosphy*, 4:517–558, 1982.

[10] Proof nets and sequent normalisation have also been employed to get around spurious ambiguity (i.e. multiple proof for the same sentence, with the same semantics). Our approach does not exhibit this problem.

[11] Of course, without the $\beta$ rule not all open trees generated will constitute downward saturated sets, since they might contain formulae which are not completely analysed.

[2] K. Ajdukiewicz. Die syntaktische konnexität. *Studia Philosophica*, 1(1–27), 1935. (Translation in S. McCall (ed) *Polish Logic 1920-1939* Oxford).

[3] Association for Computational Linguistics. *7th Conference of the European Chapter of the ACL*, Dublin, Ireland, March 1995. Morgan Kaufmann.

[4] Guy Barry and Glyn Morrill. *Studies in Categorial Grammar*, volume 5 of *Edinburgh Working Papers in Cognitive Science*. Centre for Cognitive Science, 1990.

[5] J. Barwise, D. Gabbay, and C. Hartonas. On the logic of information flow. *Bulletin of the IGPL*, 3(1):7–50, 1995. http://www.mpi-sb.mpg.de/guide/staff/ohlbach/igpl/Bulletin.html.

[6] Johan van Benthem. The semantics of variety. In Wojciech Buszkowski, Witold Marciszewski, and Johan van Benthem, editors, *Categorial Grammar*, volume 25, chapter 6, pages 141–151. John Benjamins Publishing Company, Amsterdam, 1988.

[7] Marcello D'Agostino and Dov Gabbay. A generalization of analytic deduction via labelled deductive systems I: Basic substructural logics. *Journal of Automated Reasoning*, To appear. Revised in March 1994.

[8] Marcello D'Agostino and Marco Mondadori. The taming of the cut. *Journal of Logic and Computation*, 4:285–319, 1994.

[9] David Dowty. Type raising, functional composition, and non-constituent conjunction. In Deirdre Wheeler Richard T. Oehrle, Emmon Bach, editor, *Categorial Grammars and Natural Language Structures*, pages 153–197. Reidel Publishing Co, Dordrecht, 1988.

[10] Melvin Fitting. *First-order Logic and Automatic Theorem Proving.* Texts and Monographs in Computer Science. Springer-Verlag, New York, 1990.

[11] Dov M. Gabbay. LDS – Labelled Deductive Systems, volume 1 — foundations. Technical Report MPI-I-94-223, Max-Planck-Institut fiir Informatik, 1994.

[12] Mark Hepple Glyn Morrill, Neil Leslie and Guy Barry. Categorial deductions and structural operations. In Barry and Morrill [4], pages 1 – 21.

[13] Joachim Lambek. On the calculus of syntactic types. In *Proceedings of the Symposia in Applied Mathematics*, volume XII, pages 166–178, Providence, Rhode Island, 1961. American Mathematics Society.

[14] Neil Leslie. Contrasting styles of categorial derivations. In Barry and Morrill [4], pages 113–126.

[15] Saturnino F. Luz Filho and Patrick Sturt. A new approach to categorial theorem proving. In preparation.

[16] Michael Moortgat. *Categorial Investigations.* Foris Publications, Dordrecht, 1988.

[17] Michael Moortgat. Labelled deductive systems for categorial theorem proving. Technical Report OTS-WP-CL-92-003, OTS, Utrecht, NL, 1992.

[18] Michael Moortgat. Lecture notes on categorial grammar. Reader for the course on Categorial Grammar given at the 5th ESSLLI - University of Lisbon, August 1993.

[19] Glyn Morril. Higher-order logic programming of categorial deduction. In EACL95 [3], pages 133–140.

[20] Martin Pickering and Guy Barry. Dependency categorial grammar and coordination. *Linguistics*, 31(5):855–902, 1993.

[21] Raymond M Smullyan. *First-Order Logic*, volume 43 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer-Verlag, Berlin, 1968.

# A UNIFICATION–BASED ID/LP PARSING SCHEMA

**Frank Morawietz***
Universität Tübingen
Wilhelmstr. 113
72074 Tübingen
Germany
frank@sfs.nphil.uni-tuebingen.de

**Abstract**

In contemporary natural language formalisms like HPSG (Pollard and Sag 1994) the ID/LP format is used to separate the information on dominance from the one on linear precedence thereby allowing significant generalizations on word order. In this paper, we define unification ID/LP grammars. But as mentioned in Seiffert (1991) there are problems concerning the locality of the information determining LP acceptability during parsing. Since one is dealing with partially specified data, the information that is relevant to decide whether the local tree under construction is LP acceptable might be instantiated further during processing. In this paper we propose a modification of the Earley/Shieber algorithm on direct parsing of ID/LP grammars. We extend the items involved to include the relevant underspecified information using it in the completion steps to ensure the acceptability of the resulting structure. Following Sikkel (1993) we define it not as an algorithm, but as a parsing schema to allow the most abstract representation.

## 1 Introduction

The immediate dominance/linear precedence distinction was introduced into linguistic formalisms to encode word order generalizations by Gazdar, Klein, Pullum and Sag (1985) for GPSG and contemporary formalisms like HPSG (Pollard and Sag 1994) still want to express linearization facts with LP rules. But HPSG does not provide definitions for how to incorporate the ID/LP format into the formalism. This paper tries to handle the ID/LP distinction on another level. Instead of incorporating it into the theory, the information will be used during processing to determine validity of structures.

The simplest approach to parsing of ID/LP grammars is to fully expand the grammar into the underlying phrase structure grammar. But the expansion creates a huge number of grammar rules[1] which dominates the parsing complexity and can therefore not be a basis for a reasonable implementation. This parsing with the object grammar is called *indirect parsing*. On the other hand, there are approaches, called *direct parsing* which try to interleave the use of ID and LP rules. The approach taken in this paper is a result of augmenting this paradigm.

*The research presented in this paper was sponsored by Teilprojekt A8 of the SFB 340 of the Deutsche Forschungsgemeinschaft. I want to thank Tom Cornell, Dale Gerdemann, Thilo Götz, Frank Richter and Klaas Sikkel for helpful discussion. Needless to say, all errors and infelicities that remain are my own.

[1]For unification grammars the number of rules may even be infinite.

The concept of direct parsing was developed by Shieber (1984). He modifies Earley's algorithm (Earley 1970) to cope with ID/LP grammars. (Context–free) ID/LP grammars are defined by treating the formerly context–free rules as ID rules and by adding LP rules. The modification to the algorithm consists of two parts. The right hand side of a rule has no longer a fixed order, but is treated as a multiset. The predictor and the completer are limited to the LP acceptable structures by testing LP acceptability on the considered local trees, i.e. a nonterminal is extracted from the multiset and tested whether it may precede the remaining categories. The LP rules are taken and matched directly against proposed structures. Since this proposal obviously does not suffice to handle HPSG style grammars, it has to be augmented to feature based grammars as is done in Seiffert (1991).

Unification based ID/LP grammars are defined by augmenting the domain of the nonterminals to feature structures and by formalizing when an LP rules applies. This happens in case the LP elements subsume two categories contained in a local tree. A violation occurs if the category which is subsumed by the first LP element follows the category which is subsumed by the second LP element. The Earley/Shieber parser is then used almost unchanged. The only change is that nonterminals are not longer identical, so that application of rules is determined by unification. But consider the grammar in figure 1 (taken from Seiffert (1991)).[2]

$$
\begin{aligned}
\text{Lexicon} &= \left\{ \begin{array}{lcl}
\left[\textsc{cat}\ d\right] & \rightarrow & h \\
\left[\textsc{cat}\ e\right] & \rightarrow & i \\
\left[\begin{array}{ll}\textsc{cat} & f \\ \textsc{f1} & one\end{array}\right] & \rightarrow & j \\
\left[\begin{array}{ll}\textsc{cat} & g \\ \textsc{f2} & two\end{array}\right] & \rightarrow & k
\end{array} \right\} \\
\text{ID–Rules} &= \left\{ \begin{array}{lcl}
\left[\textsc{cat}\ a\right] & \rightarrow & \left[\begin{array}{ll}\textsc{cat} & b \\ \textsc{f} & \boxed{1}\end{array}\right], \left[\begin{array}{ll}\textsc{cat} & c \\ \textsc{f} & \boxed{1}\end{array}\right] \\
\left[\begin{array}{ll}\textsc{cat} & b \\ \textsc{f} & \left[\begin{array}{ll}\textsc{f1} & \boxed{2} \\ \textsc{f2} & \boxed{3}\end{array}\right]\end{array}\right] & \rightarrow & \left[\begin{array}{ll}\textsc{cat} & d \\ \textsc{f1} & \boxed{2}\end{array}\right], \left[\begin{array}{ll}\textsc{cat} & e \\ \textsc{f2} & \boxed{3}\end{array}\right] \\
\left[\begin{array}{ll}\textsc{cat} & c \\ \textsc{f} & \left[\begin{array}{ll}\textsc{f1} & \boxed{4} \\ \textsc{f2} & \boxed{5}\end{array}\right]\end{array}\right] & \rightarrow & \left[\begin{array}{ll}\textsc{cat} & f \\ \textsc{f1} & \boxed{4}\end{array}\right], \left[\begin{array}{ll}\textsc{cat} & g \\ \textsc{f2} & \boxed{5}\end{array}\right]
\end{array} \right\} \\
\text{LP–Rules} &= \left\{ \begin{array}{lcl}
\left[\textsc{f1}\ one\right] & \prec & \left[\textsc{f2}\ two\right] \\
\left[\textsc{cat}\ b\right] & \prec & \left[\textsc{cat}\ c\right]
\end{array} \right\} \\
\text{Start Symbol} &= \left[\textsc{cat}\ a\right]
\end{aligned}
$$

**Fig. 1.** Seiffert's example grammar

On input *ihjk* – which is not well formed – Seiffert's parser can not determine on the first pass whether the local tree (CAT *b*, (CAT *e*, CAT *d*)) is well formed since the information that is relevant for the LP rule, namely that *one* has to precede *two*, is not available yet. Only after the local trees (CAT *c*, (CAT *f*, CAT *g*)) and (CAT *a*, (CAT *b*, CAT *c*)) have been constructed, the information becomes accessible through structure sharing. The nonlocal flow can be seen in the parse tree given in figure 2.[3] The information *one* and *two* comes from the lexical entries for *f* and *g* and is passed to *e* and *d* via *c* and *b*. Only then is the information that the value for F2 at *e* is *two* and that the value for F1 at *d* is *one* available and the local tree (CAT *b*, (CAT

[2] To simplify notation, we are going to represent the trees of categories in the discussion of this example in term notation with just the category feature present.

[3] For readability reasons the arrows are drawn just for one value, namely *two*.

$e$, CAT $d$)) has to be ruled out since the LP rule $[\text{F1 } one] \prec [\text{F2 } two]$ is violated.

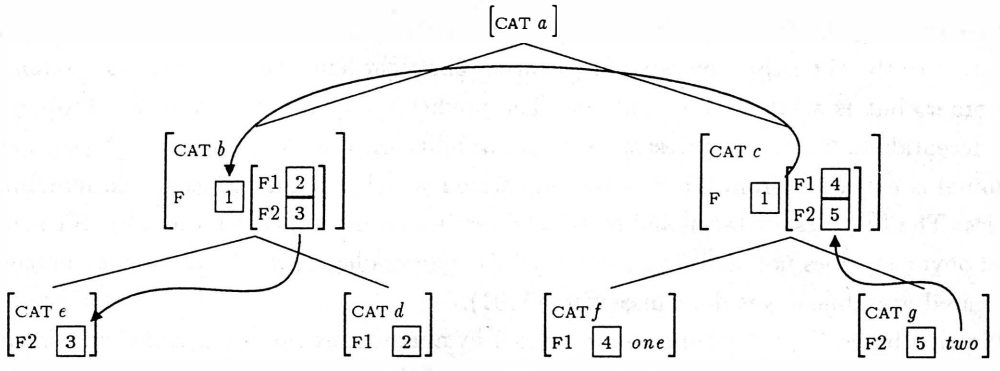



**Fig. 2.** The parse tree for input **ihjk* and Seiffert's grammar (see figure 1)

This phenomenon makes it necessary for Seiffert to take the resulting parse tree and check it for any remaining LP violations. Clearly this is inefficient and not desired.

Although this paper is not concerned with linguistics, we nevertheless try to argue to some extent that the problem is indeed not only a technical one. Nonlocality is a well known phenomenon in natural language, but so far without any influence on word order. This section tries to sketch the fact that in a reasonably large fragment for German such nonlocal word order phenomena do exist. Since this is not the main goal of the paper, the presentation needs to be brief. It can be understood without an exact knowledge of the linguistic theories involved if one accepts the claims made during the discussion. But for a complete understanding of the details involved, we assume some knowledge of the analysis of partial verb phrases by Hinrichs and Nakazawa (1993), on word order by Lenerz (1977) and the empirical analyses by Richter and Sailer (1995).

Some data on word order in German sentences seem to suggest that there exists a connection between the main verb and the order of its complements. The example sentences in 1 do indicate the different behaviour of the verbs *geben* (to give) and *überlassen* (to leave).

**1 Example Sentences**

*(1)* a. Karl wird dem Kind das Geschenk geben wollen.
Karl will the child the present give want.
Karl will want to give the child the present.

b. *Karl wird das Geschenk $\begin{bmatrix}\text{RHEM } plus\\ \text{CASE } acc\end{bmatrix}$ dem Kind $\begin{bmatrix}\text{RHEM } minus\\ \text{CASE } dat\end{bmatrix}$ geben wollen.

*(2)* a. Karl wird das Geschenk dem Kind überlassen wollen.
Karl will the present the child leave want.
Karl will want to leave the child the present.

b. *Karl wird dem Kind $\begin{bmatrix}\text{RHEM } plus\\ \text{CASE } dat\end{bmatrix}$ das Geschenk $\begin{bmatrix}\text{RHEM } minus\\ \text{CASE } acc\end{bmatrix}$ überlassen wollen.

The *(a)* versions of the sentences do not exhibit any limitation on the order of the complements. The *(b)* versions reflect the fact that they do not allow the first NP to be rhematic in the sense of Lenerz (1977). With the verb *geben* this disallows the order of the accusative

rhematic NP preceding the dative not rhematic NP, and in the case of *überlassen* the dative rhematic NP must not precede the accusative not rhematic NP. So the acceptance of the word order of the accusative and dative object seems to depend on the main verb and whether the first of those objects is to be rhematic. To express this in a grammar, one would have to ensure that there exists some connection from the main verb to the complements. We do not go into any detail how this connection could be formulated since it suffices to know that one has to exist. We do give a sketch for a parse tree for one of the example sentences in figure 3; $V_a$ stands for an auxiliary verb, $V_m$ for the main verb and $V_k$ for a verbal complex. These parse trees rely on the analysis of German verb phrases in Hinrichs and Nakazawa (1993).

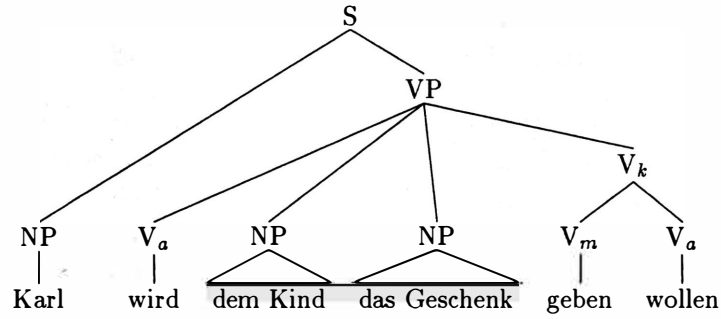



**Fig. 3.** A sketched parse tree for the sentence *Karl wird dem Kind das Geschenk geben wollen.*

As one can see, the NP complements are not in one local tree with the main verb due to the necessary argument raising induced by the auxiliary in the verbal complex. But this is not sufficient to cause the problem presented previously. If one considers direct parsing of such a sentence, the local tree labeled VP in the figure 3 which has to decide on the LP acceptability of the order of the two NP complements is not completed unless the local tree of the verbal complex has been recognized. And by this, the necessary connection of the NP complements to the main verb can be validated so that the local tree containing the NP complements would not be completed.

A further complication is necessary to cause the problem – the topicalization of the verbal complex. The data to support the claim is not as sharp as for the non topicalized sentences, although it is plausible that topicalization does not alter the behaviour of the main verbs concerning the word order of their complements. The examples are given below.

**2 Example Sentences**

*(3)* a. Geben wollen wird Karl dem Kind das Geschenk.

b. *Geben wollen wird Karl das Geschenk $\begin{bmatrix}\text{RHEM } plus\\ \text{CASE } acc\end{bmatrix}$ dem Kind $\begin{bmatrix}\text{RHEM } minus\\ \text{CASE } dat\end{bmatrix}$.

*(4)* a. Überlassen wollen wird Karl das Geschenk dem Kind.

b. *Überlassen wollen wird Karl dem Kind $\begin{bmatrix}\text{RHEM } plus\\ \text{CASE } dat\end{bmatrix}$ das Geschenk $\begin{bmatrix}\text{RHEM } minus\\ \text{CASE } acc\end{bmatrix}$.

As can be seen in the sketched parse tree in figure 4, the verbal complex is completely independent of the local tree containing the NP complements. This creates the desired constellation. If one considers right to left traversal of the input string for parsing, as is done for example in ALE (Carpenter 1993), the local tree whose mother is labeled VP has to be completed before

the verbal complex is parsed. Therefore the information which main verb appears in the verbal complex is not yet known. The local tree would be LP acceptable although the whole parse tree may later contain information that the VP in question was not LP acceptable.

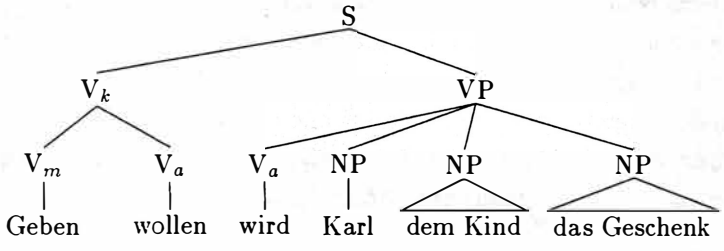



**Fig. 4.** A sketched parse tree for the sentence *Geben wollen wird Karl dem Kind das Geschenk.*

Although the presentation is somewhat sketchy, we showed that in natural language there may well be cases where the technical problem noted by Seiffert does indeed occur. Since all the mechanisms involved are present in current HPSG analyses of German and their interaction is complex, one may conclude that such phenomena are not easily detected by the grammar writer and can not necessarily be avoided. Therefore each parser dealing with unification based ID/LP grammars has to take care of those nonlocal phenomena.

## 2 Unification ID/LP Grammars

We are going to formalize unification ID/LP grammars very similarly to the approach taken in Sikkel (1993), because this seems close to a minimal definition which can be extended to almost all the versions of unification grammars. We use the notation and the feature machinery developed there, see chapter 8. This assumes disjoint, linearly ordered sets $Feat$ of features and $Const$ of constants. A standard representation of DAGs is used to define feature structures and from there it is straightforward to define a feature lattice. The definitions are augmented to deal with composite feature structures, i.e. multiply rooted feature structures and composite feature lattices to allow for the representation of phrase structure rules. The formalization assumes a context free backbone, but nothing in the treatment of the problem hinges on that.

**3 Definition (Unification ID/LP Grammar)** *A Unification ID/LP grammar is a tuple* $\mathcal{G} = (G, \Phi, \varphi_0, \varsigma_0, W, Lex)$ *such that:*

$G = \langle N, \Sigma, ID, LP, S\rangle$ *is a (context free) ID/LP grammar.*

$\Phi = \phi(Feat, Const)$ *is the lattice of feature structures.*

$\varphi_0 : ID \rightarrow \Phi$ *is a function that assigns a composite feature structure to each rule in ID.*

$\varsigma_0 : LP \rightarrow \Phi$ *is a function that assigns a composite feature structure to each rule in LP.*

$W$ *is the set of the word forms, i.e. lexical entries.*

$Lex : W \rightarrow \wp(\Phi)$ *is a function that assigns a set of feature structures to each lexical entry.*

*We write* $V$ *for* $N \cup \Sigma$, $\varphi_0(X_i)$ *for* $\varphi_0(X_0 \rightarrow X_1, \ldots, X_k)|_{X_i}$, $\varsigma_0(A)$ *for* $\varsigma_0(A \prec B)|_A$ *in case of the LP rules, and* $a, b, \ldots$ *for elements of* $W$. *It is assumed that* CAT $\in Feat$, $V \subseteq Const$ *and* $V \cap W = \emptyset$. *ID and LP are multisets of rules to allow different feature structures to occur with the same context free backbone. Take* $\varsigma(LP)$ *to be the set of all the* $\varsigma_0(X_i)$, *then* $\langle \varsigma(LP), \prec\rangle$ *is a transitive, asymmetric, irreflexive relation. Furthermore it is required that*

$\forall 0 \leq i \leq k\ \varphi_0(X_i).\textsc{cat} = X_i$ *for each production* $X_0 \rightarrow X_1, \ldots, X_k$ *and likewise for the LP rules and the lexical entries. Additionally we demand that* $\varphi(a).\textsc{cat} \in \Sigma$. *The class of unification ID/LP grammars is denoted as* $\mathcal{G}_{\mathcal{ID}/\mathcal{LP}}$.

As already mentioned in the previous section, an LP rule applies under subsumption. We assume a sequence of categories and check whether they are informative enough to determine applicability of the LP rules.

**4 Definition (applies)** *An LP rule* $A \prec B$ *applies to a sequence* $\sigma = \sigma_1 \ldots \sigma_n$, $\sigma_k \in V$ *iff there exist* $i, j$, $i \neq j$, $1 \leq i, j \leq n$ *such that* $\varsigma_0(A) \sqsubseteq \varphi(\sigma_i)$ *and* $\varsigma_0(B) \sqsubseteq \varphi(\sigma_j)$.

If no LP rule applies to a structure such that it causes a violation of the imposed order, it has to be LP acceptable.

**5 Definition (LP acceptable)** *A sequence* $\sigma$ *is LP acceptable iff no LP rule applies to it with* $j < i$.

We overload the definition of LP acceptability in the sense that we use it on sets of sequences of feature structures as well. The need for this will become clear later.

A requirement for the definition of parse trees is the ability to generate ordered sequences from the multisets which represent the right hand sides of ID rules. Therefore we assume a permutation function on sequences of nonterminals called *permute*. Then we can define parse trees for ID/LP grammars by including the demand that each local tree conforms to the LP rules. We assume a standard tree formalization like the one from Partee, ter Meulen and Wall (1990) and augment it by a composite feature structure for the parse tree.[4]

**6 Definition (parse tree)** *A grammar* $\mathcal{G} = (G, \Phi, \varphi_0, \varsigma_0, W, Lex)$ *generates a parse tree* $\langle \tau, \varphi(\tau) \rangle$ *for the sentence* $a_1 \ldots a_n$ *iff*

*the root is labeled with the initial symbol* $S$ *of* $G$ *and* $\varphi_0(S) \sqsubseteq \varphi(S)$;

$\forall i\ 1 \leq i \leq n\ a_i \in W$ *and* $\varphi'(a_i) \in Lex(a_i)$ *such that* $\varphi'(a_i) \sqsubseteq \varphi(a_i)$;

*for all subtrees* $\tau_i \in \tau$, *where* $T_0$ *immediately dominates* $T_1 \ldots T_n$, *there is a rule* $T_0 \rightarrow \alpha$ *in ID such that* $T_1 \ldots T_n \in permute(\alpha)$ *and*
$\forall i\ 1 \leq i \leq n\ \varphi_0(T_i) \sqsubseteq \varphi(T_i)$ *and*
*the yield of the local tree* $\tau_i$ *is LP acceptable.*

The parse trees defined thus are not necessarily adequately decorated, i.e. might have some features and values that do not derive from the application of the rules and lexical entries. The formalization of this can be taken from Sikkel (1993). We just demand that parse trees are minimal in this sense. This allows the specification of the language generated by a unification ID/LP grammar.

**7 Definition (Language of the Grammar** $(L(\mathcal{G}))$**)** *The language generated by* $\mathcal{G}$ *is defined as* $L(\mathcal{G}) = \{w = w_1 \ldots w_n |$ *there exists a parse tree* $\langle \tau, \varphi(\tau) \rangle$ *with yield* $w_1 \ldots w_n \,\&\, \varphi(\tau) \neq \bot\}$.

The parsing problem for the class of grammars in $\mathcal{G}_{\mathcal{ID}/\mathcal{LP}}$ can now be stated as follows.

**8 The Parsing Problem** *Given a sentence* $a_1 \ldots a_n \in W^*$ *and a grammar* $\mathcal{G} \in \mathcal{G}_{\mathcal{ID}/\mathcal{LP}}$, *find all the parse trees for the sentence with result* $\varphi(S)$.

This completes the definitions which present unification ID/LP grammars. But for the solution to the problem of nonlocal feature passing, we need some auxiliary definitions.

Since it may happen that a structure is not specific enough to determine LP acceptability,

[4] Note that structure sharing is indicated with a plain $=$, whereas equality is denoted with $\doteq$.

we have to formalize when these cases do occur. This is done by weak application. It tests for the success of a unification, i.e. whether the information could be there.

**9 Definition (weakly applies)** *An LP rule $A \prec B$ weakly applies to a sequence $\sigma = \sigma_1 \ldots \sigma_n$, $\sigma_k \in V$ iff there exist $i, j$, $i \neq j$, $1 \leq i, j \leq n$ such that $\varsigma_0(A) \sqcup \varphi(\sigma_i)$ and $\varsigma_0(B) \sqcup \varphi(\sigma_j)$.*

Since we now can tell when there might be enough information to determine LP acceptability, it is straightforward to define possible LP violation. But if a feature structure is subsumed by an element of an LP rule, it unifies with it as well. So we can not exclude the cases of application of an LP rule from our definition of weak application. So our definition has to presuppose that the structure is LP acceptable, under definition 5, i.e. that no LP rule applies to it and is violated. In that case we do not need to check for further (possible) violations since the sequence is ruled out anyway. If we could exclude *application* from *weak application*, this would not be necessary. And additionally, the weak application has to result in a violation if it is to be relevant at all.

**10 Definition (possibly LP violated)** *A sequence $\sigma$ is possibly LP violated iff it is LP acceptable, but an LP rule weakly applies to it with $j < i$.*

Furthermore we have to add some bookkeeping device, the LP store. This is a list of the pending structures, i.e. those that are not specific enough to be determined yet[5]. The LP store is assumed to be part of each feature structure that is assigned to a rule. The formalism presented in Sikkel (1993) can be extended by demanding that every composite feature structure that represents a rule includes a list representation of the LP store. The following defines the union of two such stores. It will be used in the deduction steps.

**11 Definition (Store union ($\bowtie$))** *The union of an LP store $\Theta$ with another LP store $\Omega$ with resulting LP store $\Delta$ ($\varphi(\Theta) \bowtie \varphi(\Omega) \doteq \varphi(\Delta)$) is recursivly defined as*
$\varphi(\Delta) \doteq \varphi(\Omega)$ *if* $\varphi(\Theta).\text{HD} \doteq e\_list$
$\varphi(\Delta) \doteq \varphi(\Theta).\text{TL} \bowtie \varphi(\Omega)$ *if* $\varphi(\Theta).\text{HD} \doteq \varphi(A) \wedge \varphi(A) \in \varphi(\Omega)$
$\varphi(\Delta).\text{HD} \doteq \varphi(A) \wedge \varphi(\Delta).\text{TL} \doteq \varphi(\Theta).\text{TL} \bowtie \varphi(\Omega)$ *if* $\varphi(\Theta).\text{HD} \doteq \varphi(A) \wedge \varphi(A) \notin \varphi(\Omega)$

## 3 An ID/LP Parsing Schema

Sikkel (1993) develops a framework called *parsing schemata* for comparing parsing algorithms by abstracting from control information and data structures contained in their specifications. We are going to use this framework because it is general enough to allow the specification of nontrivial algorithms without requiring implementation specific details. The presentation of a parsing schema is done in three steps: firstly we define two kinds of parsing systems which then allow the specification of the parsing schema. A parsing system is a logical deduction system restricted to a single given grammar and a single given sentence.

**12 Parsing System** *A parsing system for some grammar $G$ and input string $a_1, \ldots, a_n$ is a triple $\mathcal{P}(\mathcal{I}, \mathcal{H}, \mathcal{D})$ with $\mathcal{I}$ a set of items; $\mathcal{H}$ a finite set of items (not necessarily a subset of $\mathcal{I}$), the hypotheses; and $\mathcal{D}$ a set of deduction steps $\eta_1, \ldots, \eta_m \vdash \xi$ with $\eta_i \in \mathcal{I} \cup \mathcal{H}$ for $0 \leq i \leq m$ and $\xi \in \mathcal{I}$.*

There can be several kinds of items. But in the following, we are using an expanded version of the familiar items of an Earley or chart parser, e.g. $[A \rightarrow \alpha \bullet \beta, i, j]$. Now we extend this

[5] This is necessary since the feature logic does not support set valued feature structures.

definition to an uninstantiated parsing system. This obviates the need for a fixed input string.

**13 Uninstantiated Parsing System** *An uninstantiated parsing system for a grammar $G$ is a triple $\langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$ with $\mathcal{H}$ a function that assigns a set of hypotheses to each string $a_1, \ldots, a_n \in \Sigma^*$ such that $\langle \mathcal{I}, \mathcal{H}(a_1, \ldots, a_n), \mathcal{D} \rangle$ is a parsing system.*

It can easily be seen that in all our examples the function $\mathcal{H}$ is defined as follows: $\mathcal{H}(a_1, \ldots, a_n) = \left\{ [a, i-1, i] \,\middle|\, a = a_i, 1 \leq i \leq n \right\}$. Therefore we are not going to make a difference in the following between uninstantiated parsing systems and parsing systems.

**14 Parsing Schema** *A parsing schema for a class of grammars $\mathcal{G}$ is a function that assigns an (uninstantiated) parsing system to every grammar in $\mathcal{G}$.*

In the ID/LP parsing schema, we will interleave the two steps of generation via permute and testing via LP acceptability completely so that acceptance can be determined in just one traversal of the input. To achieve this, the deduction steps allowed by the direct parsing algorithm have to be modified. Augmenting Seiffert's and Shieber's definitions in the last section, we defined a relation that tells when an LP rule *weakly applies* to a sequence. Whenever this occurs, we add a sequence of feature structures to the LP store. This store may be thought of as information how further instantiation of the rule in question may be restricted. The restriction is not immediately obvious, but as soon as the rule is further instantiated, a test for LP acceptability is performed and the hypothesis discarded if the structure contained an LP violation. This added sequence of categories is structure shared with the sequence the parser assumes to be LP acceptable. So any changes that are made to the sequence via structure sharing are present on the LP store as well. In each deduction step the elements on the LP store have to be LP acceptable which achieves the desired effect. Since this LP violation can only happen in completion, this completion is ruled out, thereby preventing the parser from constructing invalid structures. The old structure is not discarded since it may be used in some other way which does not lead to an LP violation. Additionally, the algorithm has to percolate the LP stores via *store union* to ensure that all the edges do contain the pending LP information of their subcomponents.[6]

Before proceding to the presentation of the ID/LP parsing schema, a short remark on notation. The items have been augmented by the LP store $\Delta$. We abbreviate in the following the notation of items by indexing them with lowercase greek letters and then refering to the index, i.e. $[A \rightarrow \alpha \bullet \beta, i, j, \Delta]_\xi$ is referred to as $\xi$. $\varphi(\xi)$ refers to the composite feature structure of the item, naturally excluding the string indices.

**15 ID/LP Parsing System** *A parsing system $\mathcal{P}(\mathcal{I}_{ID/LP}, \mathcal{H}_{ID/LP}, \mathcal{D}_{ID/LP})$ for a unification ID/LP grammar $\mathcal{G} = (G, \Phi, \varphi_0, \varsigma_0, W, Lex) \in \mathcal{G}_{\mathcal{ID}/\mathcal{LP}}$ is defined by*

$$\mathcal{I}_{ID/LP} = \left\{ [A \rightarrow \alpha \bullet \beta, i, j, \Delta]_\xi \,\middle|\, \begin{array}{l} A \rightarrow \alpha\beta \in ID \\ 0 \leq i \leq j \\ \varphi_0(A \rightarrow \alpha\beta) \sqsubseteq \varphi(\xi) \\ \varphi(\xi) \not\equiv \bot \\ \alpha \text{ and } \Delta \text{ are LP acceptable} \end{array} \right\}$$

$$\mathcal{H}_{ID/LP} = \left\{ [a, j-1, j, \{\}] \,\middle|\, \varphi(a) \in Lex(a_j) \right\}$$

[6] These processes may be improved upon in an actual implementation, see Morawietz (1995) for the implementation in TROLL (Gerdemann, Götz and Morawietz forthcoming).

$$\mathcal{D}^{Init} = \left\{ \vdash [S \rightarrow \bullet\gamma, 0, 0, \{\}]_\xi \;\middle|\; \varphi(\xi) \doteq \varphi_0(S \rightarrow \gamma) \right\}$$

$$\mathcal{D}^{Scan} = \left\{ \begin{array}{c} [A \rightarrow \alpha \bullet \beta a \delta, i, j, \Delta]_\eta, [a, j, j+1, \{\}]_\zeta \vdash \\ [A \rightarrow \alpha a \bullet \beta \delta, i, j+1, \Delta]_\xi \end{array} \;\middle|\; \varphi(\xi) \doteq \varphi(\eta) \sqcup \varphi(\zeta) \right\}$$

$$\mathcal{D}^{Compl1} = \left\{ \begin{array}{c} [A \rightarrow \alpha \bullet \beta B \delta, i, j, \Delta]_\eta, [B \rightarrow \gamma \bullet, j, k, \Omega]_\zeta \vdash \\ [A \rightarrow \alpha B \bullet \beta \delta, i, k, \Theta]_\xi \end{array} \;\middle|\; \begin{array}{l} \varphi(\xi) \doteq \varphi(\eta) \sqcup \varphi(\zeta) \\ \varphi(\Theta_\xi) \doteq \varphi(\Delta_\eta) \bowtie \varphi(\Omega_\zeta) \\ \varphi((\alpha B)_\xi) \textit{ is not possibly LP violated} \end{array} \right\}$$

$$\mathcal{D}^{Compl2} = \left\{ \begin{array}{c} [A \rightarrow \alpha \bullet \beta B \delta, i, j, \Delta]_\eta, [B \rightarrow \gamma \bullet, j, k, \Omega]_\zeta \vdash \\ [A \rightarrow \alpha B \bullet \beta \delta, i, k, \Theta]_\xi \end{array} \;\middle|\; \begin{array}{l} \varphi(\xi) \doteq \varphi(\eta) \sqcup \varphi(\zeta) \\ \varphi((\alpha B)_\xi) \textit{ is possibly LP violated} \\ \varphi(\Theta_\xi) \doteq \varphi((\alpha B)_\xi) \bowtie \varphi(\Delta_\eta) \bowtie \varphi(\Omega_\zeta) \\ \varphi((\alpha B)_\Theta) = \varphi((\alpha B)_\xi) \end{array} \right\}$$

$$\mathcal{D}^{Pred} = \left\{ \begin{array}{c} [A \rightarrow \alpha \bullet \beta B \delta, i, j, \Delta]_\eta \vdash \\ [B \rightarrow \bullet\gamma, j, j, \{\}]_\xi \end{array} \;\middle|\; \varphi(\xi) \doteq \varphi(B_\eta) \angle \Psi_0(B) \sqcup \varphi_0(B \rightarrow \gamma) \right\}$$

$$\mathcal{D}_{ID/LP} = \mathcal{D}^{Init} \cup \mathcal{D}^{Scan} \cup \mathcal{D}^{Compl1} \cup \mathcal{D}^{Compl2} \cup \mathcal{D}^{Pred}$$

Note that our item definition demands that both the recognized sequence of categories and the LP store have to be LP acceptable. Therefore we do not have to include these demands in the deduction steps. The sequence of feature structures to the right of the bullet is treated as a multiset. This is indicated by picking a random element from it, i.e. both $\beta$ and $\delta$ in the deduction steps can be empty or of any other cardinality. Furthermore it is important that in the definition of $\mathcal{D}^{Compl2}$ the new LP store is the result of the union of the old stores plus the sequence of categories found. This sequence is reentrant with the recognized one before the bullet. In the definition of $\mathcal{D}^{Pred}$, we included a restrictor ($\angle\Psi_0(B)$) to deal with the nontermination of the predictor. For the definition of a default restrictor, see again Sikkel (1993). Since this can be generated automatically, we did not have to include the definition of a restrictor in our grammar.

**16 ID/LP Parsing Schema** *A parsing schema* **P** *for the class of grammars* $\mathcal{G}_{\mathcal{ID}/\mathcal{LP}}$ *is a function that assigns an ID/LP parsing system* $P$ *to every grammar* $\mathcal{G} \in \mathcal{G}_{\mathcal{ID}/\mathcal{LP}}$.

Next, we are going to present the crucial steps an algorithm along the lines of the proposed schema takes to solve the problem of nonlocal feature passing. We simplify the notation in the sense that we do not differentiate between the context–free backbone and the corresponding feature structures any more. This gives us the following format for the edges. First comes the left hand side of the rule, followed by an arrow and the categories which have been recognized already, enclosed in square brackets. The dot separates those from the set of the categories yet to be found which is enclosed in curly brackets. Next are the start and end position of the recognized part of the edge. And last there is the LP store, again in curly brackets. The whole edge is enclosed in square brackets. We are going to write the values of tags, if there are any, only behind the crucial occurences.

The edges of interest are depicted in figure 5. We assume that the algorithm is supposed to

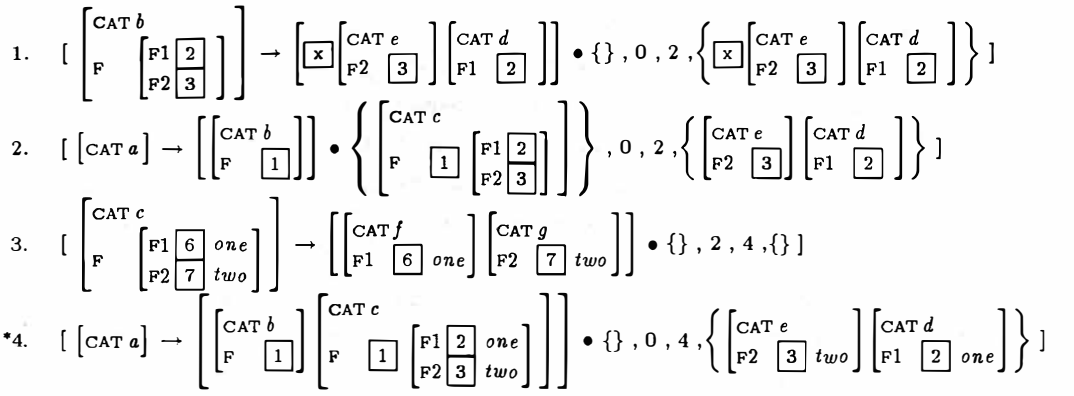


**Fig. 5.** Edges from the chart for input word *ihjk* and Seiffert's grammar (see figure 1)

recognize the input *ihjk* with the grammar given in figure 1. At some point in the computation it has to create the local tree represented by the edge in 1. Note that since the algorithm cannot decide on the LP acceptability of the recognized structure and because it is possibly LP violated, it adds the sequence of categories to the LP store via structure sharing (tag $\boxed{x}$).[7] This information is percolated to further edges which rely on the computation of 1, for example the one in 2. Independently we construct the local tree represented by the edge in 3. At some point it becomes necessary for the algorithm to complete those two edges 2 and 3. But this can not be done as shown by the edge in 4. It is included in this chart only for the reason to illustrate the point. It would *not* be constructed by the algorithm. The reason for this is as follows. The completer would try to build the edge from the edges 2 and 3 by moving the recognized feature structure with the cat feature *c* to the list to the left of the dot. This feature structure is the result of the unification between those two feature structures from the edges 2 and 3 which have the category *c*. The unification forces identity between the reentrancies $\boxed{2}$ and $\boxed{6}$ and $\boxed{3}$ and $\boxed{7}$ respectivly. This results in the values *one* and *two* on the elements of the LP store which violates the LP rule $[\text{F1 } one] \prec [\text{F2 } two]$. Since it is not possible to construct a parse tree in another way, the sentence is not in the language of Seiffert's grammar.

## 4 Conclusions

In this paper, we defined unification ID/LP grammars and gave instructions for a parsing algorithm, thereby solving the problem of nonlocal feature passing. This enables linguists to write ID/LP grammars and gives the computational linguists a clear formalization and the means to directly parse such grammars in one pass.

Our approach presents a local solution to a nonlocal phenomenon. It uses the existing mechanism of the test for LP acceptability to detect the violations as early as they can be discovered. No complex backtracking into the chart or of the whole process is necessary, since it is not the local tree where the LP violation occurs that is seen as being invalid, but rather the combination of the information from this local tree with another local tree. It is acceptable to combine the local tree with another structure in a way that does not lead to failure.

[7] This tag is special in the sense that it denotes sharing of a sequence rather than a single feature structure. So the tag is a simplification for a number of tags which would indicate the sharing of all the roots of the categories involved.

Furthermore, the solution is neutral toward processing direction, i.e. bottom up versus top down, and processing mode, i.e. parsing versus generation. The way the solution is presented here is for a top down parser, but nothing in the way the problem is handled requires any of the special mechanisms involved with top down parsing. In fact, the first prototype to be implemented was a bottom up parser. One can just leave out the deduction steps that represent prediction. And since Earley's algorithm may be used for generation as well as for parsing (Gerdemann 1991), it seems straightforward to incorporate our treatment of the ID/LP formalism for generation.

Nevertheless, some open questions remain. Some are induced by linguistic needs and some by formal considerations. Although we are not really concerned with it, from a linguistic point of view there are some demands that have not been met by the proposed approach. In some linguistic analyses, only binary branching structures are considered, for example in Uszkoreit (1984). This would lead to problems with direct parsing from a linguistic point of view because in our approach only local trees can be ordered by LP rules. If one is dealing with these binary structures, the domain of application of LP rules has to be enlarged, for example to something along the lines of a head domain, as in Engelkamp, Erbach and Uszkoreit (1992). In our approach, this would have to be done by some extra mechanism that collects the categories in the list under the recognized feature until the projection becomes itself the argument.

A technical problem that remains to be tackled is the incorporation of set–valued feature structures into the formalization. This would simplify the definitions since we would not have to use store union in the combination of two edges but could simply unify them. But since this is a new issue and an open problem, we cannot solve it here.

The implementation of the schema for the typed feature system TROLL (Gerdemann et al. forthcoming) shows that a system may use this kind of approach to provide the means for a concise representation of ID/LP grammars. The other approaches found in the literature such as for example Engelkamp et al. (1992) who incorporate a treatment of ID/LP into an HPSG style formalism by altering the ID schemata and by the creation of principles, cannot deal with the problem of nonlocal feature passing. Since in their approach the lexical items determine what must or must not appear to their left and right respectivly, they suffer from further instatiation of LP relevant information in the same way that the parser proposed by Seiffert does. All other proposed treatments of the ID/LP paradigm try to incorporate it into the formalism of HPSG directly. This reflects the different viewpoints between a rigid (denotational) formalization of HPSG and (operational) considerations which use unification grammars. Our approach defines ID/LP grammars in the latter sense.

# References


Carpenter, B. (1993). ALE The Attribute Logic Engine, User's Guide, *Technical Report*, Laboratory for Computational Linguistics, Carnegie Mellon University, Pittsburgh, PA 15213.

Earley, J. (1970). An Efficient Context-Free Parsing Algorithm, *Comm. ACM 13.2* pp. 94–102.

Engelkamp, J., Erbach, G. and Uszkoreit, H. (1992). Handling Linear Precedence Constraints by Unification, *ACL Proceedings, 23th Annual Meeting*, pp. 201—208.

Gazdar, G., Klein, E., Pullum, G. K. and Sag, I. A. (1985). *Generalized Phrase Structure Grammar*, Harvard University Press, Cambridge, Massachusetts.

Gerdemann, D. D. (1991). *Parsing and Generation of Unification Grammars*, PhD thesis, University of Illinois.

Gerdemann, D. D., Götz, T. W. and Morawietz, F. (forthcoming). *Troll – Type Resolution System – Fundamental Principles & User's Guide*, Universität Tübingen.

Hinrichs, E. W. and Nakazawa, T. (1993). Partial-VP and Split-NP Topicalization in German - An HPSG Analysis, *Arbeitspapiere des SFB 340 No 58*, Universität Tübingen.

Lenerz, J. (1977). *Zur Abfolge nominaler Satzglieder im Deutschen*, TBL Verlag Gunter Narr, Tübingen.

Morawietz, F. (1995). Formalization and Parsing of Unification–Based ID/LP Grammars, *Arbeitspapiere des SFB 340 No 68*, Universität Tübingen. (Available in WWW under "http://www.sfs.nphil.uni-tuebingen.de/~frank").

Partee, B. H., ter Meulen, A. and Wall, R. E. (1990). *Mathematical Methods in Linguistics*, Vol. 30 of *Studies in Linguistics and Philosophy*, Kluwer Academic Publishers.

Pollard, C. J. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*, University of Chicago Press and CSLI Publications.

Richter, F. and Sailer, M. (1995). *Remarks on Linearization*, Master's thesis, Universität Tübingen.

Seiffert, R. (1991). Unification–ID/LP Grammars: Formalization and Parsing, *in* O. Herzog and C.-R. Rollinger (eds.), *Text Understanding in LILOG*, Springer, Berlin, pp. 63–73.

Shieber, S. M. (1984). Direct Parsing of ID/LP Grammars, *Linguistics and Philosophy* **7**: 135–154.

Sikkel, K. (1993). *Parsing Schemata*, PhD thesis, University of Twente, The Netherlands. (Available in WWW under "http://orgwis.gmd.de/~sikkel/papers/PhD.html").

Uszkoreit, H. (1984). *Word Order and Constituent Structure in German*, PhD thesis, University of Texas, Austin. Published as Lecture Notes No. 8, CSLI Lecture Notes Series (1987).

# PARSING WITHOUT GRAMMAR

Shinsuke Mori and Makoto Nagao
Section of Electronics and Communication, Kyoto University
Yoshida-honmachi, Sakyo, Kyoto, 606-01 Japan
{mori,nagao}@kuee.kyoto-u.ac.jp

**Abstruct**

We describe and evaluate experimentally a method to parse a tagged corpus without grammar modeling a natural language on context-free language. This method is based on the following three hypotheses. 1) Part-of-speech sequences on the right-hand side of a rewriting rule are less constrained as to what part-of-speech precedes and follows them than non-constituent sequences. 2) Part-of-speech sequences directly derived from the same non-terminal symbol have similar environments. 3) The most suitable set of rewriting rules makes the greatest reduction of the corpus size. Based on these hypotheses, the system finds a set of constituent-like part-of-speech sequences and replaces them with a new symbol. The repetition of these processes brings us a set of rewriting rules, a grammar, and the bracketed corpus.

## 1 Introduction

A standard approach to a natural language analysis is to characterize it with a set of rules, a grammar. Given the difficulty in developing a grammar manually, it is necessary to build a method for automatic grammar induction. One of the most promising results of grammar inference is based on the inside-outside algorithm, which can be used to train a stochastic context-free grammar. It is an extension of the forward-backward algorithm. [Pereira and Schabes, 1992] and [Schabes *et al.*, 1993] proposed a method to infer the parameters of a stochastic context-free grammar from a partially parsed corpus and evaluated the results. [Brill, 1993] describes another technique for grammar induction: "the system learns a set of ordered transformations and applies it to a new sentence."

These two methods profit from corpora annotated with syntactic structure, the Penn Treebank [Marcus and Santorini, 1993]. Corpora in the Penn Treebank have two sorts of additional data: the part-of-speech of each word and the syntactic structure of each sentence. The first stage of building a corpus is an automatic tagging or parsing and the second the manual correction of errors. Since the accuracy of current parsers is not satisfactory, the manual correction of parsing results is an arduous task. As for tagging, however, the method of [Church, 1988], applied to build the Penn Treebank, is reported as "95-99% correct, depending on the definition of correct" and another tagger developed by [Brill, 1992] marks almost the same accuracy.

It follows that it is worth trying to induce a grammar from corpora without syntactic structure, that is to say, to parse corpora using only part-of-speech information. Only a few attempts, however, have been made so far at grammar induction from an unbracketed corpus. [Magerman and Marcus, 1990] proposes a parsing system using mutual information statistics and a manually written "distituent grammar," a list of tag pairs which cannot be adjacent within a constituent, such as (*prep noun*). [Brill and Marcus, 1992] also proposes a technique

for grammar induction.

The operations of extracting a grammar from a tagged corpus and applying it to the very same corpus is equal to the process of parsing the corpus without a grammar. In this paper, we propose a new method to parse a tagged corpus without a grammar based on the following three hypotheses:

1. Part-of-speech sequences on the right-hand side of a rewriting rule are less constrained as to what part-of-speech precedes and follows them than non-constituent sequences.

2. Part-of-speech sequences directly derived from the same non-terminal symbol have similar environments.

3. The most suitable set of rewriting rules makes the greatest reduction of the corpus size.

The initial state of the corpus is part-of-speech sequences. The algorithm finds a group of constituent-like part-of-speech sequences in the corpus and produces rewriting rules which have the part-of-speech sequences on the right-hand side and the same non-terminal symbol on the left-hand side. Applying these rewriting rules to the corpus, the algorithm makes the corpus shorter in terms of number of symbolic units, more parsed. The system repeats these processes until it cannot find any more part-of-speech constituent sequences. Then the system stops, with the corpus parsed and a set of rewriting rules, i.e. a grammar, created.

In subsequent sections, first we discuss the three hypotheses, secondly describe the algorithm, thirdly present results and compare them to other recent results in automatic phrase structure induction, and finally conclude this research and discuss future works.

## 2 Three Hypotheses

In the theory of formal language, the rewriting rules determine whether a sequence of alphabets is a sentential form or not. Therefore the language, a set of sentential forms, reflects the characteristics of the rewriting rules. Since our method models natural language on context-free language, a corpus is regarded as a set of sentential forms. It follows that a corpus reflects the characteristics of the rewriting rules of the natural language. In the following parts of this section, we present an example of the corpus we used, define some symbols for explanation and discuss the three hypotheses on the relation between a corpus and the rewriting rules derived from this point of view.

### 2.1 Corpus and Symbol Definitions

For our main experiment, we used part-of-speech (POS) sequences from the Wall Street Journal (WSJ) in the Penn Treebank. Table 1 shows its POS tagset and Figure 1 is an example

```
Nekoosa would n't be a  diversification .
NNP     MD    RB  VB DT NN             .
( ( ( Nekoosa ) would n't ( be ( a diversification ) ) ) . )
```

Figure 1: An example sentence in WSJ

Table 1: Penn Treebank POS tagset

| | | | | | |
|---|---|---|---|---|---|
| 1. | CC | Coordinating conjunction | 21. | RBR | Adverb, comparative |
| 2. | CD | Cardinal number | 22. | RBS | Adverb, superlative |
| 3. | DT | Determiner | 23. | RP | Particle |
| 4. | EX | Existential *there* | 24. | SYM | Symbol |
| 5. | FW | Foreign word | 25. | TO | *to* |
| 6. | IN | Preposition or subordinating conjunction | 26. | UH | Interjection |
| | | | 27. | VB | Verb, base form |
| 7. | JJ | Adjective | 28. | VBD | Verb, past tense |
| 8. | JJR | Adjective, comparative | 29. | VBG | Verb, gerund or present participle |
| 9. | JJS | Adjective, superlative | | | |
| 10. | LS | List item marker | 30. | VBN | Verb, past participle |
| 11. | MD | Modal | 31. | VBP | Verb, non-3rd person singular present |
| 12. | NN | Noun, singular or mass | | | |
| 13. | NNS | Noun, plural | 32. | VBZ | Verb, 3rd person singular present |
| 14. | NNP | Proper noun, singular | | | |
| 15. | NNPS | Proper noun, plural | 33. | WDT | Wh-determiner |
| 16. | PDT | Predeterminer | 34. | WP | Wh-pronoun |
| 17. | POS | Possessive ending | 35. | WP$ | Possessive wh-pronoun |
| 18. | PRP | Personal pronoun | 36. | WRB | Wh-adverb |
| 19. | PRP$ | Possessive pronoun | 37. | , | Comma |
| 20. | RB | Adverb | 38. | . | Sentence-final punctuation |

sentence from WSJ. We use the POS information in the second line in this figure for grammar induction and the syntactic structure in the third line for evaluation.

Our method models natural language on context-free language, which is described by a grammar $G = (N, T, P, S)$, where $N$ is the set of non-terminal symbols, $T$ is the set of terminal symbols, $P$ is the set of rewriting rules and $S$ is the start symbol. Since our method regards the corpus as a set of POS sequences, the set of terminal symbols $T$ is equal to the part-of-speech tagset of the Penn Treebank. Non-terminal symbols are, however, introduced by the system and they aren't elements of the syntactic tagset of the Penn Treebank. For the subsequent explanation, we make the following definitions:

$$pos \in T, \quad syn \in N, \quad tag \in N \cup T$$

$$\boldsymbol{syn} \in N^+, \quad \boldsymbol{pos} \in T^+, \quad \boldsymbol{tag} \in (N \cup T)^+$$

where $X^+ = X^* - \{\varepsilon\}$ generally.

### 2.2 Constituent-like Part-of-speech Sequence

The first step of this method is to extract sequences of one ore more POS from the corpus to form the right-hand side of rewriting rules. The following is the first hypothesis according to which the system extracts constituent-like POS sequences.

- POS sequences on the right-hand side of a rewriting rule are less constrained as to what POS precedes and follows them than non-constituent sequences.

To explain this hypothesis concretely, let us consider the following two different POS sequences: $\boldsymbol{pos}_a$ which appears on the right-hand side of a rewriting rule and $\boldsymbol{pos}_x = \boldsymbol{pos}_{x1} \cdot \boldsymbol{pos}_{x2}$ which

doesn't appear in any rewriting rule. In this case, let us say that $G$ may contain the following rewriting rules:

1. $syn_a \rightarrow \boldsymbol{pos}_a$
2. $syn_b \rightarrow syn_c \cdot syn_d$
3. $syn_c \rightarrow \boldsymbol{pos}_{c'} \cdot \boldsymbol{pos}_{x1}$
4. $syn_d \rightarrow \boldsymbol{pos}_{x2} \cdot \boldsymbol{pos}_{d'}$

These rules tell us that the POSs which precede or follow $\boldsymbol{pos}_a$ are more loosely restricted than those of $\boldsymbol{pos}_x$. In fact, $\boldsymbol{pos}_a$ can appear where $syn_a$ appears, while $\boldsymbol{pos}_x$ requires the last three rewriting rules to be produced, in the following way:

$$syn_b \Rightarrow syn_c \cdot syn_d \Rightarrow \boldsymbol{pos}_{c'} \cdot \underbrace{\boldsymbol{pos}_{x1} \cdot \boldsymbol{pos}_{x2}}_{\boldsymbol{pos}_x} \cdot \boldsymbol{pos}_{d'}$$

In this case, the POS which can precede $\boldsymbol{pos}_x$ is restricted to the last POS of $\boldsymbol{pos}_{c'}$ and the POS which follows $\boldsymbol{pos}_x$ is restricted to the first POS of $\boldsymbol{pos}_{d'}$. This is the foundation of the first hypothesis.

As a measure of constituency of a particular POS sequence, we use the entropies of the probability distributions of the POSs which precede or follow it. These entropies are calculated using the following equations:

$$H_l(\boldsymbol{pos}) = -\sum_i P(pos_i \cdot \boldsymbol{pos} \mid \boldsymbol{pos}) \log P(pos_i \cdot \boldsymbol{pos} \mid \boldsymbol{pos})$$

$$H_r(\boldsymbol{pos}) = -\sum_i P(\boldsymbol{pos} \cdot pos_i \mid \boldsymbol{pos}) \log P(\boldsymbol{pos} \cdot pos_i \mid \boldsymbol{pos})$$

We call $H_l$ and $H_r$ the left-side entropy and the right-side entropy respectively. The conditional probabilities $P(pos_i \cdot \boldsymbol{pos} \mid \boldsymbol{pos})$ and $P(\boldsymbol{pos} \cdot pos_i \mid \boldsymbol{pos})$ are computed from the frequencies of $\boldsymbol{pos}$, $pos_i \cdot \boldsymbol{pos}$ and $\boldsymbol{pos} \cdot pos_i$ in the corpus using the following equations.

$$P(pos_i \cdot \boldsymbol{pos} \mid \boldsymbol{pos}) = \frac{P(pos_i \cdot \boldsymbol{pos})}{P(\boldsymbol{pos})} = \frac{f(pos_i \cdot \boldsymbol{pos})}{f(\boldsymbol{pos})}$$

$$P(\boldsymbol{pos} \cdot pos_i \mid \boldsymbol{pos}) = \frac{P(\boldsymbol{pos} \cdot pos_i)}{P(\boldsymbol{pos})} = \frac{f(\boldsymbol{pos} \cdot pos_i)}{f(\boldsymbol{pos})}$$

Therefore, $n$-gram statistics, the frequencies of all the symbol sequences appearing in the corpus, are applicable to compute the entropies. To calculate $n$-gram statistics for arbitrary $n$, we adopted the algorithm proposed in [Nagao and Mori, 1994]. Notice that $n$ is more than one and does not exceed the length of the longest sentence in the corpus, because $n$-grams containing a symbol for final punctuation mark never appear in rewriting rules, except for ones with $S$ on the left-hand side.

In addition to entropy, we propose another measure of constituency: the ratio of delimiters which precede or follow the POS sequence in question. In our method the delimiters are sentence-final punctuation mark and comma.

To extract constituent-like POS sequences from the corpus, we set threshold values for the entropies ($H_{min}$) and for the delimiter ratios ($Pd_{min}$). From the discussion above, the following four inequalities are the conditions for a constituent-like POS sequence:

1. $H_l(\boldsymbol{pos}) \geq H_{min}$
2. $H_r(\boldsymbol{pos}) \geq H_{min}$
3. $Pd_l(\boldsymbol{pos}) = P(\text{"."} \cdot \boldsymbol{pos} \mid \boldsymbol{pos}) + P(\text{","} \cdot \boldsymbol{pos} \mid \boldsymbol{pos}) \geq Pd_{min}$
4. $Pd_r(\boldsymbol{pos}) = P(\boldsymbol{pos} \cdot \text{"."} \mid \boldsymbol{pos}) + P(\boldsymbol{pos} \cdot \text{","} \mid \boldsymbol{pos}) \geq Pd_{min}$

Under the conditions $H_{min} = 3$ and $Pd_{min} = 0.05$, 429 POS sequences are extracted. Table 2 shows 20 of them in order of frequency.

Table 2: Samples of extracted part-of-speech sequences

| $f$ | $H_l$ | $H_r$ | $Pd_l$ | $Pd_r$ | $pos$ | $f$ | $H_l$ | $H_r$ | $Pd_l$ | $Pd_r$ | $pos$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 54724 | 3.1 | 3.3 | 0.16 | 0.19 | NNP | 13327 | 3.6 | 3.4 | 0.06 | 0.08 | VBN |
| 39375 | 3.6 | 3.6 | 0.05 | 0.27 | NNS | 9519 | 3.8 | 3.5 | 0.08 | 0.34 | JJ·NNS |
| 23758 | 3.2 | 3.5 | 0.20 | 0.17 | DT·NN | 9385 | 3.2 | 3.2 | 0.07 | 0.22 | IN·NNP |
| 20088 | 3.3 | 3.4 | 0.24 | 0.25 | NNP·NNP | 9278 | 3.4 | 3.5 | 0.09 | 0.24 | IN·DT·NN |
| 19055 | 3.1 | 3.8 | 0.09 | 0.06 | VBD | 7753 | 3.5 | 3.0 | 0.33 | 0.05 | PRP |
| 18460 | 4.2 | 4.0 | 0.17 | 0.16 | RB | 7487 | 3.0 | 3.3 | 0.15 | 0.25 | DT·JJ·NN |
| 14550 | 3.7 | 3.1 | 0.06 | 0.17 | CD | 7296 | 3.8 | 3.5 | 0.09 | 0.33 | NN·NNS |

### 2.3 Similarity between Constituents

The second step is to cluster the POS sequences extracted in the first step. The following is our second hypothesis, which gives a measure of this clustering.

- POS sequences directly derived from the same non-terminal symbol have similar environments.

To explain the background of this hypothesis, let us consider the case in which the grammar contains the following rewriting rules.

$$syn_e \rightarrow \boldsymbol{pos}_{e1}, \quad syn_e \rightarrow \boldsymbol{pos}_{e2}$$

This means that POS sequences which can appear to the left or right of the two POS sequences $\boldsymbol{pos}_{e1}$ and $\boldsymbol{pos}_{e2}$ are those which can also be found to the left or right of $syn_e$. It must reasonably be expected that two constituent-like POS sequences which are derived from a single non-terminal symbol will have similar probability distributions for the POSs which precede and follow them. We choose, as a measure of similarity to cluster extracted POS sequences, the Euclidean distance between two probability distributions. The clustering is composed of the following three processes (see Figure 2).

1. Produce a graph whose nodes correspond to POS sequences and whose arcs correspond to the Euclidean distance between two POS sequences.
2. Delete arcs whose value is greater than a threshold value ($D_{min}$).
3. Decompose the graph into connected components by examining connectivity.

Each connected component corresponds to a cluster and the nodes to their elements. Table 3 shows the result of clustering under the conditions $H_{min} = 3$, $Pd_{min} = 0.05$ *and* $D_{min} = 0.25$. In this table, "area" is a value given to each cluster which is calculated by summing the product of the length and the frequency of each element [1].

### 2.4 Selecting Rewriting Rules

After clustering, a new set of rewriting rules can be obtained from any cluster by putting each of the POS sequences on the right-hand side of a rule and introducing a new single non-terminal symbol on the left-hand sides. Next, the system applies them to reduce the corpus for

[1] More precisely, the frequency of shorter sequences is reduced by considering their overlap with longer ones.

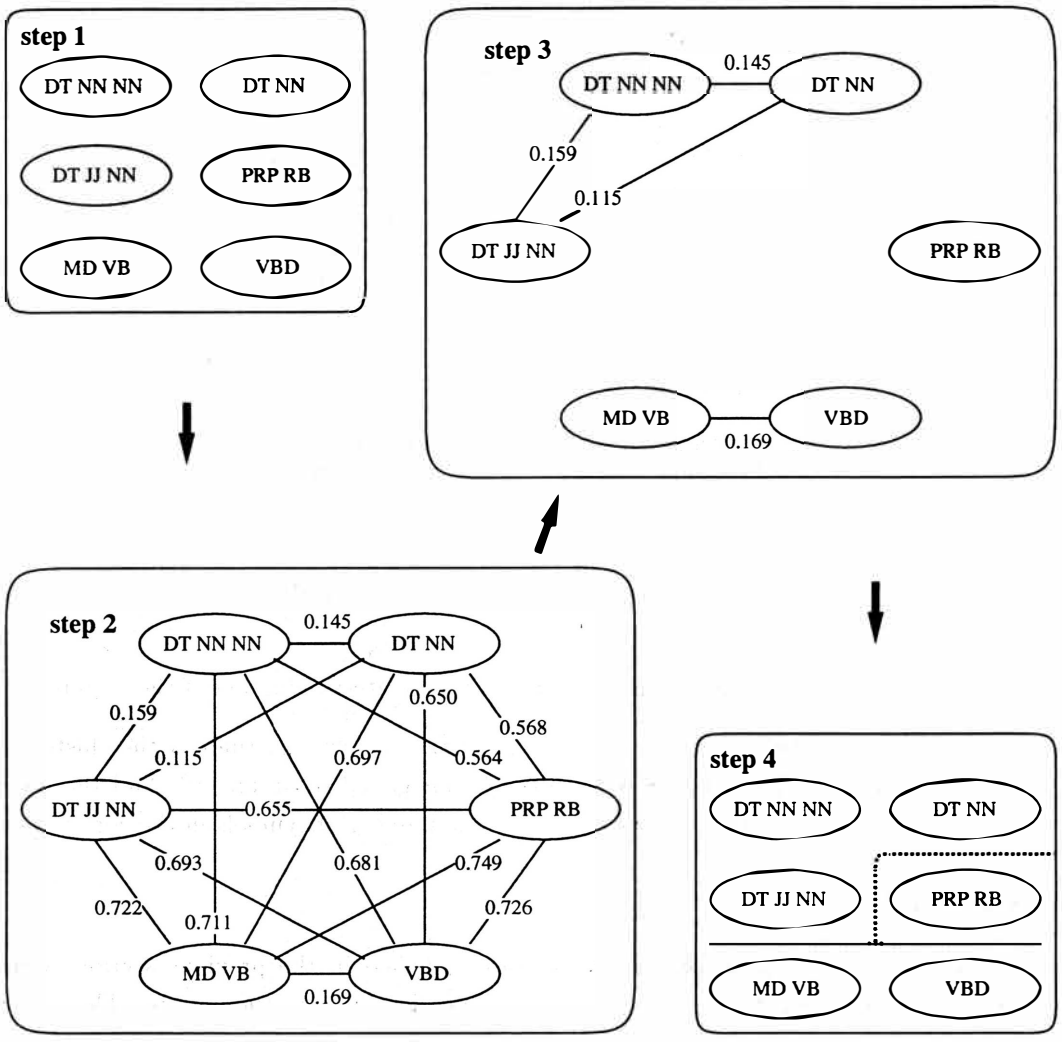



Figure 2: Clustering part-of-speech sequences ($D_{min} = 0.25$)

further extraction of rewriting rules which may contain non-terminal symbols on their right-hand side. It is possible, however, that two or more rewriting rules will conflict with each other. Let us consider the following two rewriting rules made from the first element of the second cluster in Table 3 and the third element of the third cluster, where $syn_2$ and $syn_3$ are non-terminal symbols introduced for the second cluster and the third cluster respectively.

$$syn_2 \rightarrow DT \cdot NN \tag{1}$$

$$syn_3 \rightarrow IN \cdot DT \cdot NN \tag{2}$$

In this case, rule (2) can be represented using rule (1), as follows.

$$syn_3 \rightarrow IN \cdot syn_2$$

Considering cases like this, we can conclude that changing all the clusters into rewriting rules and applying them to the corpus would disturb the appropriate extraction of rewriting rules. For this reason, the system selects only one cluster from the output of the second step. We

Table 3: An example of result of clustering

| area | $f$ | $H_l$ | $H_r$ | *pos* | area | $f$ | $H_l$ | $H_r$ | *pos* |
|---|---|---|---|---|---|---|---|---|---|
| 20260 | 19055 | 3.1 | 3.8 | VBD | 80317 | 6553 | 3.2 | 3.4 | IN·NN |
| | 1205 | 3.2 | 3.8 | RB·VBD | | 3722 | 3.1 | 3.6 | IN·NNS |
| 86473 | 23758 | 3.2 | 3.5 | DT·NN | | 9278 | 3.4 | 3.5 | IN·DT·NN |
| | 3905 | 3.0 | 3.7 | DT·NNS | | 1499 | 3.6 | 3.7 | IN·DT·NNS |
| | 2137 | 3.1 | 3.5 | PRP$·NN | | 1771 | 3.1 | 3.3 | IN·JJ·NN |
| | 7487 | 3.0 | 3.3 | DT·JJ·NN | | 2555 | 3.2 | 3.3 | IN·JJ·NNS |
| | 1102 | 3.1 | 3.7 | DT·JJ·NNS | | 1326 | 3.4 | 3.3 | IN·DT·NN·NN |
| | 1106 | 3.2 | 3.5 | DT·NN·IN·DT·NN | | 3283 | 3.0 | 3.5 | IN·DT·JJ·NN |

"area" is a value given to each cluster which is calculated by summing the product of the length and the frequency of each element.

chose the "area" of the cluster as the measure for selecting the best cluster. This is based on the third hypothesis presented below.

- The most suitable set of rewriting rules makes the greatest reduction of the corpus size.

One may expect that a large value for area means that the rewriting rules of the cluster are basic symbol sequences which appear regardless of their global environment. And the larger their area is, the more their application will reduce the number of symbols in the corpus.

## 3 Algorithm

We developed a system, based on the hypotheses discussed in the previous section, which extracts a set of rewriting rules from a corpus and applies them to the corpus. The system executes the following four processes repeatedly:

1. Compute $n$-gram statistics on the corpus.
2. Extract part-of-speech sequences whose frequency is more than $f_{min}$ and which meet the conditions $H_{min} = 3$ and $Pd_{min} = 0.05$.
3. Cluster the part-of-speech sequences and select a cluster to produce a set of rewriting rules.
4. Rewrite the corpus, applying the rewriting rules.

In the experiment, the initial value of $f_{min}$ is equal to 2,000. If no POS sequence is extracted in the second process, $f_{min}$ is multiplied by 0.9 and the second process is repeated with the new $f_{min}$. If $f_{min}$ becomes less than 50, the system stops. Below, we describe more precisely the last two processes.

### 3.1 Produce Rewriting Rules

As we described above, the system extracts constituent-like POS sequences and clusters them depending on their distributional similarity. Next it selects the cluster which has the largest "area" in the corpus to produce rewriting rules. The last process is to select a non-terminal to put on the left side of rewriting rules. There are two cases, depending on the number of

POS sequences in the cluster which consist of a single non-terminal symbol. If there is exactly one, that non-terminal symbol is put on the left-hand side to produce rewriting rules, and the rewriting rule which would have that non-terminal symbol on both sides is erased. In the other case, i.e. the number of POS sequences composed of a single non-terminal symbol is zero or more than one, the system introduces a new non-terminal symbol and puts it on the left-hand side to produce rewriting rules.

### 3.2 Rewrite Corpus

After having produced a set of rewriting rules, the system applies them to every sentence in the corpus. At this point, there are two problematic situations. The first one is that a rewriting rule is applicable in more than one way. For example, suppose that the rewriting rule is $syn_1 \rightarrow tag_1 \cdot tag_1$ and the following symbol sequence exists in the corpus:

$$\ldots tag_1 \cdot tag_1 \cdot tag_1 \ldots$$

The rule is applicable to both the first and the last $tag_1 \cdot tag_1$ at the same time. To handle cases like this, the system applies each rewriting rule simply from left to right.

The second case is that two or more rewriting rules conflict with each other. For example, suppose that there are two rewriting rules such as

$$syn_1 \rightarrow tag_1 \cdot tag_2 \tag{3}$$

$$syn_1 \rightarrow tag_1 \cdot tag_2 \cdot tag_3 \tag{4}$$

and the following symbol sequence exists in the corpus:

$$\ldots tag_1 \cdot tag_2 \cdot tag_3 \ldots$$

In this case both of the rules are applicable. To avoid this conflict, the system applies rewriting rules in the order of the length of their right-hand side. In this example, only rule (4) is applied.

## 4 Results

We conducted experiments on the sentences in WSJ, which are composed of the POS tags in Table 1. The corpus contains 24,678 sentences and 549,247 tags. The average sentence length is, therefore, 22.3 tags. Figure 3 shows the distribution by length of sentences in the corpus.

We ran two experiments with different threshold values for the distance between two distributions (Exp. 1, $D_{min} = 0.20$ and Exp. 2, $D_{min} = 0.25$). The output of each experiment is the final state of the corpus and the extracted rewriting rules.

### 4.1 Accuracy of Parsing

First we discuss the final state of the corpus, which can be regarded as the parsing results. Figure 4 shows an example of a parsed sentence, where the symbols starting with "NT" are the non-terminal symbols introduced by the system.

For the evaluation of our experiments we have chosen the measure called "crossing parenthesis accuracy" which arose out of a parser evaluation workshop [Black *et al.*, 1991]. The measure is the percentage of POS constituents as output by our system which do not cross any

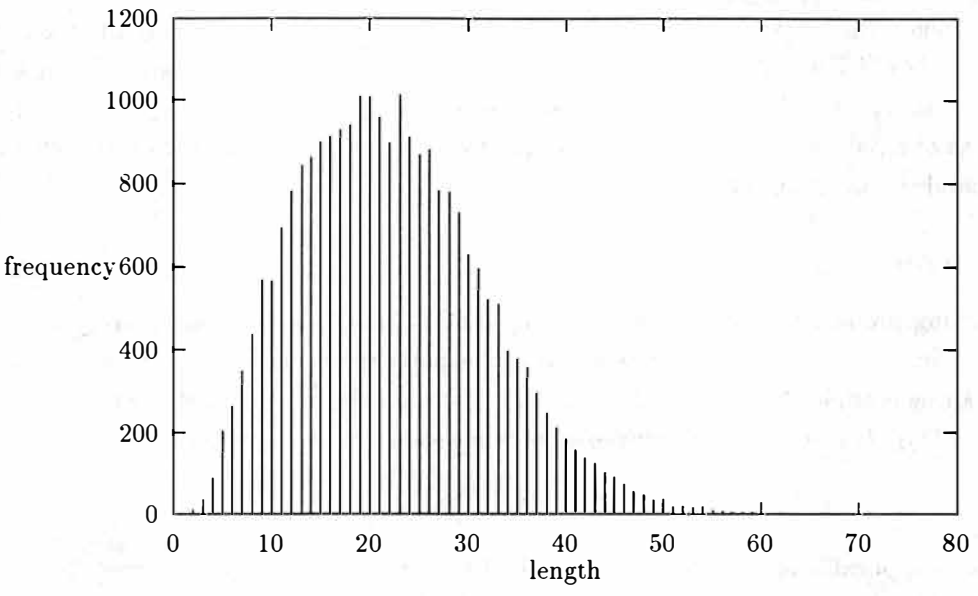



Figure 3: Sentence length distribution

constituents as parsed in the Penn Treebank. Table 4 shows the crossing parenthesis accuracy of our method. In this table, "right linear" means the right binary branching structure and "left linear" means the left binary branching structure.

Table 4: Result

| | # rules | # $N$ | C.P.A. |
|---|---|---|---|
| right linear | — | — | 56.3% |
| left linear | — | — | 24.3% |
| Exp. 1 | 956 | 19 | 73.7% |
| Exp. 2 | 594 | 25 | 74.8% |

C.P.A. = Crossing Parenthesis Accuracy

Since this method parses the corpus by applying rewriting rules which are extracted only from the POS sequence information, it is quite natural that the accuracy is less than that of grammar inference methods profiting from syntactic structure information [Pereira and Schabes, 1992] [Schabes *et al.*, 1993] [Brill, 1993]. In addition to the difference in source information, it must be noted that those methods were tested only on relatively short sentences, while in our experiments sentence length is not limited. From this viewpoint one may say that the accuracy of our method is sufficiently promising.

We must draw attention to another difference between our method and the others. They simply bracket sentences and are not able to introduce non-terminal symbols, while our method is able to infer non-terminal symbols without any information but a corpus.

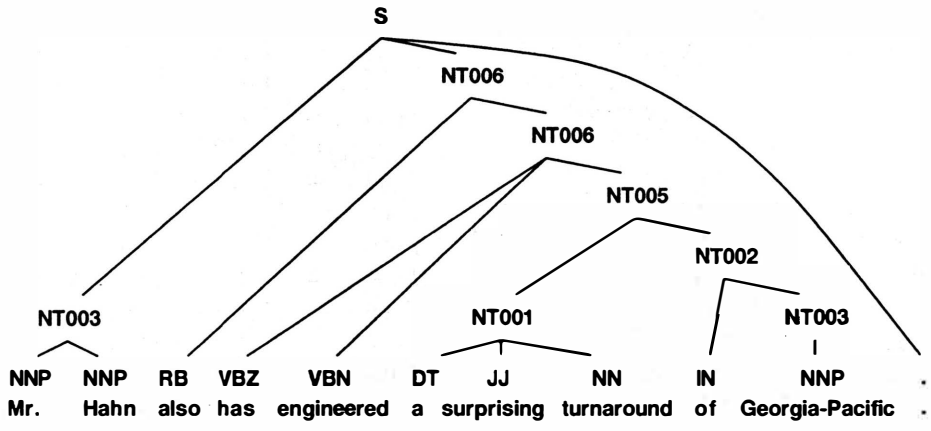



Figure 4: An example of a parsed sentence (Exp. 2)

### 4.2 Evaluation of Rewriting Rules

The other output of our system is a set of rewriting rules. Its quantitative evaluation is so difficult that we compare them with a general English grammar.

Table 5 shows some rewriting rules extracted in Experiment 2. Almost all the rules here are comprehensible from the viewpoint of general English grammar: $NT001$ represents noun phrase, $NT002$ prepositional phrase and so on. The interesting thing in this table is that the rewriting rules whose left-hand side is $NT006$ have various combinations of noun phrases and/or prepositional phrases on their right-hand side. They must be considered as verbal case frames. This indicates that it is possible to extract types of verbal case frames, which is defined a priori in various attempts at automatic case frame acquisition [Brent and Berwick, 1991] [Brent, 1991] [Manning, 1993].

Elsewhere, however, the grammar contains some inappropriate rewriting rules, as follows:

$$NT002 \rightarrow IN \cdot JJR \cdot NNS \cdot NNS$$

It would be better to represent this by the following rules instead.

$$NT002 \rightarrow IN \cdot NT001 \quad NT001 \rightarrow JJR \cdot NNS \cdot NNS$$

It is true that there are some rewriting rules like this in the grammar but, as we mentioned above, almost all the non-terminal symbols are comprehensible. Since our method assumes simple context-free language as the model and extracts a grammar based on hypotheses deduced from its characteristics, these results lead to the conclusion that structural features of language can be extracted using simple statistical methods.

It should also be added that it is difficult to evaluate a grammar because the evaluation depends on the language model of the parsed corpus to be compared.

## 5 Conclusions

In this paper, we have described a new method to parse a tagged corpus without grammar, modeling a natural language on context-free language. We proposed the following three hypotheses.

Table 5: Some rewriting rules extracted in Experiment 2

| | | | | | |
|---|---|---|---|---|---|
| $NT001$ | — | $PRP\$ \cdot NNS \cdot NNS$ | $NT005$ | → | $VBN \cdot NT002$ |
| $NT001$ | — | $PDT \cdot DT \cdot NNS \cdot NN$ | $NT005$ | → | $VBG \cdot NT001$ |
| $NT001$ | — | $NNP \cdot NNP \cdot POS \cdot NN$ | $NT005$ | → | $PRP \cdot MD \cdot VB \cdot NT002$ |
| $NT001$ | → | $DT \cdot VBG \cdot NNS \cdot NNS$ | $NT006$ | → | $VBZ \cdot NT005 \cdot NT005$ |
| $NT001$ | → | $DT \cdot JJ \cdot NN$ | $NT006$ | → | $VBZ \cdot NT005 \cdot NT002$ |
| $NT002$ | — | $TO \cdot NT001$ | $NT006$ | → | $VBZ \cdot NT005 \cdot NT001$ |
| $NT002$ | — | $TO \cdot VB \cdot NT001$ | $NT006$ | → | $VBZ \cdot NT005$ |
| $NT002$ | — | $TO \cdot VB \cdot NNS$ | $NT006$ | → | $VBZ \cdot NT002 \cdot NT005$ |
| $NT002$ | — | $RB \cdot TO \cdot NT001$ | $NT006$ | → | $VBZ \cdot NT002$ |
| $NT002$ | — | $IN \cdot NT001$ | $NT006$ | → | $VBZ \cdot NT001 \cdot NT001$ |
| $NT002$ | — | $IN \cdot RB \cdot NT001$ | $NT006$ | → | $VBZ \cdot NT001$ |
| $NT002$ | — | $IN \cdot NNS \cdot NN$ | $NT006$ | → | $VBD \cdot NT005 \cdot NT005$ |
| $NT002$ | — | $TO \cdot NT003$ | $NT006$ | → | $VBD \cdot NT002$ |
| $NT002$ | — | $RB \cdot NT002$ | $NT006$ | → | $MD \cdot VB \cdot NT001$ |
| $NT002$ | — | $RB \cdot IN \cdot NT003$ | $NT006$ | → | $MD \cdot VB \cdot NT001$ |
| $NT002$ | — | $IN \cdot NT003$ | $NT007$ | → | $VBZ \cdot VBN \cdot VBN$ |
| $NT003$ | — | $NNP \cdot NNP \cdot NNP$ | $NT007$ | — | $MD \cdot VB$ |
| $NT003$ | — | $NNP \cdot NNP$ | $NT008$ | → | $NNS \cdot NT006$ |
| $NT003$ | — | $NT003 \cdot CC \cdot NT003$ | $NT008$ | → | $NNS \cdot VBP \cdot NT005$ |
| $NT004$ | — | $NN \cdot NN$ | $NT008$ | — | $NNS \cdot VBP \cdot NT001$ |
| $NT004$ | — | $JJ \cdot NN \cdot NN$ | $NT009$ | → | $NT004 \cdot NT008$ |
| $NT005$ | — | $NT002 \cdot VBN \cdot NT002$ | $NT009$ | → | $NT003 \cdot JJ \cdot NT008$ |
| $NT005$ | — | $NT001 \cdot NT002$ | $NT010$ | → | $WRB \cdot PRP \cdot NT006$ |
| $NT005$ | — | $NT001 \cdot VBN \cdot NT002$ | $NT010$ | → | $VBG \cdot NT004 \cdot NT006$ |
| $NT005$ | — | $NT001 \cdot VBG \cdot NT002$ | $NT010$ | → | $VBG \cdot NT004$ |

1. POS sequences on the right-hand side of a rewriting rule are less constrained as to what POS precedes and follows them than non-constituent sequences.

2. POS sequences directly derived from the same non-terminal symbol have similar environments.

3. The most suitable set of rewriting rules makes the greatest reduction of the corpus size.

Then, we developed an algorithm based on these hypotheses which extracts rewriting rules and parses the sentences at the same time. The correctness of these hypotheses has been experimentally attested by the evaluation of the extracted grammar and parsing accuracy.

The method we have proposed in this paper is a bottom-up approach to grammar inference. On the other hand, a top-down approach such as [Magerman and Marcus, 1990] may also be important for grammar induction. Clustering techniques, e.g. [Hindle, 1990] and subcategorization techniques, e.g. [Brent, 1991] may bridge the gap between these two approaches. Combining these techniques including tagging techniques, larger amounts of syntactic information can be retrieved from unbracketed corpora or even from untagged corpora.

## References

[Black *et al.*, 1991] E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and

T. Strzalkowski. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the DARPA Speech and Natural Language Workshop*, 1991.

[Brent and Berwick, 1991] Michael R. Brent and Robert C. Berwick. Automatic acquisition of subcategorization frames from tagged text. In *Proceedings of the DARPA Speech and Natural Language Workshop*, 1991.

[Brent, 1991] Michael R. Brent. Automatic acquisition of subcategorization frames from untagged text. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 1991.

[Brill and Marcus, 1992] Eric Brill and Mitchell Marcus. Automatically acquiring phrase structure using distributional analysis. In *Proceedings of the DARPA Speech and Natural Language Workshop*, 1992.

[Brill, 1992] Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, 1992.

[Brill, 1993] Eric Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 1993.

[Church, 1988] Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, 1988.

[Hindle, 1990] Donald Hindle. Noun classification from predicate-argument structures. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, 1990.

[Magerman and Marcus, 1990] David M. Magerman and Mitchell P. Marcus. Parsing a natural language using mutual information statistics. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1990.

[Manning, 1993] Chiristopher D. Manning. Automatic acquisition of a large subcategorization dictionary from corpora. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 1993.

[Marcus and Santorini, 1993] Mitchell P. Marcus and Beatrice Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2), 1993.

[Nagao and Mori, 1994] Makoto Nagao and Shinsuke Mori. A new method of n-gram statistics for large number of n and automatic extraction of words and phrases from large text data of Japanese. In *Proceedings of the 15th International Conference on Computational Linguistics*, 1994.

[Pereira and Schabes, 1992] Fernando Pereira and Yves Schabes. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 1992.

[Schabes *et al.*, 1993] Yves Schabes, Michal Roth, and Randy Osborne. Parsing the Wall Street Journal with the inside-outside algorithm. In *Proceedings of the Sixth European Chapter of the Association for Computational Linguistics*, 1993.

# A FORMALISM AND A PARSER FOR LEXICALISED DEPENDENCY GRAMMARS

Alexis NASR[1]
DEA/IA2 Dassault Aviation
78, quai Marcel Dassault - cedex 300 - 92552 St.Cloud - FRANCE
and
TALANA-Université Paris 7
nasr@dassault-avion.fr

## 1 Introduction

This paper describes a formalism and a parser for dependency grammars. The grammars written in this formalism are composed of elementary structures, called elementary trees, that are associated to lexemes. Elementary trees can combine together through an operation called attachment. The parser builds a structural description of a sentence by combining the elementary trees associated to its words. An exponential parser is described, based on a stack. The inefficiency of this parser is studied, and another one, using a graph-structured stack, is proposed. Section 2 describes briefly the grammatical formalism: section 2.1 focuses on the structure of elementary trees, section 2.2 deals with word order: how to represent it in a dependency tree and how to constrain it in elementary trees, and section 2.3 introduces the attachment operation. Section 3 is devoted to parsing. The parser is introduced and its performances are discussed in 3.1. The graph-structured stack and its integration in the parser are described in 3.2.

## 2 The Formalism

In this section, we introduce the grammatical formalism we shall be using for parsing. The grammars written in this formalism consist of finite sets of dependency trees called *elementary trees*. An elementary tree describes a possible syntactic environment of a lexeme. This lexeme is refered to as the *lexical anchor* or simply *anchor* of the tree. There is therefore no strict distinction, in this framework, between the grammar and the lexicon. This is often the case in grammar formalisms proposed for dependency structures, as in [Hellwig, 1986] [Fraser, 1989] and [Starosta, 1986]. The trees can combine together through an operation called attachment. This operation will be used by the parser to build syntactic description of sentences.

### 2.1 Elementary Trees

An elementary tree is composed of the dependencies its anchor can be involved in, as governor or dependent when it occurs in a sentence. Such structures could include all the possible dependents and governors of an anchor, as it is done in [Fraser, 1989]. But such a description obliges the author of the grammar to anticipate in an elementary tree of a lexeme all of its dependents, modifiers as well as complements. Moreover, in a grammar composed of such elementary trees, each dependency is represented twice, once in the elementary tree of its governor and once in the elementary tree of its dependent. Another solution would be to represent in an elementary tree only the complements of the anchor while modifiers will have in their own elementary trees a description of their governor. This organisation can be compared with the factoring of recursion and dependencies achieved in Tree Adjoining Grammars [Joshi, 1987]. We have

[1]The author wants to thank Anne Abeillé, Farid Cerbah, Corinne Fournier and Owen Rambow for their fruitful comments on earlier versions of this paper.

represented in Figure 1 elementary trees corresponding respectively to a verb, a noun and an adjective. The lexical anchors are represented as black nodes. The labels of the dependencies are taken from [Mel'čuk and Pertsov, 1987].


Figure 1: Some elementary trees

Anchors may impose lexical, semantic or morphological restrictions on other words, these words are not always directly linked to the anchor by a syntactic dependency. In order to represent such restrictions in an elementary tree, we introduce these words as nodes in the elementary tree and state lexical, semantic or morphological constraints on them, causing, in some cases, the extension of the domain of locality. Elementary trees can hence have an arbitrarily extended domain of locality and exhibit quite complex patterns, as shown in Figure 2.

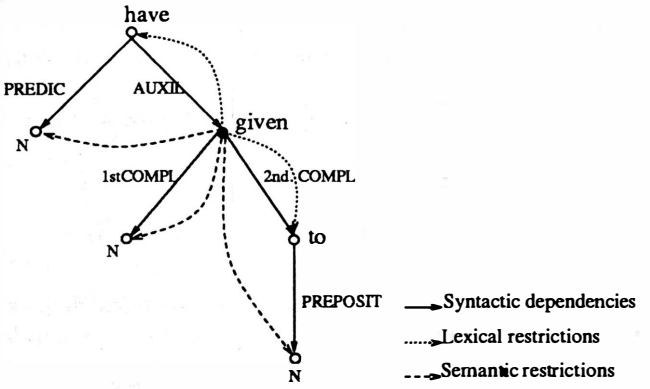



Figure 2: A complex elementary tree

### 2.1.1 Trees as Complex Feature Structures

The elementary trees are represented as complex feature structures and dependencies as complex attributes. The elementary tree of Figure 2 has been represented in Figure 3 as a complex feature matrix.

Such structures can be combined together by means of unification. This representational means and the unification operation impose that two attributes of a same structure cannot have the same label, preventing the representation of repeatable dependencies. Several ways to overcome this problem by redefining the unification operation are possible. But we will not develop this point inthis paper.

## 2.2 Word Order

The status of word order in dependency trees is not as clear as in phrase structure trees and is sometimes not represented at all, as in [Mel'čuk, 1988]. In our grammatical formalism, word order prescriptions, like all grammatical information, is represented in elementary trees. In this section, we propose some notational conventions for representing word order in dependency trees, in such a way that a tree is in correspondence with exactly one linear sequence of its nodes. This notational system will then be used to state constraints on word order in elementary trees.

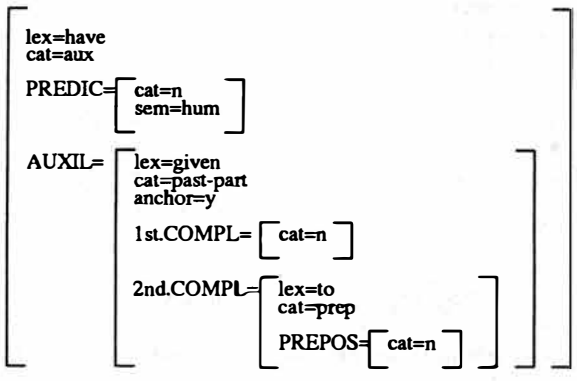



Figure 3: An elementary tree as a complex feature matrix

A dependency tree does not usually define a total order over its nodes: a single tree can correspond to many linear sequences of its nodes. Some enrichments of the trees have been proposed in [Hellwig, 1986] in order to represent linear order. These enrichments amount to indicating, for every node in the tree, its position, in the linear sequence, towards some other nodes of the tree. We shall call this information the ***linear coordinates of the node***:

LINEAR COORDINATES OF A NODE

1. its position towards its governor.
2. its position among its sister nodes.

A tree enriched with the linear coordinates of its nodes defines a total order among them, provided that every dependency respects the ***projectivity principle***[2].

THE PROJECTIVITY PRINCIPLE

A dependency is projective if its dependent is not separated, in the linear sequence, from the governor by anything apart from descendents of the governor.

We will say indifferently that a dependency or its dependent is projective. A tree made of projective dependencies will be said projective and its linear sequence as well. The projectivity principle drastically limits the number of different linear sequences corresponding to a single dependency tree. We have represented in the left part of Figure 4 a dependency tree and its unique corresponding linear sequence, provided the projectivity principle is enforced. The linear coordinates of the nodes are not represented in the figure explicitly, but implicitly, by the relative horizontal position of the nodes towards their governor and sisters.
Unfortunately, it seems that non projective dependencies cannot be avoided in a grammar and thus must be taken into account. Without the projectivity principle, the knowledge of the linear coordinates of every node of a tree is no longer sufficient to define a total order over them. The tree of Figure 4, for example, enriched with the linear coordinates of its nodes, does not define anymore a total order over its nodes if projectivity is not assumed. We have represented in the right part of Figure 4 the same tree with a non-projective dependency marked with a star and the three linear sequences that now correspond to the tree. In the example, knowing that *an* is situated to the left of its governor (*apple*) is not enough to place it in the linear sequence, it can be placed between *John* and *ate* or to the left of *John*.
We will relax the constraints imposed by the projectivity principle on word order and propose another, less restrictive, principle we will call ***pseudo-projectivity*** : a projective dependency is

[2]The projectivity principle, also known as the adjacency principle, has been studied by several researchers. The definition given here is due to D.Hudson

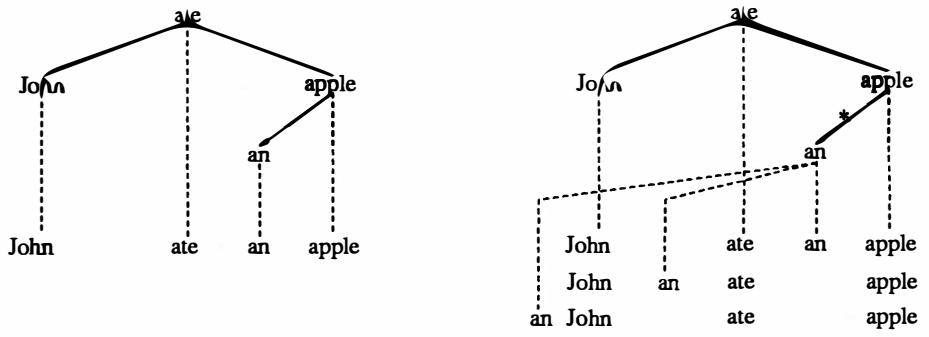



Figure 4: A projective and a non projective tree and their corresponding linear sequences

pseudo-projective but all non projective dependencies are not pseudo-projective.

THE PSEUDO-PROJECTIVITY PRINCIPLE

A dependency is pseudo-projective if its dependent D is not situated, in the linear sequence, between two dependents of a node that is not an ancestor of D.

This principle has been induced from the non projective linguistic examples appearing in [Mel'čuk, 1988] and [Beringer, 1988], all of which satisfy the pseudo-projectivity principle. The three linear sequences of the right part of Figure 4 are pseudo-projective. Contrary to a projective dependency, the dependent D of a pseudo-projective dependency can be separated, in the linear sequence, from its governor by some ancestors of D. We will call the highest of these ancestors the linear governor of D. If the dependency is projective, the linear governor of D is its governor. A node having its governor as linear governor will be said to have a zero non-projectivity level. When the linear governor is the governor's governor, the node will have its non projectivity level equal to one, and so on... We have represented in Figure 5 a tree with a pseudo-projective dependency and the linear sequences corresponding to the tree. To each sequence corresponds a level of non-projectivity of the marked dependency. Triangles hanging from some nodes represent the subtrees rooted by the nodes and the node labels between square brackets represent the linear sequences of the subtrees rooted by the node. By virtue of pseudo-projectivity, the node e cannot be situated in the sequence [a] or [c].

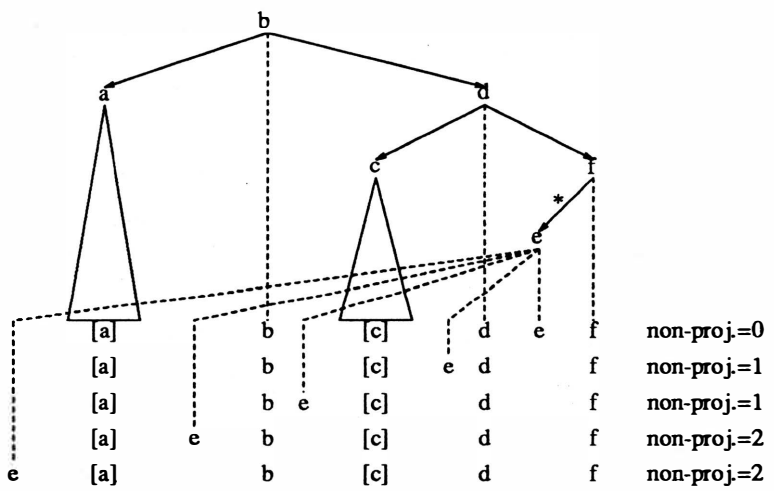



Figure 5: A tree with several non projectivity levels assigned to its non projective dependency

Linear governors will be used to represent the linear order of the nodes of a pseudo-projective tree. We shall define another coordinate system called *pseudo-projective coordinates.*

1. its non projectivity level (or its linear governor).
2. its position towards its linear governor.
3. its position among its linear sisters (dependents of its linear governor).

Enriched with this information, a pseudo-projective tree defines a total order over its nodes. It is interesting to notice that a pseudo-projective tree can be transformed into a projective tree by attaching each node to its linear governor instead of its governor.
This representation of word order will be used to represent word order constraints in elementary trees. But stating such constraints for a node N of an elementary tree might be a problem since they may refer to some nodes (the linear governor and the linear sisters of N) that are outside the domain of locality of the elementary tree. We could decide to extend the domain of locality of the elementary trees and represent the refered node in the domain, as we did for lexical, morphological and semantic restrictions in section 2.1. But such a solution will generate very extended and unnatural elementary trees. Instead we will indicate for some nodes, which of their linear sisters can separate them from their linear governor. More precisely, we shall indicate for the anchor of an elementary tree (when it is not the root of the tree) and for its dependents:

1. their non projectivity level.
2. their relative position towards their linear governor.
3. a list of their linear sisters that can separate them from their linear governor.

The set of these three informations will be refered to as the *positional constraints* of a node.

### 2.3 Combining Trees

The positional constraints we have defined in the preceding section are not checked during tree combining by unification. We define a tree combining operation we call *tree attachment* that is more specific than unification and takes into account positional constraints. Attaching a tree T1 in a tree T2 amounts to unifying the root of T1 with a node of T2. Two more conditions must be fulfilled by the tree resulting from this operation:

1. The black domain[3] of the resulting tree must be connected (i.e during the attachmentoperation, the black domain of T1 must be linked to the black domain of T2 by a dependency). For example, the attachment of Figure 6 fails because it does not lead to a connected black domain.

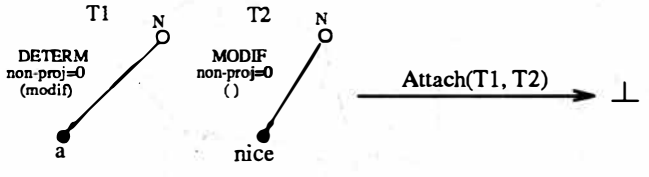



Figure 6: An unsuccessful tree attachment

2. The positional constraints of the dependency linking the black domains of T1 and T2 must be satisfied. Figure 7 shows how the construction of a tree corresponding to the linear sequence "nice a car" fails due to the non satisfaction of node *nice* positional constraints. These constraints prevent the existence of a determiner between a modifier and the modified noun, as indicated by the empty list attached to the dependency MODIF.

[3]The black domain of a tree is the structure composed of all its black nodes and all the dependencies connecting two black nodes.

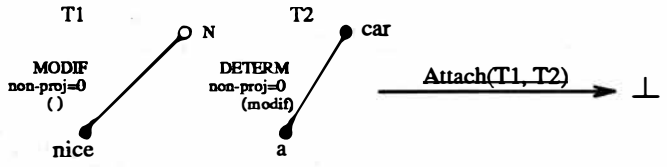



Figure 7: An unsuccessful tree attachment

The attachment of Figure 8 succeeds since the root node of T1 unifies with the node *ate*, the black domain of the resulting tree is connected and positional constraints of the node ***with*** are satisfied: it is situated to the left of *ate* and separated from it by a 1ST.COMP

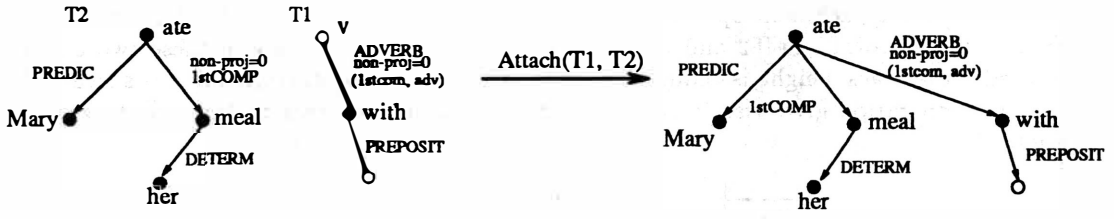



Figure 8: A successful tree attachment

Figure 9 illustrates a successful attachment with a non projective dependency, the node ***than*** is situated directly to the right of node ***salary***, its linear governor.

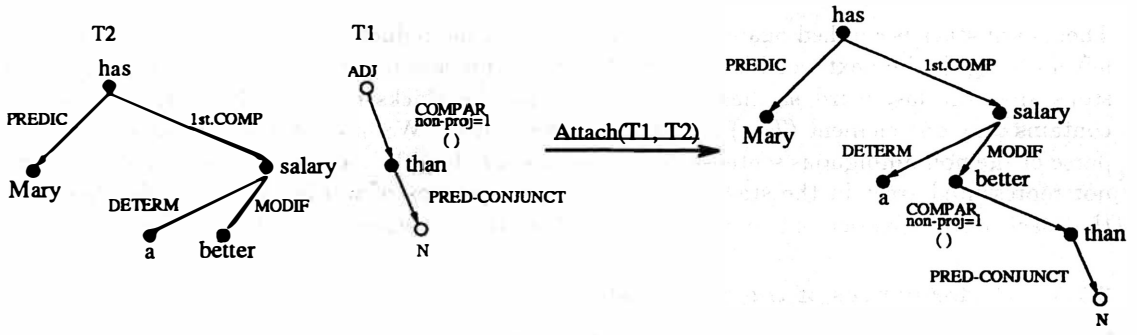



Figure 9: A non projective tree attachment

## 3 Parsing

In our framework, parsing a sentence amounts to combining, by means of attachment, the elementary trees corresponding to each word of the sentence. The main task of the parser is hence to choose the trees that must be combined together and the order in which they will be combined to build a structural description of a sentence. We describe in this section a first algorithm based on a stack which will turn out to be inefficient. The reasons of this inefficiency are studied and a more efficient parser, that makes use of a graph-structured stack, is proposed.

### 3.1 A Stack Parser

The usage of a stack in a parser for dependency structures is not a novelty. It has been advocated in [Fraser, 1989] and [Genthial, 1991], mainly to reduce the number of word pairs that must be checked for dependency. The parser processes a sentence from left to right, for each word $w_i$

read, its elementary trees $T_{i,i}$ [4] are extracted from the lexicon and pushed on the stack. If $w_i$ has $N$ elementary trees, the stack is duplicated $N$ times and each elementary tree is pushed on one stack, as shown in Figure 10.

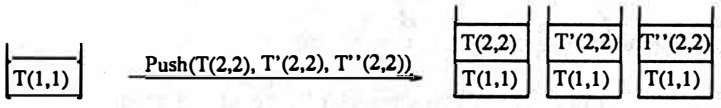



Figure 10: Pushing three elementary trees on a stack

After $w_i$ has been pushed, the $N$ stacks are checked for reduction. The reduction operation tries to combine the top element T1 of the stack with the next one down, T2. The combination is done by the attachment operation. During a stack reduction, two attachments are tested, the attachment of T1 in T2 and the attachment of T2 in T1. If any of these two operations succeeds, the stack height is reduced by one. When the two attachments are successful, the reduction operation gives rise to two trees, in which case, the stack is duplicated, as shown in Figure 11.

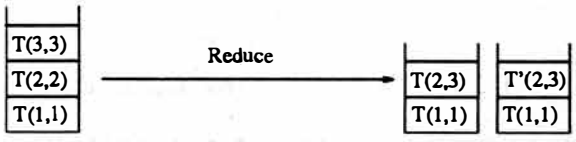



Figure 11: Reducing a stack

Then, each stack is checked again for reduction. When the reduction operation fails, the stack is left unchanged, the next word is read and the whole process reiterated. The parsing algorithm stops after the last word $w_n$ has been pushed and the stacks reduced. If at least one stack contains only one element ($T_{1,n}$) the parsing is successful. We have represented in Figure 12 a parse of the non ambiguous sentence "The man ate a red apple". For sake of simplicity we have not represented trees in the stacks but the linear sequences of words. We have also assumed that each word is associated to a single elementary tree, avoiding stack duplication.

### 3.1.1 Performances of the Stack Parser

The number of stacks during parsing, and hence the number of attachment operations done, grows exponentially with the number of words processed. There are two places where the number of stacks is increased:

1. During the push operation of a word $w_i$, the number of stacks is multiplied by the number of elementary trees associated to $w_i$.

2. During the reduction of a stack, when the two attachment operations are successful, the number of stacks is increased.

This parsing algorithm is not satisfactory since a same attachment can be tested independently several times. Such a configuration is illustrated in Figure 13 where the attachments of $T_{2,2}$ and $T_{3,3}$ are tested twice.
This problem is due to the fact that the same trees are represented several times in separate stacks and, during reduction, each stack is reduced independently, without any consideration of what has been done in the other stacks.

[4] $T_{i,j}$ is a tree corresponding to the segment of the sentence $[w_i, w_j]$ with $i \leq j$. $T_{x,x}$ is an elementary tree corresponding to $w_x$.

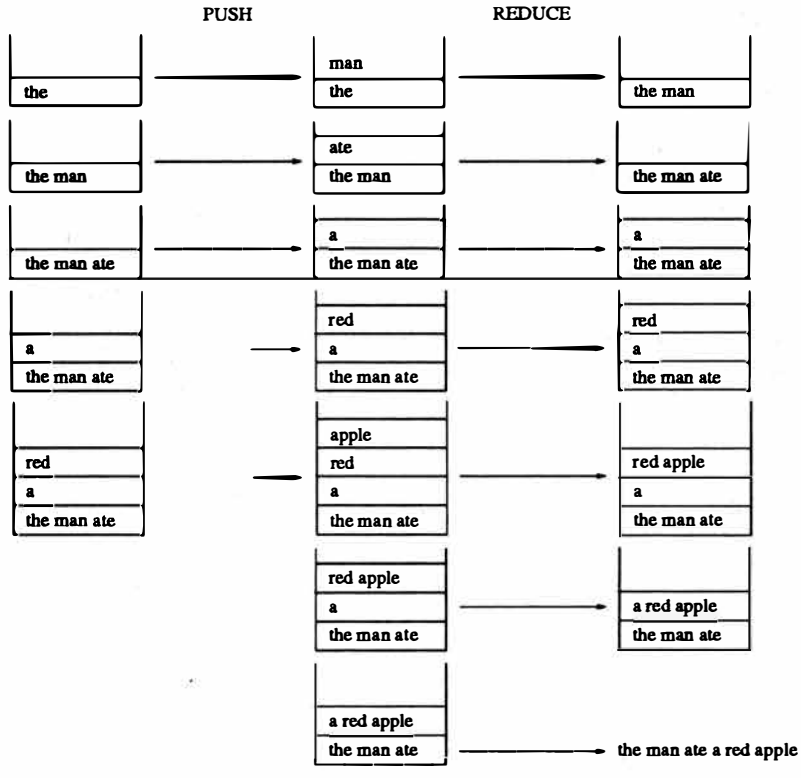
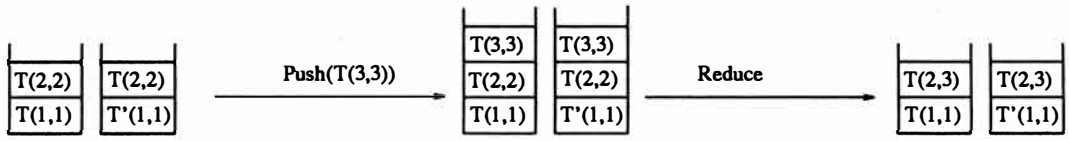



Figure 12: Parsing the sentence "The man ate a red apple"

### 3.2 Using a Graph-structured Stack

In order to prevent the same work from being done several times, we have used the graph-structured stack described in [Tomita, 1988]. A graph-structured stack can be seen as a factoring of several stacks having some elements in common, as shown in Figure 14.
With such a device, any tree is represented once and the attachment of identical pairs of tree is tried once.
The introduction of the graph-structured stack does not change the overall schema of the algorithm. Changes affect only the push and the reduction operation.
Pushing $N$ elementary trees on a graph-structured stack does not duplicate anymore the stack but adds to it $N$ new extremities and creates a link between every former extremity and every new one, as shown in Figure 15.
The reduction operation amounts to trying to combine each extremity of the stack with all its predecessors. When an attachment operation between an extremity of the stack and one of its predecessors succeeds, the number of extremities of the stack can be increased. In the example of Figure 16 the attachment of extremity $T_{j+1,j+1}$ and its predecessor $T_{i,j}$ produces two trees:




Attachment tested during reduction:

$$2 * Attach(T_{2,2}, T_{3,3}) \ ; \ 2 * Attach(T_{3,3}, T_{2,2})$$

Figure 13: Same operations repeated twice

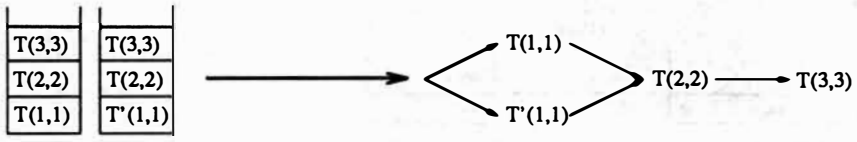



Figure 14: Factoring two stacks into a graph-structured stack

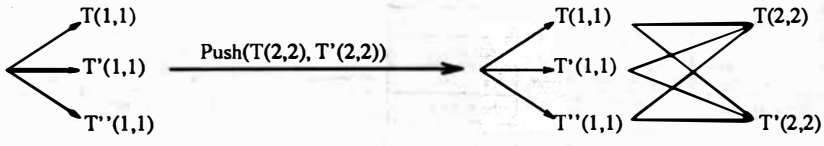



Figure 15: Pushing two elementary trees on a graph-structured stack

$T_{i,j+1}$ and $T'_{i,j+1}$, increasing by one the number of extremities.

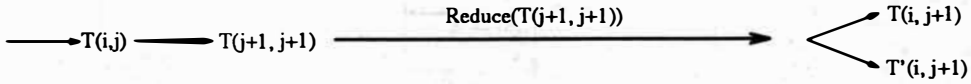
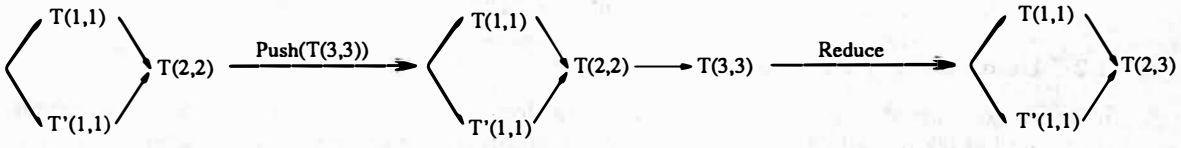



Figure 16: Reducing an extremity of the graph-based stack

The repetition of the same operations that occurred in Figure 13 will not happen with the graph-structured stack, as shown in Figure 17.




Attachment tested during reduction:

$Attach(T_{2,2}, T_{3,3})$ ; $Attach(T_{3,3}, T_{2,2})$

Figure 17: Avoiding repetition of similar operations

The adoption of a graph-structured stack improves the time complexity of the parser. This improvement is mainly due to the fact that the number of extremities of the graph stack does not grow exponentially with the number of words processed, and hence the number of attachment operations carried out. As predicted by Tomita, the time complexity of the parser is polynomial.

## 4 Conclusions and Future Work

We have described in this paper a formalism and a parser for dependency grammars, focussing on the representation of word order in dependency trees and the advantages of using a graph-structured stack for parsing. This work is part of a larger project of sentence paraphrasing. Paraphrasing, in this framework, is based on substitution of elementary trees in syntactic description of sentences. After a sentence has been parsed and a dependency tree built, some of the elementary trees forming the dependency tree of the sentence will be replaced by other elementary trees to which they are linked. The modified syntactic description gives rise to a paraphrase of the inital sentence. Future work will concern the relations between elementary trees and the replacement of an elementary tree by another in a larger tree.

## References


[Beringer, 1988] Beringer, H. (1988). *Traitement Automatique des Ambiguités Structurales du Français utilisant la Théorie Sens-Texte.* PhD thesis, Université de Paris 6.

[Fraser, 1989] Fraser, N. (1989). Parsing and dependency grammar. *UCL Working Papers in Linguistics*, 1:296–319.

[Genthial, 1991] Genthial, D. (1991). *Contribution à la construction d'un système robuste d'analyse du français.* PhD thesis, Université Joseph Fourier.

[Hellwig, 1986] Hellwig, P. (1986). Dependency unification grammar. In *coling86*, Bonn.

[Joshi, 1987] Joshi, A. K. (1987). An introduction to Tree Adjoining Grammars. In Manaster-Ramer, A., editor, *Mathematics of Language*, pages 87–115. John Benjamins, Amsterdam.

[Mel'čuk, 1988] Mel'čuk, I. A. (1988). *Dependency Syntax: Theory and Practice.* State University of New York Press, New York.

[Mel'čuk and Pertsov, 1987] Mel'čuk, I. A. and Pertsov, N. V. (1987). *Surface Syntax of English.* John Benjamins, Amsterdam/Philadelphia.

[Starosta, 1986] Starosta, S. (1986). Lexicase parsing : A lexicon-driven approach to syntactic analysis. In *Proceedings of the 11th International Conference on Computational Linguistics (COLING'86)*, Bonn.

[Tomita, 1988] Tomita, M. (1988). Graph structured stack and natural language parsing. In *26th Meeting of the Association for Computational Linguistics (ACL'88)*, Buffalo, NY.

# Error-tolerant Finite State Recognition

Kemal Oflazer
Department of Computer Engineering and Information Science,
Bilkent University, Ankara, TR-06533, Turkey
ko@cs.bilkent.edu.tr

**Abstract**

Error-tolerant recognition enables the recognition of strings that deviate slightly from any string in the regular set recognized by the underlying finite state recognizer. In the context of natural language processing, it has applications in error-tolerant morphological analysis, and spelling correction. After a description of the concepts and algorithms involved, we give examples from these two applications: In morphological analysis, error-tolerant recognition allows misspelled input word forms to be corrected, and morphologically analyzed concurrently. The algorithm can be applied to the morphological analysis of any language whose morphology is fully captured by a single (and possibly very large) finite state transducer, regardless of the word formation processes (such as agglutination or productive compounding) and morphographemic phenomena involved. We present an application to error-tolerant analysis of agglutinative morphology of Turkish words. In spelling correction, error-tolerant recognition can be used to enumerate correct candidate forms from a given misspelled string within a certain edit distance. It can be applied to any language whose morphology is fully described by a finite state transducer, or with a word list comprising all inflected forms with very large word lists of root and inflected forms (some containing well over 200,000 forms), generating all candidate solutions within 10 to 45 milliseconds (with edit distance 1) on a SparcStation 10/41. For spelling correction in Turkish, error-tolerant recognition operating with a (circular) recognizer of Turkish words (with about 29,000 states and 119,000 transitions) can generate all candidate words in less than 20 milliseconds (with edit distance 1). Spelling correction using a recognizer constructed from a large word German list that simulates compounding, also indicates that the approach is applicable in such cases.

## 1 Introduction

Error-tolerant finite state recognition enables the recognition of strings that deviate *slightly* from any string in the regular set recognized by the underlying finite state recognizer, by perturbations such as deletion, insertion or replacement of symbols. For example, suppose we have a recognizer for the regular set over $\{a, b\}$ described by the regular expression $(aba + bab)^*$, and we would like to recognize inputs which may be corrupted (but not too much) due to a number of reasons: e.g., *abaaaba* may be matched to *abaaba* correcting for a spurious *a*, or *ababba* may be matched to either *abaaba* correcting a *b* to an *a*, or to *ababab* correcting the reversal of the last two symbols. There have been a number of approaches to error-tolerant recognition, e.g., [19, 12]. These are however suited for applications where the pattern or the regular expression is small, and the sequence to be matched is large which is typical in information retrieval applications. For example, Myers and Miller [12], present an $O(MN)$ algorithm with $M$ being the length of the string and $N$, being the length of the regular expression. In the applications we are concerned with, the recognizers (and the corresponding regular expressions) are very large. The approach presented in this paper uses the finite state recognizer that recognizes the regular set, but relies on a very efficiently controlled recognition algorithm based on depth-first search of the state graph of the recognizer.

## 2 Error-tolerant Finite State Recognition

We can informally define error-tolerant recognition with a finite state recognizer as the recognition of all strings in the regular set (accepted by the recognizer), and additional strings which can be obtained from any string in the set *by a small number of unit editing operations.* Error-tolerant recognition requires an error metric for measuring how much two strings deviate from each other. The *edit distance metric* between two strings measures the minimum number of unit editing operations of *insertion, deletion, replacement of a symbol, and transposition of adjacent symbols* [3], that are necessary to convert one string into another. Let $X = x_1, x_2, \ldots, x_m$, and $Y = y_1, y_2, \ldots, y_n$ denote strings of length $m and n respectively of m$ symbols from an alphabet $A$. $X[j]$ ($Y[j]$) denotes the initial substring of $X$ ($Y$) up to and including the $j^{th}$ symbol. Given $X$ and $Y$, the edit distance $ed(X[m], Y[n])$ computed according to the recurrence below, gives the minimum number of unit editing operations to convert one string to the other [4].

$$
\begin{array}{rcll}
ed(X[i+1], Y[j+1]) & = & ed(X[i], Y[j]) & \text{if } x_{i+1} = y_{j+1} \\
 & & & \text{(last characters are same)} \\
 & = & 1 + min\{ed(X[i-1], Y[j-1]), & \text{if both } x_i = y_{j+1} \\
 & & \quad ed(X[i+1], Y[j]), & \text{and } x_{i+1} = y_j \\
 & & \quad ed(X[i], Y[j+1])\} & \\
 & & & \text{(last two characters are} \\
 & & & \text{transposed)} \\
 & = & 1 + min\{ed(X[i], Y[j]), & \text{otherwise} \\
 & & \quad ed(X[i+1], Y[j]), & \\
 & & \quad ed(X[i], Y[j+1])\} & \\
ed(X[0], Y[j]) & = & j & 0 \leq j \leq n \\
ed(X[i], Y[0]) & = & i & 0 \leq i \leq m \\
 & & & \\
ed(X[-1], Y[j]) & = & ed(X[i], Y[-1]) = \max(m, n) & \text{(Boundary definitions)}
\end{array}
$$

For example $ed(recoginze, recognize) = 1$, since transposing *i* and *n* in the former string would give the latter. Similarly $ed(sailn, failing) = 3$ as in the former string, one could change the initial *f* to *s*, insert an *i* before the *n*, and insert a *g* at the end to obtain the latter.

Given a (deterministic) finite state recognizer, $R$,[1] let $L \subseteq A^*$ be the regular language accepted by $R$, and let $t > 0$, be the edit distance error threshold. We define a string $X[m] \notin L$ to be recognized by $R$ with an error at most $t$, if the set $C = \{Y[n] \mid Y[n] \in L \text{ and } ed(X[m], Y[n]) \leq t\}$ is not empty.

Any finite state recognizer can also be viewed as a directed graph with arcs are labeled with symbols in $A$.[2] Standard finite state recognition corresponds to traversing a path (possibly involving cycles) in the graph of the recognizer, starting from the start node, to one of the final nodes, so that the concatenation of the labels on the arcs along this path matches the input string. For error-tolerant recognition one needs to find *all paths from the start node to one of the final nodes, so that when the labels on the arcs along a path are concatenated, the resulting string is within a given edit distance threshold t, of the (erroneous) input string.* With $t > 0$, the recognition procedure becomes a search on this graph.

Searching the graph of the recognizer has to be very fast if error-tolerant recognition is to be of any practical use. This means that paths that can lead to no solutions have to be pruned so that the search can be limited to a very small percentage of the search space. Thus we need to make sure that any candidate string that is generated as the search is being performed, does not deviate from certain initial

[1] R is described by a 5-tuple $R = (Q, A, \delta, q_0, F)$ with $Q$ denoting the set of states, $A$ denoting the input alphabet, $\delta : Q \times A \rightarrow Q$, denoting the state transition function, $q_0 \in Q$ denoting the initial state, and $F \subseteq Q$ denoting the final states.The approach presented here not limited to deterministic machines, though.

[2] We may use state and node, and transition and arc, interchangeably.

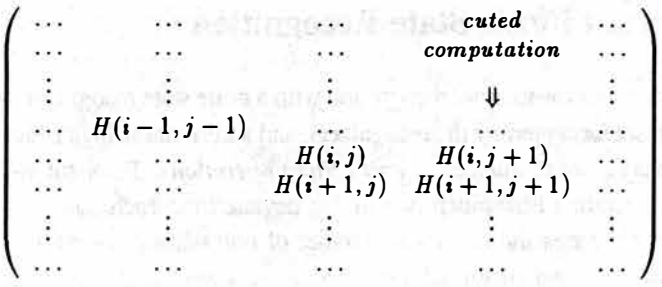



Figure 2: Computation of the elements of the $H$ matrix.

matrix $H$ with element $H(i,j) = ed(X[i], Y[j])$ [4]. We can note that the computation of the element $H(i+1, j+1)$ recursively depends on only $H(i,j), H(i, j+1), H(i+1, j)$ and $H(i-1, j-1)$, from the earlier definition of the edit distance. During the depth first search of the state graph of the recognizer, entries in column $n$ of the matrix $H$ have to be (re)computed, only when the candidate string is of length $n$. During backtracking, the entries for the last column are discarded, but the entries in prior columns are still valid. Thus all entries required by $H(i+1, j+1)$, except $H(i, j+1)$, are already available in the matrix in columns $i-1$ and $i$. The computation of $cuted(X[m], Y[n])$ involves a loop in which the minimum is computed. This loop (indexing along column $j+1$), computes $H(i, j+1)$ before it is needed for the computation of $H(i+1, j+1)$. (See Figure 2.)

We now present in Figure 3 a simple example for this search algorithm for a simple finite state recognizer for the regular expression $(aba + bab)^*$, and the search graph for the input string $ababa$. The

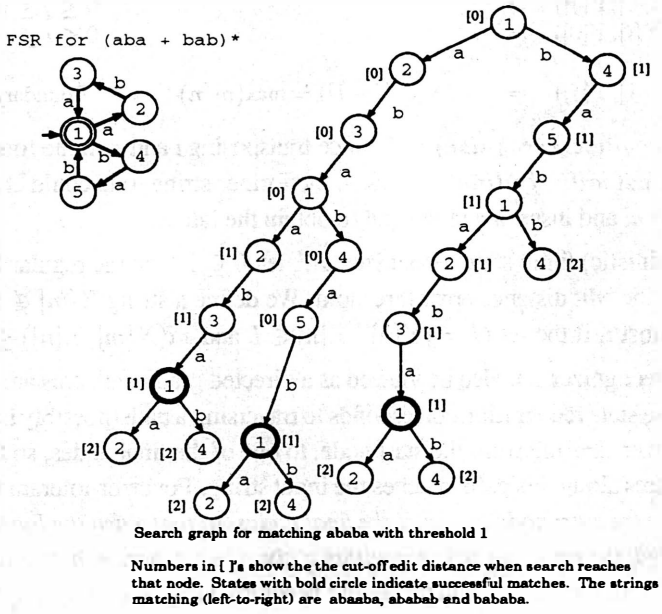



Figure 3: Recognizer for $(aba + bab)^*$ and search graph for $ababa$.

thick circles from left to right indicate the nodes at which we have the matching strings $abaaba$, $ababab$ and $bababa$, respectively. Prior visits to the final state 1, violate the final edit distance constraint. (Note that the visit order of siblings depend on how one orders the outgoing arcs from a state.)

```
/*push empty candidate, and start node to start search */
push((ε, q0))
while stack not empty
    begin
       pop((Y', qi)) /* pop partial surface string Y'
                        and the node */
       for all qj and a such that δ(qi, a) = qj
          begin /* extend the candidate string */
             Y = concat(Y', a) /* n is the current length of Y */
          /* check if Y has deviated too much, if not push */
             if cuted(X[m], Y[n]) ≤ t then push((Y, qj))
          /* also see if we are at a final state */
             if ed(X[m], Y[n]) ≤ t and qj ∈ F then output Y
          end
    end
```

Figure 1: Algorithm for error-tolerant recognition

substrings of the erroneous string by more than the allowed threshold. To detect such cases, we use the notion of a *cut-off edit distance*. The cut-off edit distance measures the minimum edit distance between an initial substring of the incorrect input string, and the (possibly partial) candidate correct string. Let $Y$ be a partial candidate string whose length is $n$, and let $X$ be the incorrect string of length $m$. Let $l = \min(1, n-t)$ and $u = \max(m, n+t)$. The cut-off edit distance $cuted(X[m], Y[n])$ is defined as

$$cuted(X[m], Y[n]) = \min_{l \le i \le u} ed(X[i], Y[n]).$$

For example, with $t = 2$:

$$cuted(\text{reprter}, \text{repo}) = \{\min\{ed(\text{re}, \text{repo}) = 2, ed(\text{rep}, \text{repo}) = 1, ed(\text{repr}, \text{repo}) = 1, ed(\text{reprt}, \text{repo}) = 2, ed(\text{reprte}, \text{repo}) = 3\} = 1.$$

Note that except at the boundaries, the initial substrings of the incorrect string $X$ considered are of length $n-t$ to length $n+t$. Any initial substring of $X$ shorter than $n-t$ needs more than $t$ insertions, and any initial substring of $X$ longer than $n+t$ requires more than $t$ deletions, to at least equal $Y$ in length, violating the edit distance constraint.

Given an incorrect string $X$, a partial candidate string $Y$ is generated by successively concatenating relevant labels along the arcs as transitions are made, starting with the start state. Whenever we extend $Y$, we check if the cut-off edit distance of $X$ and the partial $Y$, is within the bound specified by the threshold $t$. If the cut-off edit distance goes beyond the threshold, the last transition is backed off to the source node (in parallel with the shortening of $Y$) and some other transition is tried. Backtracking is recursively applied when the search can not be continued from that state. If, during the construction of $Y$, a final state is reached without violating the cutoff edit distance constraint, and $ed(X[m], Y[n]) \le t$ at that point, then $Y$ is a valid correct form of the incorrect input string.[3]

Denoting the states by subscripted $q$'s ($q_0$ being the initial state) and the symbols in the alphabet (and labels on the directed edges) by $a$, we present the algorithm for generating all $Y$'s by a (slightly modified) depth-first probing of the graph in Figure 1. The crucial point in this algorithm is that the cut-off edit distance computation can be performed very efficiently by maintaining a *global* $m$ by $n$

[3]Note that we have to do this check since we may come to other irrelevant final states during the search.

# 3 Application to Error-tolerant Morphological Analysis

Error-tolerant finite state recognition can be applied to morphological analysis, in which, instead of rejecting a given misspelled form, the analyzer attempts to apply the morphological analysis to forms that are within a certain (configurable) edit distance of the incorrect form. Two-level transducers [10] provide a suitable model for the application of error-tolerant recognition. Such transducers capture all morphotactic and morphographemic phenomena, and alternations in the language in a uniform manner. They can be abstracted as finite state transducers over an alphabet of lexical and surface symbol pairs `l:s`, where either `l` or `s` (but not both) may be the null symbol `0`. It is possible to apply error-tolerant recognition to languages whose word formations employ productive compounding and/or agglutination, and in fact to any language whose morphology is described completely as one (very large) finite state transducer. Full scale descriptions using this approach already exist for a number of languages like English, French, German, Turkish, Korean [8].

Application of error-tolerant recognition to morphological analysis proceeds as described earlier. After a successful match with a surface symbol, the corresponding lexical symbol is appended to the output gloss string. During backtracking the candidate surface string and the gloss string are again shortened in tandem. The basic algorithm for this case is given in Figure 4.[4] The actual algorithm is a slightly optimized version of this where transitions with null surface symbols are treated as special during forward and backtracking traversals to avoid unnecessary computations of the cut-off edit distance.

```
/*push empty candidate string, and start node
to start search on to the stack */
push((ε, ε, q0))
while stack not empty
    begin
       pop((surface', lexical', qi)) /* pop partial strings
             and the node from the stack */
       for all qj and l: s such that δ(qi, l: s) = qj
          begin /* extend the candidate string */
             surface = concat(surface', s)
             if cuted(X[m], surface[n]) ≤ t then
                begin
                     lexical = concat(lexical', l)
                     push((surface, lexical, qj))
                     if ed(X[m], surface[n]) ≤ t and qj ∈ F then
                       output lexical
                end
          end
    end
```

Figure 4: Algorithm for error-tolerant morphological analysis.

We can demonstrate error-tolerant morphological analysis with a two–level transducer for the analysis of Turkish morphology. Agglutinative languages such as Turkish, Hungarian or Finnish, differ from languages like English in the way lexical forms are generated. Words are formed by productive affixations of derivational and inflectional affixes to roots or stems. Furthermore, roots and affixes may undergo changes due to various phonetic interactions. A typical nominal or a verbal root gives rise to thousands of valid forms which never appear in the dictionary. For instance, we can give the following (rather exaggerated) adverb example from Turkish:

*uygarlaştıramayabileceklerimizdenmişsinizcesine*

[4] Note that transitions are now labeled with $l : s$ pairs.

whose root is the adjective *uygar* (civilized).[5] The morpheme breakdown (with morphological glosses underneath) is:[6]

| | | | | | |
|---|---|---|---|---|---|
| *uygar* | *+laş* | *+tır* | *+ama* | *+yabil* | *+ecek* |
| civilized | +AtoV | +CAUS | +NEG | +POT | +VtoA(AtoN) |
| *+ler* | *+imiz* | *+den* | *+miş* | *+siniz* | *+cesine* |
| +3PL | +POSS-1PL | +ABL(+NtoV) | +PAST | +2PL | +VtoAdv |

The portion of the word following the root consists of 11 morphemes each of which either adds further syntactic or semantic information to, or changes the part-of-speech, of the part preceding it. Though most words one uses in Turkish are considerably shorter than this, the example serves to point out some of the fundamental difference of the nature of the word structures in Turkish and other agglutinative languages, from those of languages like English.

Our morphological analyzer for Turkish is based on a lexicon of about 28,000 root words and is a re-implementation of PC-KIMMO version of the same description [1, 13], using Xerox two-level transducer technology [9]. This description of Turkish morphology has 31 two-level rules that implement the morphographemic phenomena such as vowel harmony and consonant changes across morpheme boundaries etc., and about 150 additional rules, again based on the two-level formalism, that fine tune the morphotactics by enforcing sequential and long-distance feature sequencing and co-occurrence constraints, in addition to constraints imposed by standard alternation linkage among various lexicons to implement the paradigms. Turkish morphotactics is circular due to the relativization suffix in the nominal paradigm, and multiple causative suffixes in the verb paradigm. There is also considerable linkage between nominal and verbal morphotactics due to productive derivational suffixes. The minimized finite state transducer constructed by composing the transducers for root lexicons, morphographemic rules and morphotactic constraints, has 32,897 states and 106,047 transitions, with an average fan out of about 3.22 transitions per state (including transitions with null surface symbols). It analyzes a given Turkish lexical form into a sequence of *feature-value tuples* (instead of the more conventional sequence of morpheme glosses) that are used in a number of natural language applications.[7]

This transducer has been used as input to an analyzer implementing the error-tolerant recognition algorithm in Figure 4. The analyzer first attempts to parse the input with $t = 0$, and if it fails, relaxes $t$ up to 2, if it can not find any parse with a smaller $t$, and can process about 150 (correct) forms a second on a Sparcstation 10/41.[8,9] Below, we provide a transcript of a run:[10]

```
ENTER WORD > getirn
Threshold 0 ...  1 ...


getiri   =>  ((CAT NOUN)(ROOT getiri)(AGR 3SG)(POSS NONE)(CASE NOM))
             (return (on investment))
getirin  =>  ((CAT NOUN)(ROOT getiri)(AGR 3SG)(POSS 2SG)(CASE NOM))
             (your return)
getiren  =>  ((CAT VERB)(ROOT getir)(SENSE POS)(CONV ADJ YAN))
             (s/he who brings)
getirin  =>  ((CAT VERB)(ROOT getir)(SENSE POS)(MOOD IMP)(AGR 2PL))
             (bring!  (imp 2pl))
getir    =>  ((CAT VERB)(ROOT getir)(SENSE POS)(MOOD IMP)(AGR 2SG))
             (bring!  (imp 2sg))
getire   =>  ((CAT VERB)(ROOT getir)(SENSE POS)(MOOD OPT)(AGR 3SG))
             (let him bring)
```

[5] This is a manner adverb meaning roughly "(behaving) as if you were one of those whom we might not be able to civilize."

[6] Glosses in parentheses indicate derivations not explicitly indicated by a morpheme.

[7] The Xerox software allows the resulting finite state transducer to be exported in a tabular form which can be imported to other applications.

[8] No attempt was made to compress the finite state recognizer.

[9] The Xerox *infl* program working on the proprietary compressed representation of the same transducer can process about 1000 forms/sec on the same platform.

[10] The outputs have been slightly edited for formatting. The feature names denote the usual morphosyntactic features. CONV denotes derivations to the category indicated by the second token with a suffix or derivation type denoted by the third token, if any. We have also added English glosses that most closely approximate the semantics of the word.

```
getirt  =>  ((CAT VERB)(ROOT getir)(VOICE CAUS)(SENSE POS)(MOOD IMP)(AGR 2SG))
            (have him/her bring!  (imp 3sg))


ENTER WORD > akıllınnikiler
Threshold 0 ...  1 ...  2 ...


akıllınınkiler => ((CAT NOUN)(ROOT akıl)(CONV ADJ LI)
                      (CONV NOUN)(AGR 3SG) (POSS NONE)(CASE GEN)
                      (CONV PRONOUN REL)(AGR 3PL)(POSS NONE)(CASE NOM))
                  ((things) of the smart (one))
akıllınınkiler => ((CAT NOUN)(ROOT akıl)(CONV ADJ LI)
                      (CONV NOUN)(AGR 3SG)(POSS 2SG)(CASE GEN)
                      (CONV PRONOUN REL)(AGR 3PL)(POSS NONE)(CASE NOM))
                  ((things) of your smart (one))
akıllındakiler => ((CAT NOUN)(ROOT akıl)(CONV ADJ LI)
                      (CONV NOUN)(AGR 3SG)(POSS 2SG)(CASE LOC)
                      (CONV ADJ REL)(CONV NOUN)(AGR 3PL)(POSS NONE)(CASE NOM))
                  ((things) at your smart (one))


ENTER WORD > eviminkinn
Threshold 0 ...  1 ...


eviminkini =>     ((CAT NOUN)(ROOT ev)(AGR 3SG)(POSS 1SG)(CASE GEN)
                      (CONV PRONOUN REL)(AGR 3SG)(POSS NONE)(CASE ACC))
                  ((thing) of my house (accusative))
eviminkine =>     ((CAT NOUN)(ROOT ev)(AGR 3SG)(POSS 1SG)(CASE GEN)
                      (CONV PRONOUN REL)(AGR 3SG)(POSS NONE)(CASE DAT))
                  (to the (thing) of my house)
eviminkinin =>    ((CAT NOUN)(ROOT ev)(AGR 3SG)(POSS 1SG)(CASE GEN)
                      (CONV PRONOUN REL)(AGR 3SG)(POSS NONE)(CASE GEN))
                  (of the (thing) of my house)


ENTER WORD > uygarlaştıramadıklarmıızdanmışsınızcasına
Threshold 0 ...  1 ...


uygarlaştıramadıklarımızdanmışsınızcasına =>
           ((CAT ADJ)(ROOT uygar)(CONV VERB LAS)(VOICE CAUS)(SENSE NEG-C)
                 (CONV ADJ DIK)(AGR 3PL)(POSS 1PL)(CASE ABL)
                 (CONV VERB)(TENSE NARR-PAST)(AGR 2PL)
                 (CONV ADVERB CASINA)(TYPE MANNER))
           (behaving) as if you were one of those whom we could not civilize


ENTER WORD > okatulna
Threshold 0 ...  1 ...  2 ...


okutulma =>       ((CAT VERB)(ROOT oku)(VOICE CAUS)(VOICE PASS)(SENSE NEG)
                     (MOOD IMP)(AGR 2SG))
                  (don't be forced to read!  (imp 2sg))
okutulma =>       ((CAT VERB)(ROOT oku)(VOICE CAUS)(VOICE PASS)(SENSE POS)
                     (CONV NOUN MA)(TYPE INFINITIVE)
                     (AGR 3SG)(POSS NONE)(CASE NOM))
                  ((the act of) being forced to read))
okutulan =>       ((CAT VERB)(ROOT oku)(VOICE CAUS)(VOICE PASS)(SENSE POS)
                     (CONV ADJ YAN))
                  ((thing) that is caused to be read)
okutulana =>      ((CAT VERB)(ROOT oku)(VOICE CAUS)(VOICE PASS)(SENSE POS)
                     (CONV ADJ YAN)(CONV NOUN)(AGR 3SG)(POSS NONE)(CASE DAT))
                  (to the (thing) that is caused to be read)
okutulsa =>       ((CAT VERB)(ROOT oku)(VOICE CAUS)(VOICE PASS)(SENSE POS)
                     (MOOD COND)(AGR 3SG))
                  (if (it) were caused to be read)
okutula =>        (CAT VERB)(ROOT oku)(VOICE CAUS)(VOICE PASS)(SENSE POS)
                     (MOOD OPT)(AGR 3SG))
                  (let it be caused to be read)
```

In an application context, the candidates that are generated by such a morphological analyzer can be disambiguated or filtered to a certain extent by constraint-based tagging techniques, e.g., [16, 18] that take into account syntactic context for morphological disambiguation.

# 4 Applications to Spelling Correction

Spelling correction is an important application for error-tolerant recognition. There has been substantial amount of work on spelling correction (see the excellent review by Kukich [11]). Most methods essentially enumerate plausible candidates which resemble the incorrect word, and use additional heuristics to rank the results.[11] However, most techniques assume a word list of all words in the language. These approaches are suitable for languages like English for which it is possible to enumerate such a list. They are not directly suitable or applicable to languages like German, that have very productive compounding, or to agglutinative languages like Finnish, Hungarian or Turkish, in which the concept of a word is much larger than what is normally found in a word list. For example, Finnish nouns have about 2000 distinct forms while Finnish verbs have about 12,000 forms [6, pages 59–60]. The case in Turkish is also similar where, for instance nouns may have about 170 different forms, not counting the forms for adverbs, verbs, adjectives, or other nominal forms, generated (sometimes circularly) by derivational suffixes. Hankamer [7] gives much higher figures (in the millions) for Turkish, presumably by taking into account derivations.

There have been some recent approaches to spelling correction using morphological analysis techniques. Veronis [17] presents a method for handling quite complex combinations of typographical and phonographic errors, the latter being the kind of errors usually made by language learners using computer-aided instruction. This method takes into account phonetic similarity in addition to standard types of errors. Aduriz *et al.* [5] have used a two-level morphology approach to spelling correction in Basque which uses two-level rules to describe common insertion and deletion errors, in addition to the two-level rules for the morphographemic component. Oflazer and Güzey [15] have used a two-level morphology approach to spelling correction in agglutinative languages, which has used a coarser morpheme-based morphotactic description instead of the finer lexical/surface symbol approach presented here. More recently, Bowden and Kiraz [2] have used a multi-tape morphological analysis technique for spelling correction in Semitic languages which, in addition to the insertion, deletion, substitution and transposition errors, allows for various language specific errors.

For languages like English where all inflected forms can be included in a word list, the word list can be used to construct a finite state recognizer structured as a standard letter tree recognizer (which has an acyclic graph) to which error-tolerant recognition can be applied. Furthermore, just as in morphological analysis, transducers for morphological analysis can obviously be used for spelling correction, so one algorithm can be applied to any language whose morphology is described using such transducers. We demonstrate the application of error-tolerant recognition to candidate generation in spelling correction by constructing finite state recognizers in the form of letter tree from large word lists that contain root and inflected forms of words for 10 languages, obtained from a number of resources on the Internet. Table 1 gives statistics about the word lists used. The Dutch, French, German, English (two different lists), and Italian, Norwegian, Swedish, Danish and Spanish word lists contained some or all inflected forms in addition to the basic root forms. The Finnish word list contained unique word forms compiled from a corpus, although the language is agglutinative.

For edit distance thresholds, 1, 2, and 3, we selected randomly, 1000 words from each word list and perturbed them by random insertions, deletions, replacements and transpositions so that each misspelled word had the respective edit distance from the correct form. Kukich [11], citing a number of studies, reports that typically 80% of the misspelled words contain a single error of one of the unit operations, though there are specific applications where the percentage of such errors are lower. Our earlier study of an error model developed for spelling correction in Turkish also indicated similar results [15].

[11] Ranking is dependent on the language, the application, and the error model. It is an important component of the spelling correction problem, but is not addressed in this paper.

Table 1: Statistics about the language word lists used

| Language | Words | Arcs | Average Word Length | Maximum Word Length | Average Fan-out |
|---|---|---|---|---|---|
| Finnish | 276,448 | 968,171 | 12.01 | 49 | 1.31 |
| English-1 | 213,557 | 741,835 | 10.93 | 25 | 1.33 |
| Dutch | 189,249 | 501,822 | 11.29 | 33 | 1.27 |
| German | 174,573 | 561,533 | 12.95 | 36 | 1.27 |
| French | 138,257 | 286,583 | 9.52 | 26 | 1.50 |
| English-2 | 104,216 | 265,194 | 10.13 | 29 | 1.40 |
| Spanish | 86,061 | 257,704 | 9.88 | 23 | 1.40 |
| Norwegian | 61,843 | 156,548 | 9.52 | 28 | 1.32 |
| Italian | 61,183 | 115,282 | 9.36 | 19 | 1.84 |
| Danish | 25,485 | 81,766 | 10.18 | 29 | 1.27 |
| Swedish | 23,688 | 67,619 | 8.48 | 29 | 1.36 |

Table 2: Correction Statistics for Threshold 1

| Language | Average Misspelled Word Length | Average Correction Time (msec) | Avg. Time to First Solution (msec) | Average Number of Solutions Found | Average % of Space Searched |
|---|---|---|---|---|---|
| Finnish | 11.08 | 45.45 | 25.02 | 1.72 | 0.21 |
| English-1 | 9.98 | 26.59 | 12.49 | 1.48 | 0.19 |
| Dutch | 10.23 | 20.65 | 9.54 | 1.65 | 0.20 |
| German | 11.95 | 27.09 | 14.71 | 1.48 | 0.20 |
| French | 10.04 | 15.16 | 6.09 | 1.70 | 0.28 |
| English-2 | 9.26 | 17.13 | 7.51 | 1.77 | 0.35 |
| Spanish | 8.98 | 18.26 | 7.91 | 1.63 | 0.37 |
| Norwegian | 8.44 | 16.44 | 6.86 | 2.52 | 0.62 |
| Italian | 8.43 | 9.74 | 4.30 | 1.78 | 0.46 |
| Danish | 8.78 | 14.21 | 1.98 | 2.25 | 1.00 |
| Swedish | 7.57 | 16.78 | 8.87 | 2.83 | 1.57 |

Table 2 present the results from correcting these misspelled word lists for edit distance threshold 1. The runs were performed on a Sparcstation 10/41. The second column in these tables gives the average length of the misspelled string in the input list. The third column gives the time in milliseconds to generate *all* solutions, while the fourth column gives the time to find the first solution. The fifth column gives the average number of solutions generated from the given misspelled strings with the given edit distance. Finally, the last column gives the percentage of the search space (that is, the ratio of forward traversed arcs to the total number of arcs) that is searched when generating all the solutions. Due to space limitations, we are not able to give full results for thresholds 2, and 3, but for $t = 2$, the average correction time ranged from 80 msecs to 320 msecs, with about 1.3% to 7.5% of the search space being searched. For $t = 3$, which is very unlikely in real applications, the average correction time ranged from 200 msecs to 1200 msecs, with about 3% to 17% of the search space being searched. (See Oflazer [14] for details on these results.)

Table 3: Spelling correction results for the Turkish finite state recognizer.

| Threshold t | Average Misspelled Word Length | Average Correction Time (msec) | Avg. Time to First Solution (msec) | Average Number of Solutions Found | Average % of Space Searched |
|---|---|---|---|---|---|
| 1 | 8.63 | 17.90 | 7.41 | 4.92 | 1.23 |
| 2 | 8.59 | 164.81 | 57.87 | 55.12 | 11.12 |
| 3 | 8.57 | 907.02 | 63.59 | 442.17 | 60.00 |

## 4.1 Spelling Correction for Agglutinative Word Forms

The transducer for Turkish developed for morphological analysis, using the Xerox software, was also used for spelling correction. However, the original transducer had to be simplified into a recognizer for two reasons. For morphological analysis, the concurrent generation of the lexical gloss string requires that occasional transitions with an empty surface symbol be taken, to generate the gloss properly. Secondly, a given surface form can morphologically be interpreted in many ways which is important in morphological processing. In spelling correction, the presentation of only one of such surface forms is sufficient. To remove all empty transitions and analyses with same surface forms from the Turkish transducer, a recognizer recognizing only the surface forms was extracted by using the Xerox tool *ifsm*. The resulting recognizer had 28,825 states and 118,352 transitions labeled with just surface symbols. The average fan-out of the states in this recognizer was about 4. This transducer was then used to perform spelling correction experiments in Turkish.

In the first set of experiments three word lists of 1000 words each were generated from a Turkish corpus, and words were perturbed as described before, for error thresholds of 1, 2, and 3 respectively. The results for correcting these words are presented in Table 3. It should be noted that the percentage of search space searched may not be very meaningful in this case since the same transitions may be taken in the forward direction, more than once.

On a separate experiment which would simulate a real correction application, about 3000 misspelled Turkish words again compiled from a corpus, were processed by successively relaxing the error threshold starting with $t = 1$. Of these set of words, 79.6% had an edit distance of 1 from the intended correct form, while 15.0% had edit distance 2, and 5.4% had edit distance 3 or more.[12] The average length of the incorrect strings was 9.63 characters. The average correction time was 77.43 milliseconds (with 24.75 milliseconds for the first solution). The average number of candidates offered per correction was 4.29, with an average of 3.62% of the search space being traversed, indicating that this is a very viable approach for real applications. For comparison, the same recognizer running as a spell checker ($t = 0$) can process correct forms at a rate of about 500 words/sec.

## 4.2 Spelling correction with compounding

Using the 174,249 word German word list used in the experiments above, a finite state recognizer recognizing $G^+$, where $G$ is a German word in the list, was constructed. The intention was to simulate the compounding process though clearly the finite state recognizer constructed is not a correct compounding recognizer for German. One hundred (pseudo-) German compounds were formed by concatenating words form the original word list, and then introducing errors randomly. The results with thresholds

[12] In almost all of these, the misspelling were due to rendering of non-ASCII Turkish characters with the nearest ASCII neighbors, e.g., *ü* typed as *u*.

Table 4: Spelling correction results for (simulated) German compounding.

| Threshold t | Average Misspelled Word Length | Average Correction Time (msec) | Avg. Time to First Solution (msec) | Average Number of Solutions Found | Average % of Space Searched |
|---|---|---|---|---|---|
| 1 | 32.96 | 332.58 | 84.50 | 1.77 | 0.30 |
| 2 | 32.03 | 1138.63 | 308.52 | 5.52 | 2.45 |
| 3 | 31.53 | 4757.70 | 966.27 | 9.71 | 10.17 |

1,2, and 3 are presented in Table 4. It should however be stressed again that the numbers for search space percentages reflect the percentage with respect to the number of static recognizer links. One can, however, get a reasonable approximation to the actual percentage of the search space by observing that by the average length of the words in the test file is about three times the average word length in the original German word list. Thus on the average about three traversals of the whole finite state recognizer are made and thus the search percentages above should be divided by three.

# 5 Conclusions

This paper has presented an algorithm for error-tolerant finite state recognition which enables a finite state recognizer to recognize strings that deviate mildly from some string in the underlying regular set, along with results of its application to error-tolerant morphological analysis, and candidate generation in spelling correction. The approach is very fast and applicable to any language with a given a list of root and inflected forms, or with a finite state transducer recognizing or analyzing its word forms. It differs from previous error-tolerant finite state recognition algorithms in that it uses a given finite state machine, and is more suitable for applications where the number of patterns (or the finite state machine) is very large and the string to be matched is small.

# 6 Acknowledgments

This research was supported in part by a NATO Science for Stability Project Grant TU-LANGUAGE. I would like to thank Xerox Advanced Document Systems, and Lauri Karttunen of Xerox Parc and of Rank Xerox Research Centre (Grenoble) for providing us with the two-level transducer development software. Kemal Ülkü and Kurtuluş Yorulmaz of Bilkent University implemented some of the algorithms.

# References

[1] Evan L. Antworth. *PC-KIMMO: A two-level processor for Morphological Analysis.* Summer Institute of Linguistics, Dallas, Texas, 1990.

[2] Tanya Bowden and George A. Kiraz. A morphographemic model for error correction in nonconcatenative strings. In *Proceedings of the* $33^{rd}$ *Annual Meeting of the Association for Computational Linguistics*, pages 24–30, Boston, MA, 1995.

[3] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the Association for Computing Machinery*, 7(3):171–176, 1964.

[4] M .W. Du and S. C. Chang. A model and a fast algorithm for multiple errors spelling correction. *Acta Informatica*, 29:281–302, 1992.

[5] I. Aduriz *et al*. A morphological analysis based method for spelling correction. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, Utrecht, The Netherlands, April 1993.

[6] Gerald Gazdar and Chris Mellish. *Natural Language Processing in PROLOG, An Introduction to Computational Linguistics*. Addison-Wesley Publishing Company, 1989.

[7] Jorge Hankamer. Morphological parsing and the lexicon. In W. Marslen-Wilson, editor, *Lexical Representation and Process*. MIT Press, 1989.

[8] Lauri Karttunen. Constructing lexical transducers. In *Proceedings of the* $16^{th}$ *International Conference on Computational Linguistics*, volume 1, pages 406–411, Kyoto, Japan, 1994. International Committee on Computational Linguistics.

[9] Lauri Karttunen and Kenneth. R. Beesley. Two-level rule compiler. Technical Report, XEROX Palo Alto Research Center, 1992.

[10] Lauri Karttunen, Ronald M. Kaplan, and Annie Zaenen. Two-level morphology with composition. In *Proceedings of the* $15^{th}$ *International Conference on Computational Linguistics*, volume 1, pages 141–148, Nantes, France, 1992. International Committee on Computational Linguistics.

[11] Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24:377–439, 1992.

[12] Eugene W. Myers and Webb Miller. Approximate matching of regular expressions. *Bulletin of Mathematical Biology*, 51(1):5–37, 1989.

[13] Kemal Oflazer. Two-level description of Turkish morphology. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics (A full version appears in* Literary and Linguistic Computing, *Vol.9 No.2, 1994.)*, Utrecht, The Netherlands, April 1993.

[14] Kemal Oflazer. Error-tolerant finite state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1), 1996. To appear.

[15] Kemal Oflazer and Cemalettin Güzey. Spelling correction in agglutinative languages. In *Proceedings of the* $4^{th}$ *Conference on Applied Natural Language Processing*, pages 194–195, Stuttgart, Germany, October 1994.

[16] Kemal Oflazer and İlker Kuruöz. Tagging and morphological disambiguation of Turkish text. In *Proceedings of the* $4^{th}$ *Conference on Applied Natural Language Processing*, pages 144–149, Stuttgart, Germany, October 1994.

[17] Jean Veronis. Morphosyntactic correction in natural language interfaces. In *Proceedings of* $13^{th}$ *International Conference on Computational Linguistics*, pages 708–713. International Committee on Computational Linguistics, 1988.

[18] Atro Voutilainen and Pasi Tapanainen. Ambiguity resolution in a reductionistic parser. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, pages 394–403, Utrecht, The Netherlands, April 1993.

[19] Sun Wu and Udi Manber. Fast text searching with errors. Technical Report TR91–11, Department of Computer Science, University of Arizona, 1991.

# A Novel Framework for Reductionistic Statistical Parsing

**Christer Samuelsson**
Universität des Saarlandes
FR 8.7, Computerlinguistik, Postfach 1150
D-66041 Saarbrücken, Germany
Internet: christer@coli.uni-sb.de

## Abstract

A reductionistic statistical framework for part-of-speech tagging and surface syntactic parsing is presented that has the same expressive power as the highly successful Constraint Grammar approach, see [Karlsson *et al* 1995]. The structure of the Constraint Grammar rules allows them to be viewed as conditional probabilities that can be used to update the lexical tag probabilities, after which low-probability tags are repeatedly removed.

Experiments using strictly conventional information sources on the Susanne and Teleman corpora indicate that the system performs as well as a traditional HMM-based part-of-speech tagger, yielding state-of-the-art results. The scheme also enables using the same information sources as the Constraint Grammar approach, and the hope is that it can improve on the performance of both statistical taggers and surface-syntactic analyzers.

## 1 Introduction

Part-of-speech (PoS) tagging consists in assigning to each word of an input text a tag from a finite set of possible tags, a tagset. The reason that this is a research issue is that a word can in general be assigned different tags depending on context. This assignment can be done in a number of different ways. One of these is statistical tagging, which is advocated in [Church 1988], [Cutting *et al* 1992] and many other articles. Here, the relevant information is extracted from large sets of often hand-tagged training data and fitted into a statistical language model, which is then used to assign the most likely tag to each word in the input text.

An alternative approach is taken in rule-based tagging, where linguistic knowledge is coded into rules that are applied to the input text to determine an appropriate tag for each word. The by far most successful work in this field is done in the Constraint Grammar [Karlsson *et al* 1995] and Finite-State Grammar [Koskenniemi *et al* 1992] frameworks. We will in this article focus on the former. First, however, we will describe what seems to be the standard procedure in statistical part-of-speech tagging, and then examine the Constraint Grammar approach a bit closer, before we discuss ways of combining them.

## 2 Traditional Statistical Part-of-Speech Tagging

This section describes a generic, but somewhat vanilla-flavoured statistical part-of-speech tagger. The tagger will use the following two information sources:

- **Lexical probabilities:**
  The probability of each tag $T^i$ conditional on the word $W$ that is to be tagged, $P(T^i \mid W)$.
- **Tag n-grams:**
  The probability of tag $T^i$ at position $k$ in the input string, denoted $T_k^i$, given that tags $T_{k-n+1} \ldots T_{k-1}$ have been assigned to the previous $n-1$ words. Often $n$ is set to two or three, and thus bigrams or trigrams are employed. In the latter case, this quantity is $P(T_k^i \mid T_{k-2}, T_{k-1})$.

A tagger is usually trained on a pretagged training corpus, which is divided into a training set, used to estimate these statistical parameters, and a set of held back data used to cope with sparse data by way of back-off smoothing. For example, the tag trigram probabilities might be estimated as follows:

$$P(T_k^i \mid T_{k-2}, T_{k-1}) \approx \lambda_3 f(T_k^i \mid T_{k-2}, T_{k-1}) + \lambda_2 f(T_k^i \mid T_{k-1}) + \lambda_1 f(T_k^i)$$

Here $f$ is the relative frequence in the training set. The parameters $\lambda_j = \lambda_j(T_{k-2}, T_{k-1}, T_k)$ may depend on the particular tags, but are required to be nonnegative and to sum to one over $j$. Appropriate values for these parameters can be estimated using the held-out portion of the training corpus by employing any of a number of techniques; much used today are deleted interpolation [Brown *et al* 1992] and modified Good-Turing smoothing, [Gale & Church 1990].

Information sources $S_1, \ldots, S_n$ are combined by multiplying the scaled probabilities:

$$\frac{P(T \mid S_1, \ldots, S_n)}{P(T)} \approx \prod_{i=1}^{n} \frac{P(T \mid S_i)}{P(T)}$$

This formula can be established by Bayesian inversion, then performing the independence assumptions, and renewed Bayesian inversion:

$$P(T \mid S_1, \ldots, S_n) = \frac{P(T) \cdot P(S_1, \ldots, S_n \mid T)}{P(S_1, \ldots, S_n)} \approx$$

$$\approx P(T) \cdot \prod_{i=1}^{n} \frac{P(S_i \mid T)}{P(S_i)} = P(T) \cdot \prod_{i=1}^{n} \frac{P(T) \cdot P(S_i \mid T)}{P(T) \cdot P(S_i)} = P(T) \cdot \prod_{i=1}^{n} \frac{P(T \mid S_i)}{P(T)}$$

At runtime, a tagger typically works as follows: First, each word is assigned the set of all possible tags according to the lexicon. This will create a lattice. A dynamic programming technique is then used to find the sequence of tags $T_1 \ldots T_n$ that maximizes

$$P(T_1 \ldots T_n \mid W_1 \ldots W_n) =$$

$$= \prod_{k=1}^{n} P(T_k \mid T_1 \ldots T_{k-1}, W_1 \ldots W_n) \approx \prod_{k=1}^{n} P(T_k \mid T_{k-2}, T_{k-1}, W_k) \approx$$

$$\approx \prod_{k=1}^{n} \frac{P(T_k \mid T_{k-2}, T_{k-1}) \cdot P(T_k \mid W_k)}{P(T_k)} = \prod_{k=1}^{n} \frac{P(T_k \mid T_{k-2}, T_{k-1}) \cdot P(W_k \mid T_k)}{P(W_k)}$$

Since $P(W_k)$ does not depend on the tag sequence, we might as well maximize

$$\prod_{k=1}^{n} P(T_k \mid T_{k-2}, T_{k-1}) \cdot P(W_k \mid T_k)$$

The dynamic search algorithm (Viterbi search) employed to this end is well-described in for example [DeRose 1988].

HMM-based taggers work in exactly the same way; the abstract way in which they are viewed is however a bit different. In HMM-based tagging, $P(T_k \mid T_{k-2}, T_{k-1})$ is often referred to as the *language model* and $P(W_k \mid T_k)$ as the *signal model*. This is since each word is viewed as a signal emitted from some (hidden) internal state consisting of a tag in the bigram case, or a pair of tags in the trigram case. The signal model describes the probability of each word being emitted by some state (tag / tag pair) and the language model the probability of state (tag / tag pair) transitions. The tagging task then consists in finding the most likely sequence of states given the observed sequence of words. It is also possible to estimate the statistical parameters from untagged text using a lexicon and an initial bias, see [Rabiner 1989] for an excellent tutorial on HMMs as applied to speech and language processing in general.

# 3 The Constraint Grammar Approach

Although not yet fully realized, the basic philosophy behind the novel statistical approach proposed in the current article is to give it the same expressive power as the highly successful Constraint Grammar system. This system performs remarkably well; [Voutilainen & Heikkilä 1994] report 99.7 percent recall, or 0.3 percent error rate, which is ten times smaller than that of the best statistical taggers. These impressive results are achieved by:

1. Utilizing a number of different information sources, and not only the stereotyped lexical statistics and n-gram tag statistics that have become the de facto standard in statistical part-of-speech tagging. For example, it uses linguistically motivated rules referring to an arbitrarily long context.

2. Not fully resolving all ambiguities when this would jeopardize the recall. In fact, the quoted 99.7 percent recall corresponds to a remaining ambiguity of approximately 1.05 tags per word, or a precision of 95 percent.

3. Evaluating on more consistently annotated data than normally employed in the field.

4. Employing an appropriate tag set, amongst other things avoiding to introduce ambiguities based on subtle distinctions of relatively small importance.

Strategy 2 means that the system trades precision for recall, making it ideal as a preprocessor for natural language systems performing deeper analysis. It has been argued that this is the sole source of the success of the Constraint Grammar approach. The experiments in this article and [de Marcken 1990] clearly indicate otherwise. The corresponding tradeoff points for statistical PoS tagging employing traditional information sources are, in the best cases, 99.2% recall at 2.17 tags per word and 97.2% recall at 1.05 tags per word, see Table 2. This is lightyears away from the 99.7% recall at 1.05 tags per word achieved by the Constraint Grammar system.

Strategy 3 is controversial as it is sometimes claimed that expert human evaluators will invariably disagree on approximately three percent of the tags, and that thus the reported error rate of 0.3 percent is theoretically impossible. We argue that: a) This is not the case. b) Even if it were the case, it is irrelevant. c) Finally, even if it were in fact relevant, it would still be consistent with the reported results. In addition to this, other experiments indicate that it is in fact possible to achieve 100 percent inter-judge agreement, see [Voutilainen & Järvinen 1995].

Firstly, this claim is based on the fact that there is a three percent disagreement amongst the persons who annotated the Brown corpus [Francis & Kucera 1982]. This does not constitute solid empirical evidence that human annotators are incapable of much more consistent hand annotation — It merely states that on one occasion, a few linguists disagreed in three percent of the cases. Anyone who has ever tried to get even a small number of linguists to agree on a choice of restaurant, not to mention on a treatment of some linguistic phenomenon, will find this figure surprisingly low. This problem seems to be more social than scientific.

In fact, if this were actually true, it would introduce some very strange effects. For example, every twelve-word sentence would have a 30 percent chance of having at least one word tagged differently by any two observers; every 23-word sentence would have a 50-percent chance of this. If one observer is the author or utterer of the sentence, and the other is a reader or a listener, and it is crucial for understanding that the receiver gets the tags right, this would render human-human communication virtually impossible. If the exact tags do not really matter, why bother with PoS tagging in the first place? Or why not, like the Constraint Grammar system, avoid trying to disambiguate these apparently irrelevant ambiguities?

Secondly, even if there were an upper limit to the level of agreement amongst any set of human classifiers, this would not necessarily imply that there is no automatic procedure for performing the classification task with much greater accuracy. Assume that a number of professors of mathematics were given limited time to individually and manually classify the numbers between 1,000,000 and 1,100,000 into primes and non-primes. No one would seriously argue that it is impossible to automatically determine whether or not a number is a prime based

on the highly likely outcome that all the professors did not arrive at identical classifications. In fact, mathematicians would most likely manually carry out an automatic procedure in the nonobvious cases, given a sufficient amount of time.

Thirdly, in the quoted experiments, the Constraint Grammar system left in five percent ambiguity, which could easily accommodate the cases where the human evaluators are claimed to necessarily disagree.

More importantly, the Constraint Grammar system is being used by an increasing number of people for robust surface-syntactic analysis. Although the syntactic analysis is shallow, and by no means as accurate as the morphological analysis, it has for example still proved sufficient for the system to constitute the sole language-analysis component of a speech interface to a virtual-reality environment, see [Karlgren *et al* 1995].

The Constraint Grammar system works as follows: First, the input string is assigned all possible tags from the lexicon, or rather, from the morphological analyzer. Then, tags are removed iteratively by repeatedly applying a set of rules, or constraints, to the tagged string. The rules are applied one by one, and the order in which they are applied is not important. Thus the effects of the various information sources are separated. When no more tags are removed by the last iteration, the process terminates, and morphological disambiguation is concluded. Then a set of syntactic tags are assigned to the tagged input string and a similar process is performed for syntactic disambiguation. This method is often referred to as *reductionistic tagging*.

The rules are formulated as finite state automata, which allows very fast processing. In more detail, the rules could be formulated as finite-state transducers, but they are not. Instead, an equivalent, and very efficient intermediate format between the original rule format and finite-state transducers is employed [Tapanainen, personal communication].

Each rule applies to a current word with a set of candidate tags. The structure of a rule is typically:

> "In this and this context, discard the following tags."

or

> "In this and this context, commit to the following tag."

We will call discarding or committing to tags the *rule action*. A typical *rule context* is:

> "There is a word to the left that is unambiguously tagged with the following tag, and there are no intervening words tagged with such and such tags."

These rules are hand-coded by a skilled linguist, a laborious and time-consuming task. (Although [Chanod & Tapanainen 1995] indicates differently.)

# 4 A Novel Reductionistic Statistical Tagger

The structure of the Constraint Grammar rules readily allows their contexts to be viewed as the conditionings of conditional probabilities, and the actions have an obvious interpretation as the corresponding probabilities.

Each context type can be seen as a separate information source, and we will again combine information sources $S_1, \ldots, S_n$ by multiplying the scaled probabilities:

$$\frac{P(T \mid S_1, \ldots, S_n)}{P(T)} \approx \prod_{i=1}^{n} \frac{P(T \mid S_i)}{P(T)} \tag{1}$$

The context will in general not be fully disambiguated. Rather than employing dynamic programming over the lattice of remaining candidate tags, the new approach uses the weighted average over the remaining candidate tags to estimate the probabilities:

$$P(T \mid \cup_{i=1}^{n} C_i) = \sum_{i=1}^{n} P(T \mid C_i) \cdot P(C_i \mid \cup_{i=1}^{n} C_i) \tag{2}$$

It is assumed that $\{C_i : i = 1, \ldots, n\}$ constitutes a partition of the context $C$, i.e., that $C = \cup_{i=1}^{n} C_i$ and that $C_i \cap C_j = \emptyset$ for $i \neq j$. In particular, trigram probabilities are combined as follows:

$$P(T \mid C) = \sum_{(T_l, T_r) \in C} P(T \mid T_l, T_r) \cdot P((T_l, T_r) \mid C)$$

Here $T$ denotes a candidate tag of the current word, $T_l$ denotes a candidate tag of the immediate left neighbour, and $T_r$ denotes a candidate tag of the immediate right neighbour. $C$ is the set of ordered pairs $(T_l, T_r)$ drawn from the set of candidate tags of the immediate neighbours. $P(T \mid T_l, T_r)$ is the symmetric trigram probability (centered around the current word).

The tagger works as follows: First tag probabilities are assigned to the input word string on purely lexical basis. Then the probabilities are updated using the various contextual information sources, corresponding to the rules of the Constraint Grammar, according to Eq. 1. Then low-probability candidate tags are removed and the probabilities are recalculated. The process terminates when the probabilities have stabilized and no more tags can be removed without jeopardizing the recall. The latter is accomplished by only removing candidate tags if their probabilities are below a certain threshold value.

As pointed out, there is no restriction on what information sources can be used; anything that can be formulated as a Constraint Grammar rule constitutes a valid conditional probability. It is of course a relevant question whether or not the independence assumptions behind Eq. 1 are valid; or, more importantly, if the resulting language model proves useful. The latter can only be evaluated empirically.

The most challenging task, which has yet only started, is to devise methods for automatically extracting "rules" from (pretagged) corpora. The current approach is to find conditionings that drastically alter the probability distributions compared to very similar conditionings; then the extra information must be decisive. Exploring the entire space of possible conditionings and comparing them is intractable, and heuristics are needed to guide the search. A simple rule that has been automatically extracted is based on the fact that the distribution is quite different when we know that there is a determiner to the left of the current word, but that there are no intervening nouns, as opposed to when there are intervening nouns.

Another thing that has not been investigated is how to incorporate the search for clause boundaries into the new framework. This could be done in a number of ways: One would be to insert them in a preprocessing step and treat them as facts in the further processing. Another approach would be to treat them the same way as other tags, and use Eq. 2 to handle uncertainty. Also this is an empirical question.

One would expect a slight decrease in performance using weighted averages instead of separate paths for handling ambiguous contexts. Incidentally, this is the main difference between the Constraint Grammar and Finite-State Grammar approaches; the rules of the latter can refer to paths through the lattice, something the rules of the former cannot. In order to ascertain that the proposed statistical method is not greatly inferior to traditional statistical PoS tagging when employing strictly conventional information sources, a series of experiments have been carried out as described in the following section.

# 5 Experiments

The novel reductionistic tagger was compared with a traditional HMM-based tagger; the latter is described in [Brants & Samuelsson 1995]. Both taggers employed strictly conventional information sources — lexical and trigram statistics. The experiments were carried out on two different corpora, the Susanne and Teleman corpora, using both the original and a reduced tagset. The Susanne corpus [Sampson 1995] is a re-annotated part of the Brown corpus [Francis & Kucera 1982] of contemporary English, and the Teleman corpus [Teleman 1974] is a corpus of contemporary Swedish, both comprising a variety of different text genres.

For the experiments, both corpora were divided into three sets, one large set (A) and two smaller sets (B and C). Three different divisions into training and test sets were used. First,

all three sets were used for both training and testing. In the second and third case, training and test sets were disjoint, the large set and one of the small sets were used for training, the remaining small set was used for testing. To indicate what is gained by taking the context into account, an additional set of experiments using only lexical probabilities and ignoring context were performed as a baseline.

Unknown words were handled by creating a decision tree of the four last letters from words with three or less occurrences. Each node in the tree was associated with a probability distribution (over the tagset) extracted from these words, and the probabilities were smoothened through linear successive abstraction, see [Brants & Samuelsson 1995].

There were two cut-off values for contexts: Firstly, any context with less than 10 observations was discarded. Secondly, any context where the probability distributions did not differ substantially from the unconditional one was also discarded. Only the remaining ones were used for disambiguation. Due to the computational model employed, omitted contexts are equivalent to backing off to whatever the current probability distribution is. The distributions conditional on contexts are however susceptible to the problem of sparse data. This was handled using partial successive abstraction as described in [Brants & Samuelsson 1995].

The results are shown in Tables 1 and 2. [1] They clearly indicate that:

- Using contextual information, i.e., trigrams, improves tagging accuracy.
- The performance of the reductionistic tagger is on par with the HMM tagger and comparable to state-of-the-art statistical part-of-speech taggers.
- Tagging the Teleman corpus is the more difficult task.

The results using the Susanne corpus are similar to those reported for the Lancaster-Oslo-Bergen (LOB) corpus in [de Marcken 1990], where a statistical N-best-path approach was employed to trade precision for recall.

The tagging speed was typically a couple of hundred words per second on a SparcServer 1000, but varied with the size of the tagset and the amount of remaining ambiguity.

# 6 Conclusions

It is reassuring to see that the reductionistic tagger performs as well as the HMM tagger, indicating that the new framework is as powerful as the conventional one when using strictly conventional information sources. The new framework also enables using the same sort of information as the highly successful Constraint Grammar approach, and the hope is that the addition of further information sources can advance the performance of statistical taggers.

Viewed as an extension of the Constraint Grammar approach, the new scheme allows making decisions on the basis of not fully disambiguated portions of the input string. The absolute value of the probability of each tag can be used as a quantitative measure of when to remove a particular candidate tag and when to leave in the ambiguity. This provides a quantitative tool to control the tradeoff between recall (accuracy) and precision (remaining ambiguity).

Extracting data directly from corpora, rather than constructing rules by introspection, as is currently the case when developing constraint grammars, is less susceptible to human error, and should consequently result in less brittle systems. Thus the proposed method can most likely also constitute an improvement on surface-syntactic analyzers.

# Acknowledgments

I would like to thank Thorsten Brants and Atro Voutilainen for comments and suggestions to improvements, and the anonymous reviewers for their constructive criticism of Section 3. I am most grateful to Thorsten Brants for providing the experimental data using the HMM tagger.

[1] **It is not very interesting to compare the accuracy for the same threshold probability, but rather for the same remaining ambiguity.**

Table 1: Results of the reductionistic experiments with the Teleman corpus

| Training | Testing | Threshold: | 0.00 | 0.05 | 0.075 | 0.10 | 0.15 | 0.20 | 0.30 | 0.50 | HMM |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Small Tagset** | | | | | | | | | | | |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,B,C | A,B,C | Recall (%) | 100.00 | 99.02 | 98.66 | 98.35 | 97.78 | 97.37 | 96.65 | 95.55 | 96.22 |
| | | Tags/word | 2.38 | 1.15 | 1.12 | 1.10 | 1.07 | 1.05 | 1.03 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 100.00 | 98.96 | 98.53 | 98.29 | 97.69 | 97.28 | 96.36 | 95.10 | 95.13 |
| | | Tags/word | 2.38 | 1.25 | 1.17 | 1.14 | 1.09 | 1.07 | 1.03 | 1.00 | 1.00 |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,B | C | Recall (%) | 98.98 | 97.72 | 97.25 | 96.81 | 96.20 | 95.53 | 94.67 | 93.34 | 92.88 |
| | | Tags/word | 2.54 | 1.21 | 1.17 | 1.14 | 1.10 | 1.07 | 1.04 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 98.98 | 97.61 | 97.14 | 96.87 | 96.15 | 95.63 | 94.26 | 92.55 | 89.27 |
| | | Tags/word | 2.54 | 1.34 | 1.25 | 1.21 | 1.14 | 1.11 | 1.04 | 1.00 | 1.00 |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,C | B | Recall (%) | 98.99 | 97.80 | 97.44 | 96.94 | 96.34 | 95.84 | 98.81 | 93.50 | 92.81 |
| | | Tags/word | 2.51 | 1.23 | 1.18 | 1.15 | 1.11 | 1.08 | 1.04 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 98.99 | 97.67 | 97.33 | 97.07 | 96.45 | 95.84 | 94.34 | 92.52 | 90.42 |
| | | Tags/word | 2.51 | 1.34 | 1.26 | 1.21 | 1.14 | 1.10 | 1.04 | 1.00 | 1.00 |
| **Large Tagset** | | | | | | | | | | | |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,B,C | A,B,C | Recall (%) | 100.00 | 98.36 | 97.92 | 97.54 | 97.03 | 96.41 | 95.31 | 93.75 | 98.35 |
| | | Tags/word | 3.69 | 1.23 | 1.18 | 1.15 | 1.11 | 1.08 | 1.04 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 100.00 | 98.30 | 97.63 | 97.20 | 96.67 | 95.57 | 93.65 | 90.59 | 90.65 |
| | | Tags/word | 3.69 | 1.43 | 1.31 | 1.26 | 1.22 | 1.16 | 1.08 | 1.00 | 1.00 |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,B | C | Recall (%) | 97.46 | 94.93 | 93.94 | 93.35 | 92.35 | 91.15 | 88.53 | 85.56 | 83.78 |
| | | Tags/word | 4.16 | 1.47 | 1.37 | 1.31 | 1.24 | 1.18 | 1.08 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 97.46 | 95.23 | 94.24 | 93.69 | 92.93 | 91.51 | 87.92 | 83.62 | 78.84 |
| | | Tags/word | 4.16 | 1.69 | 1.53 | 1.44 | 1.34 | 1.26 | 1.11 | 1.00 | 1.00 |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,C | B | Recall (%) | 96.64 | 94.04 | 93.00 | 92.09 | 90.92 | 89.46 | 86.94 | 83.58 | 81.01 |
| | | Tags/word | 4.18 | 1.48 | 1.38 | 1.32 | 1.24 | 1.18 | 1.08 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 96.64 | 94.51 | 93.27 | 92.50 | 91.02 | 89.68 | 85.86 | 81.69 | 78.05 |
| | | Tags/word | 4.18 | 1.71 | 1.54 | 1.44 | 1.34 | 1.24 | 1.10 | 1.00 | 1.00 |

Table 2: Results of the reductionistic experiments with the Susanne corpus

| Training | Testing | Threshold: | 0.00 | 0.05 | 0.075 | 0.10 | 0.15 | 0.20 | 0.30 | 0.50 | HMM |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Small Tagset** | | | | | | | | | | | |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,B,C | A,B,C | Recall (%) | 100.00 | 99.46 | 99.35 | 99.23 | 99.03 | 98.82 | 98.43 | 97.75 | 98.35 |
| | | Tags/word | 2.07 | 1.08 | 1.07 | 1.06 | 1.04 | 1.03 | 1.02 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 100.00 | 99.33 | 99.20 | 98.94 | 98.67 | 98.10 | 97.43 | 95.28 | 95.28 |
| | | Tags/word | 2.07 | 1.18 | 1.16 | 1.14 | 1.11 | 1.08 | 1.05 | 1.00 | 1.00 |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,B | C | Recall (%) | 99.22 | 98.43 | 98.28 | 98.11 | 97.78 | 97.43 | 96.91 | 95.99 | 95.76 |
| | | Tags/word | 2.23 | 1.14 | 1.11 | 1.09 | 1.07 | 1.05 | 1.02 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 99.22 | 98.27 | 98.03 | 97.78 | 97.45 | 96.80 | 96.15 | 93.42 | 91.41 |
| | | Tags/word | 2.23 | 1.25 | 1.23 | 1.19 | 1.15 | 1.11 | 1.06 | 1.00 | 1.00 |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,C | B | Recall (%) | 99.22 | 98.46 | 98.22 | 97.99 | 97.58 | 97.15 | 96.49 | 95.54 | 95.18 |
| | | Tags/word | 2.17 | 1.13 | 1.10 | 1.09 | 1.06 | 1.05 | 1.02 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 99.22 | 98.21 | 97.88 | 97.61 | 97.35 | 96.47 | 95.46 | 92.87 | 91.20 |
| | | Tags/word | 2.17 | 1.24 | 1.21 | 1.17 | 1.15 | 1.10 | 1.06 | 1.00 | 1.00 |
| **Large Tagset** | | | | | | | | | | | |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,B,C | A,B,C | Recall (%) | 100.00 | 99.25 | 99.12 | 98.96 | 98.74 | 98.44 | 98.04 | 96.87 | 99.80 |
| | | Tags/word | 2.61 | 1.10 | 1.08 | 1.07 | 1.06 | 1.04 | 1.03 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 100.00 | 99.05 | 98.88 | 98.59 | 98.20 | 97.58 | 96.72 | 93.98 | 93.98 |
| | | Tags/word | 2.61 | 1.23 | 1.20 | 1.17 | 1.14 | 1.10 | 1.07 | 1.00 | 1.00 |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,B | C | Recall (%) | 98.31 | 96.94 | 96.52 | 96.19 | 95.68 | 95.02 | 94.21 | 92.70 | 92.61 |
| | | Tags/word | 3.01 | 1.22 | 1.18 | 1.15 | 1.11 | 1.08 | 1.04 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 98.31 | 96.91 | 96.49 | 95.94 | 95.50 | 94.40 | 93.42 | 90.26 | 86.98 |
| | | Tags/word | 3.01 | 1.41 | 1.35 | 1.28 | 1.20 | 1.14 | 1.08 | 1.00 | 1.00 |
| | | **Trigram and lexical statistics** | | | | | | | | | |
| A,C | B | Recall (%) | 98.49 | 97.03 | 96.72 | 96.41 | 95.88 | 95.16 | 94.29 | 92.71 | 93.07 |
| | | Tags/word | 2.83 | 1.21 | 1.18 | 1.15 | 1.11 | 1.08 | 1.04 | 1.00 | 1.00 |
| | | **Lexical statistics only** | | | | | | | | | |
| | | Recall (%) | 98.49 | 96.95 | 96.55 | 96.05 | 95.57 | 94.44 | 93.26 | 90.31 | 88.16 |
| | | Tags/word | 2.83 | 1.36 | 1.31 | 1.25 | 1.19 | 1.13 | 1.08 | 1.00 | 1.00 |

# References


[Brants & Samuelsson 1995] Thorsten Brants and Christer Samuelsson. "Tagging the Teleman Corpus", in *Procs. 10th Nordic Conference on Computational Linguistics*, pp. 7–20, 1995.

[Brown *et al* 1992] P. F. Brown, V. J. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer and P. S. Roossin. "Class-based *n*-gram models of natural language", *Computational Linguistics 18(4)* pp. 467–479, 1992.

[Chanod & Tapanainen 1995] Jean-Pierre Chanod and Pasi Tapanainen. "Tagging French – Comparing a Statistical and a Constraint-Based Method", in *Procs. 7th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 149–156, ACL 1995.

[Church 1988] Kenneth W. Church. "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text", in *Procs. 2nd Conference on Applied Natural Language Processing*, pp. 136–143, 1988.

[Cutting *et al* 1992] Douglass R. Cutting, Julian Kupiec, Jan Pedersen and Penelope Sibun. "A Practical Part-of-Speech Tagger". in *Procs. 3rd Conference on Applied Natural Language Processing*, pp. 133–140, ACL, 1992.

[DeRose 1988] Steven J. DeRose. "Grammatical Category Disambiguation by Statistical Optimization", in *Computational Linguistics 14(1)*, pp. 31–39, 1988.

[Francis & Kucera 1982] N. W. Francis and H. Kucera. *Frequency Analysis of English Usage*, Houghton Mifflin, Boston, 1982.

[Gale & Church 1990] W. A. Gale and K. W. Church. "Poor Estimates of Context are Worse than None", in *Proc. of the Speech and Natural Language Workshop*, pp. 283–287, Morgan Kaufmann, 1990.

[Karlgren *et al* 1995] Jussi Karlgren, Ivan Bretan, Niklas Frost and Lars Jonsson. "Interaction Models, Reference, and Interactivity for Speech Interfaces to Virtual Environments", in *Procs. 2nd Eurographics Workshop on Virtual Environments — Realism and Real Time*, Monte Carlo 1995.

[Karlsson *et al* 1995] Fred Karlsson, Atro Voutilainen, Juha Heikkilä and Arto Anttila (eds). *Constraint Grammar. A Language-Independent System for Parsing Unrestricted Text*, Mouton de Gruyter, Berlin / New York, 1995.

[Koskenniemi *et al* 1992] Kimmo Koskenniemi, Pasi Tapanainen and Atro Voutilainen. "Compiling and Using Finite-State Syntactic Rules", in *Procs. 14th International Conference on Computational Linguistics*, pp. 156–162, ICCL 1992.

[de Marcken 1990] Carl G. de Marcken. "Parsing the LOB Corpus", in *Procs. 28th Annual Meeting of the Association for Computational Linguistics*, pp. 243–251, ACL 1990.

[Rabiner 1989] L. R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition", in *Proceedings of the IEEE 77(2)*, pp. 257–285, 1989.

[Sampson 1995] Geoffrey Sampson. *English for the Computer*, Oxford University Press, 1995.

[Teleman 1974] Ulf Teleman. *Manual för grammatisk beskrivning av talad och skriven svenska*, (in Swedish), Studentlitteratur, Lund, Sweden 1974.

[Voutilainen & Heikkilä 1994] Atro Voutilainen and Juha Heikkilä. "An English constraint grammar (ENGCG): a surface-syntactic parser of English", in *Procs. 14th International Conference on English Language Research on Computerized Corpora*, pp. 189–199, Zürich, 1994.

[Voutilainen & Järvinen 1995] Atro Voutilainen and Timo Järvinen. "Specifying a shallow grammatical representation for parsing purposes", in *Procs. 7th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 210–214, ACL 1995.

# A Corpus-based Probabilistic Grammar with Only Two Non-terminals

**Satoshi SEKINE** **Ralph GRISHMAN**

Computer Science Department

New York University

715 Broadway, 7th floor

New York, NY 10003, USA

*sekine,grishman@cs.nyu.edu*

**Abstract**

The availability of large, syntactically-bracketed corpora such as the Penn Tree Bank affords us the opportunity to automatically build or train broad-coverage grammars, and in particular to train probabilistic grammars. A number of recent parsing experiments have also indicated that grammars whose production probabilities are dependent on the context can be more effective than context-free grammars in selecting a correct parse.

To make maximal use of context, we have automatically constructed, from the Penn Tree Bank version 2, a grammar in which the symbols `S` and `NP` are the only real non-terminals, and the other non-terminals or grammatical nodes are in effect embedded into the right-hand-sides of the `S` and `NP` rules. For example, one of the rules extracted from the tree bank would be `S -> NP VBX JJ CC VBX NP [1]` (where NP is a non-terminal and the other symbols are terminals – part-of-speech tags of the Tree Bank). The most common structure in the Tree Bank associated with this expansion is `(S NP (VP (VP VBX (ADJ JJ) CC (VP VBX NP)))) [2]`. So if our parser uses rule [1] in parsing a sentence, it will generate structure [2] for the corresponding part of the sentence.

Using 94% of the Penn Tree Bank for training, we extracted 32,296 distinct rules (23,386 for S, and 8,910 for NP). We also built a smaller version of the grammar based on higher frequency patterns for use as a back-up when the larger grammar is unable to produce a parse due to memory limitation. We applied this parser to 1,989 Wall Street Journal sentences (separate from the training set and with no limit on sentence length). Of the parsed sentences (1,899), the percentage of no-crossing sentences is 33.9%, and Parseval recall and precision are 73.43% and 72.61%.

## 1 Introduction

The availability of large, syntactically-bracketed corpora such as the University of Pennsylvania Tree Bank affords us the opportunity to automatically build or train broad-coverage grammars. Although it is inevitable that a structured corpus will contains errors, statistical methods and the size of the corpus may be able to ameliorate the effect of individual errors. Also, because a large corpus will include examples of many rare constructs, we have the potential of obtaining broader coverage than we might with a hand-constructed grammar. Furthermore, experiments over the past few years have shown the benefits of using probabilistic information in parsing, and the large corpus allows us to train the probabilities of a grammar [8] [7] [11] [2] [4] [12].

A number of recent parsing experiments have also indicated that grammars whose production probabilities are dependent on the context can be more effective than context-free grammars in selecting a correct parse. This context sensitivity can be acquired easily using a large corpus, whereas human ability to compute such information is obviously limited. There have been

several attempts to build context-dependent grammars based on large corpora. [14] [11] [13] [2] [4] [12].

As is evident from the two lists of citations, there has been considerable research involving both probabilistic grammar based on syntactically-bracketed corpora and context-sensitivity. For example, Black proposed 'History based parsing', a context-dependent grammar trained using a large corpus [2]. History-based parsing uses decision-tree methods to identify the most useful information in the prior context for selecting the next production to try. This approach, however, requires a hand-constructed grammar as a starting point.

Bod [4] has described how to parse directly from a tree bank. A new parse tree can be assembled from arbitrary subtrees drawn from the tree bank; the parser searches for the combination with highest probability. This can, in principle, take advantage of arbitrary context information. However, the search space is extremely large, so a full search is not practical, even for a small tree bank (Bod proposes using Monte Carlo methods instead). Results have been reported only for a small tree bank of 750 ATIS sentences.

In this paper we will present a parsing method which involves both probabilistic techniques based on a syntactically-bracketed corpus and context sensitivity. We will describe a very simple approach which allows us to create an efficient parser and to make use of a very large tree bank.

## 2 An "Ultimate Parser" and a Compromise

### An "Ultimate parser" : parsing by look-up

Because of the existence of large syntactically-bracketed corpora and the advantage of context-sensitive parsing, we can contemplate an ultimate parsing strategy - parsing by table look-up. This approach is based on the assumption that the corpus covers most of the possible sentence structures in a domain. In other words, most of the time, you can find the structure of any given sentence in the corpus. If this assumption were correct, we could parse a sentence just by look-up. The system first assigns parts-of-speech to an input sentence using a tagger, and then just searches for the same sequence of parts-of-speech in the corpus. The structure of the matched sequence is the output of the system. Now we will see if the assumption is correct. We investigated the Penn Tree Bank corpus version 2, which is one of the largest syntactically-bracketed corpora, but it turned out that this strategy does not look practical. Out of 47219 sentences in the corpus[1], only 2,232 sentences (4.7%) have exactly the same structure as another sentence in the corpus. This suggests that, if we apply the above strategy, we would find a match, and hence be able to produce a parse for only 4.7% of the sentences in a new text.

### Compromise

We therefore have to make a compromise. Instead of seeking a complete match for the part-of-speech sequence of an entire sentence, we introduce partial sequence matchings based on the two important non-terminals in the Penn Tree Bank, S (sentence) and NP (noun phrase). We try to find a nested set of S and NP fragments in the given sentence so that the whole sentence is derived from a single S and then apply the look-up strategy for each fragment. In other words, at first the system collects, for each instance of S and NP in the training corpus, its expansion into S's, NP's, and lexical categories; this is, in effect, a production in a grammar with non-terminals S and NP. It also records the full constituent structure for each instance. In analyzing a new sentence, it tries to find the best segmentation of the input into S's and NP's; it then outputs the combination of the structures of the chosen segments. To assure that this strategy is applicable, we collected statistics from the Penn Tree Bank (Table 1). From the table, we can see that there are a considerable number of multiple occurrences of structures, and that a very small number of structures covers a large number of instances in the corpus. The most frequent structures for S and NP are shown just below. The numbers on the left indicate their frequency. Most of

[1] We used 96% of the corpus which will be used for the grammar training. We also processed minor modifications 1-3, which will be described later.

| Category | S | NP |
|---|---|---|
| Total instances | 88,921 | 351,113 |
| Distinct structures | 24,465 | 9,711 |
| Number of structures which cover 50% of instances | 114 | 7 |
| percentage of instances covered by structures of 2 or more occurrences | 77.2% | 98.1% |
| percentage of instances covered by top 10 structures | 27.5% | 57.9% |

Table 1: Statistics of S and NP structures

the symbols are from Penn Tree Bank[2]; symbols which we have introduced (for example, NNX or VBX) are explained later.

```
6483 (S NP (VP VBX NP))
4931 (S -NONE-)
4188 (S NP (VP VBX (SBAR S)))
1724 (S NP (VP VBG NP)
1549 (S S , CC S)
```

```
36470 (NP DT NNX)
34408 (NP NP (PP IN NP)
32641 (NP NNPX)
27432 (NP NNX)
17731 (NP PRP)
```

Although we see that many structures are covered by the data in the corpus, there could be ambiguities where two or more structures can be created from an identical part-of-speech sequence. For example, the prepositional attachment problem leads to such ambiguities. The survey on Penn Tree Bank shows us the percentage of the sequences which could be derived from S and NP with different structures are 7% and 12%, respectively. The maximum number of different structures for the same part-of-speech sequence are 7 for S and 12 for NP. However, by taking the most frequent structure for each ambiguous sequence, we can keep such mistakes to a minimum. We find that the errors caused by this are 8% and 3% for S and NP, respectively. We believe these errors can be reduced by introducing lexical or semantic information in the parsing. This will be discussed later.

From these statistics, we can conclude that many structures of S and NP can be covered by the data in the Penn Tree Bank. This result supports the idea of the parsing with two non-terminals, S and NP which segment the input, and the structure inside the segment is basically decided by table look-up. However, because we introduce non-terminals and hence introduce ambiguities of segment boundary, the overall process becomes more like parsing rather than just table look-up.

# 3 Grammar

Guided by the considerations in the last section, we try to build a grammar automatically from the Penn Tree Bank. The grammar has symbols S and NP as the only non-terminals, and the other non-terminals or grammatical nodes in the Tree Bank are in effect embedded into the right-hand-sides of the S and NP rules. For example, the following is one of the extracted rules.

```
S -> NP VBX JJ CC VBX NP
    :structure "(S <1> (VP (VP <2> (ADJ <3>)) <4> (VP <5> <6>)))";
```

(where S and NP are non-terminals and the other symbols in the rules are terminals – part-of-speech tags of the Penn Tree Bank). By this rule, S is replaced by the sequence NP VBX JJ CC

[2]Some category symbols defined in the Penn Tree Bank: VBP: non-3rd singular present-tense verb, VBZ: 3rd-person singular present-tense verb, VBD: past-tense verb, VBG: present-particle verb, NNP: singular proper noun, NNPS: plural proper noun , NN: singular or mass noun, NNS: plural noun, CD: cardinal number, FW: foreign word, SYM: symbol, CC: coordinating conjunction, IN: preposition and subordinate conjunction.

VBX NP, and in addition the rule creates a tree with grammatical non-terminals, VP's and ADJ. When the parser uses the rule in parsing a sentence, it will generate the associated structure. For example, Figure 1 shows how the sentence "This apple pie looks good and is a real treat" is parsed. The first three words and the last three words in the sentence are parsed as

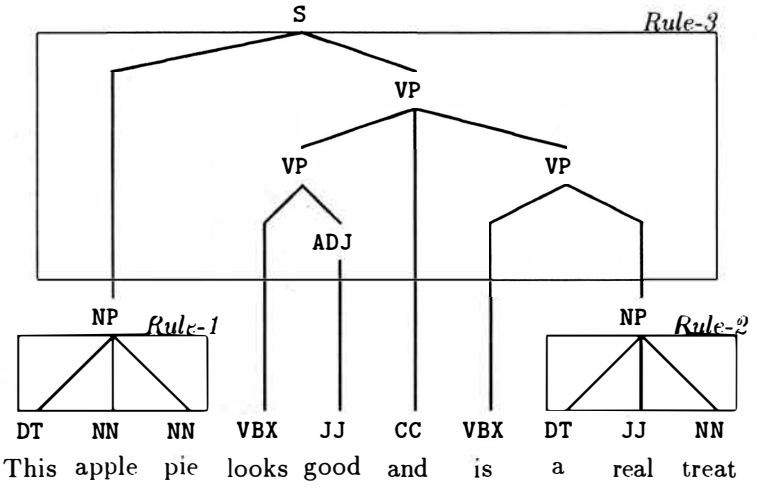



Figure 1: Example of parsed tree

usual, using the rules, NP -> DT NN NN (*Rule-1*) and NP -> DT JJ NN (*Rule-2*), respectively. The remainder is parsed by the single rule, S -> NP VBX JJ CC VBX NP (*Rule-3*). This rule constructs the entire structure under the root S. The whole tree is generated based on the three rules, although there are more than three grammatical non-terminals in the tree.

## Minor modification

We made four kinds of minor modification to the grammar, in order to improve its coverage and accuracy. First, the punctuation mark at the end of each sentence is deleted. This is to keep consistency at the end of sentences, which sometimes have a period, another symbol or no punctuation. Second, similar categories, in terms of grammatical behavior, are merged into a single category. This reduces the number of grammar rules and increases coverage of the grammar. For example, present tense and past tense verbs play a similar role in determining grammatical structure. Third, sequences of particular categories are collapsed into single instances of the category. For example, sequences of numbers, proper nouns or symbols are replaced automatically by a single instance of number, proper noun or symbol. This modification also works to reduce the number of grammar rules. Finally, we know that the Penn Tree Bank project tried to reduce the number of part-of-speech categories in order to ease the tagging effort. The Penn Tree Bank manual [10] says that they combine categories, in cases where finer distinctions can be recovered based on lexical information. So, by introducing new categories for a set of words which have different behavior from the other words in the same category, we can expect to get more information and more accurate parses.

The following is the list of modifications in the grammar:

1. Delete punctuation at the end of sentences
2. Merge Categories
   VBX=(VBP, VBZ, VBD), NNPX=(NNP, NNPS), NNX=(NN, NNS)

3. Collapse sequence into single instance
   **NNP, CD, FW, SYM**

4. Introduce new categories
   **@OF** = **of**;
   **@SNC** = Subordinating conjunction which introduce sentence (**although**, **because**, **if**, **once**, **that**, **though**, **unless**, **whether**, **while**);
   **@DLQ** = Pre-quantifier adverbs (**about**, **all**, **any**, **approximately**, **around**, **below**, **even**, **first**, **just**, **next**, **not**, **only**, **over**, **pretty**, **second**, **so**, **some**, **too**)

### Tagging

As the first step in parsing a sentence, one or more part-of-speech tags are assigned to each input token, based on the tags assigned to the same token in the training corpus. (Note that this introduces ambiguity.) Each tag has a probability which will be used in the score calculation in parsing. The probability is based on the relative frequency of the tag assigned to the token in the corpus. We set the threshold for the probability to 5% in order to make the parser efficient. Tags with smaller probability than the threshold are discarded.

$$P_{tag}(t|w) = \frac{Frequency\ of\ word\ w\ with\ tag\ t}{Frequency\ of\ word\ w} \tag{1}$$

### Score Calculation

The formulae for the probability of an individual rule $P_{rule}$ and the score of a parsed tree $S_{tree}$ are the following. The probability of a rule, $X->Y$, where $Y$ is a sequence, is based on the frequency with which $X$ dominates $Y$ in the corpus, and the frequency of the non-terminal, $X$. The score for a parse tree is the product of probability of each rule used to build the tree together with square of probability of the tag for each word in the sentence. The square factor results in putting more weight on tag-probability over rule-probability, which produce better results than balancing the weights. The best parsed tree has the highest score among the trees possibly derived from the input.

$$P_{rule}(X->Y) = \frac{Frequency\ with\ which\ X\ is\ expanded\ as\ Y}{Frequency\ of\ X} \tag{2}$$

$$S_{tree}(T) = \prod_{R:\ rules\ in\ T} P_{rule}(R) \prod_{t:\ tags\ in\ T} (P_{tag}(t|w))^2 \tag{3}$$

### Backup Grammar

Although we built a parser which can handle a large grammar, as described in the next section, it is unable to parse some long sentences, because of computer memory limitations. So we prepared a smaller grammar, for use in case the larger grammar can't parse a sentence. The small grammar consists of the rules having frequency more than 2 in the corpus. Because the number of rules is small, parsing is rarely blocked by space limitations. The parsed result of this grammar is used only when the larger grammar does not produce a parse tree. Table 2 shows the numbers of rules in the larger grammar(G-0) and the smaller grammar(G-2). The number of rules in G-0 is smaller than the number of 'distinct structures' shown in Table 1, because if there are several structures associated with one sequence, only the most common structure is kept.

## 4 Parser

It is not easy to handle such a large grammar. For example, a simple LR parser would need a huge table, while a simple chart parser would need a large number of active nodes. We therefore

| Category | Grammar-0 | Grammar-2 |
|---|---|---|
| S | 23,386 | 2,478 |
| NP | 8,910 | 2,087 |
| Total | 32,296 | 4,565 |

Table 2: Number of rules

developed a chart parser which can handle a large grammar. The key technique is that it factors grammar rules with common prefixes. Actually, the grammar rules are stored like a finite state automaton. As the grammar has thousands of rules which start from, for instance, DT, a simple chart parser has to have that same number of active nodes when it find a determiner in the input sentence. However, since active nodes indicate grammar rules which can be extended after that point, we can replace the thousands of nodes by a single pointer which points to the corresponding node in the grammar automaton. Because this node has an arc to all the possible successors, it is equivalent to thousands of active nodes in a conventional chart. Also, because we try to find only the best (highest possibility) parse tree, we can eliminate inactive nodes whose score is lower than another inactive node of the same category and span. The limits for active nodes and inactive nodes are set to 3,000,000 and 10,000, because of memory limitations.

# 5 Experiment

For our experiments, the WSJ corpus of the Penn Tree Bank is divided into two portions. One is used for training (96%) and the other part is used for testing. The training corpus is used to extract grammar rules and the test corpus contains 1,989 sentences. The parsing results are shown in Table 3. Here, "G-0" is the parsed result using the grammar with all the produced

| Grammar | number of sentence | no parse | space exhausted | sentence length | run time (sec./sent.) |
|---|---|---|---|---|---|
| G-0 | 1989 | 20 | 293 | 19.9 | 13.6 |
| G-2 | 1989 | 172 | 42 | 22.2 | 8.1 |

Table 3: Parsing Result

rules, "G-2" is the grammar with rules of frequency more than 2. "No parse" means the parser can't get S for the whole structure, "space exhausted" means that the node space of the parser is exhausted in the middle of the parsing. "Sentence length" is the average length of parsed sentences, and the run time is the parse time per sentence in seconds using a SPARC 5. Although the average run time is quite high, more than half of the sentences can be parsed in less than 3 seconds while a small number of sentences take more than 60 seconds. We can see that the number of "no parse" sentences with G-2 is larger than that with G-0. This is because there are fewer grammar rules in G-2, so some of the sequences have no matching pattern in the grammar. It is natural that the number of sentences which exhaust memory is larger for G-0 than for G-2, because of the larger number of rules.

Next, the evaluation using parseval method on parsed sentences is shown in Table 4. "Parseval" is the measurement of parsed result proposed by Black et.al. [1]. The result in the table is the result achieved by the G-0 grammar, supplemented by the result using the G-2 grammar, if the larger grammar can't generate a parse tree. These numbers are based only on the sentences which parsed (1,899 out of 1,989 sentences; in other words 90 sentences are left as unable-to-be-parsed sentences even using the back-up method). Here, "complete match" means that the result is exactly the same as the corresponding tree in the Penn Tree Bank. "No-crossing" is the

| Total sentences | 1899 |
|---|---|
| No-crossing | 643 (33.9%) |
| Ave.crossing | 2.64 |
| Parseval (recall) | 73.43% |
| Parseval (precision) | 72.61% |

Table 4: Evaluation Result

number of sentences which have no crossing brackets between the result and the corresponding tree in the Penn Tree Bank. "Ave.crossing" is the average number of crossings per sentence.

It is not easy to compare these numbers between systems, because some conditions of the test sentences, length, complexity, etc., affect the result. However, roughly speaking, these numbers are comparable to or better than the score of so-called traditional grammar, or hand-made grammars. For example, Black [3] cited the best non-crossing score using a traditional grammars as 41% and the average of several systems as 22%.

# 6 Future work

Analyzing the errors made by the parser, we found that a considerable number of unparsed sentences (including no-parse and memory-exhausted sentences) and wrongly parsed sentences contain special symbols, like ':', '–', '(' or ')'. Our strategy to enumerate structures, is not good at parsing sentences which have rare tokens, like these symbols. Furthermore, there are some odd sentences involving these symbols. For example, there are sentences which have an unbalanced parenthesis, because of incorrect division of the text into sentences or multiple sentences within a single pair of parentheses. In order to parse these sentences, we believe, special pre-treatments are needed.

As was mentioned earlier, there are several part-of-speech sequences which could generate several different structures. One source of ambiguity is annotator's inconsistencies, which we can't deal with. Another major source is attachment problems, such as prepositional attachment or conjunctive (and, or) attachment. Although it is well known that some of these ambiguities are unsolvable using only the context within the sentence, many of them are heavily related to the lexical or semantic information within the sentence. We have been conducting research on automatically acquiring these selectional constraints, and we are planning to incorporate this semantic knowledge into the parser [9], [15].

Introducing lexical information in the parser is also useful for other kinds of the structural disambiguation. We showed this in minor modification 4, by introducing new categories based on lexicon. However, the method we used to choose the candidates depended on human intuition. We are considering creating an automatic method to identify those words for which it is beneficial to make a new category.

# 7 Conclusion

We developed a corpus-based probabilistic grammar whose rules are extracted from syntactically-bracketed corpora. The grammar has only two non-terminals, `S` and `NP`. The rules of the grammar contain grammatical non-terminal within the tree. This feature introduces context-sensitivity for the terminals.

Corpus-based grammar generation has a significant advantage over building a grammar by hand, particularly if we aspire to a high degree of coverage. Once we have a syntactically-bracketed corpus for a new domain, a grammar can be automatically created for that domain. While building syntactically-bracketed corpora is not so easy, large-scale corpora have been

successfully constructed by teams of coders; building a very-broad-coverage grammar by hand has proven much more challenging.

We conducted an experiment using two grammars derived from a tagged corpus. The accuracy is about the same or better than with a conventional grammar. As this grammar uses only part-of-speech information, we may be able to improve it by incorporating lexical or semantic information.

# 8 Acknowledgments

The work reported here was supported by the Advanced Research Projects Agency under contract DABT63-93-C-0058 from the Department of the Army. Also, we would like to thank our colleagues at NYU and the member of Penn Tree Bank Project.

# References

[1] E.Black, et.al. "A procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars" *Proc. of Fourth DARPA Speech and Natural Language Workshop* (1991)

[2] E.Black, F.Jelinek, J.Lafferty, D.Magerman, R.Mercer, S.Roukos "Towards History-based Grammars: Using Richer Models for Probabilistic Parsing" *ACL-93* (1993)

[3] E.Black "Parsing English By Computer: The State Of the Art" *ATR International Workshop on Speech Translation* (1993)

[4] R.Bod "Using an Annotated Corpus as a Stochastic Grammar" *EACL-93* (1993)

[5] E.Brill "Automatic Grammar Induction and Parsing Free Text: A Transformation-Based Approach" *ACL-93* (1993)

[6] T.Briscoe, J.Carroll "Generalized Probabilistic LR Parsing of Natural Language (corpora) with Unification-Based Grammars" *Computational Linguistics Vol.19,No.1* (1993)

[7] M.Chitaro, R.Grishman "Statistical parsing of messages" *In proceedings of the Speech and Natural Language Workshop* (1990)

[8] R.Garside, F.Leech "A Probabilistic Parser" *EACL-85* (1985)

[9] R.Grishman, J.Sterling "Generalizing Automatically Generated Selectional Patterns" *COLING-94* (1994)

[10] M.Marcus, B.Santorini, M.Marcinkiewicz "Building a large annotated corpus of English: the Penn Tree bank" *in the distributed Penn Tree Bank Project CD-ROM*, Linguistic Data Consortium, University of Pennsylvania.

[11] D.Magerman, C.Weir "Efficiency, Robustness and Accuracy in Picky Chart Parsing" *ACL-92* (1992)

[12] D.Magerman "Statistical Decision-Tree Models for Parsing" *ACL-95* (1995)

[13] Y.Shabes, R.Waters "Lexicalized Context-Free Grammars" *ACL-93* (1993)

[14] R.Simmons, Y.Yu "The Acquisition and Application of Context Sensitive Grammar for English" *ACL-91* (1991)

[15] S.Sekine, S.Ananiadou, J.J.Carroll, J.Tsujii: "Linguistic Knowledge Generator" *COLING-92* (1992)

# HEURISTICS AND PARSE RANKING*

B. Srinivas
Department of Computer Science
University of Pennsylvania
srini@linc.cis.upenn.edu

Christine Doran
Department of Linguistics
University of Pennsylvania
cdoran@linc.cis.upenn.edu

Seth Kulick
Department of Computer Science
University of Pennsylvania
skulick@linc.cis.upenn.edu

**Abstract**

There are currently two philosophies for building grammars and parsers – Statistically induced grammars and Wide-coverage grammars. One way to combine the strengths of both approaches is to have a wide-coverage grammar with a heuristic component which is domain independent but whose contribution is tuned to particular domains. In this paper, we discuss a three-stage approach to disambiguation in the context of a lexicalized grammar, using a variety of domain independent heuristic techniques. We present a training algorithm which uses hand-bracketed treebank parses to set the weights of these heuristics. We compare the performance of our grammar against the performance of the IBM statistical grammar, using both untrained and trained weights for the heuristics.

## 1 Introduction

Although statistical POS taggers [Church1988] have conclusively out performed hand-crafted POS taggers [Harris1962; Klein and Simmons1963] of the pre-statistical NLP era, the same cannot be said about parsers and the grammars they use. There are currently two philosophies for building grammars and parsers. Wide-coverage grammars (WCGs) such as [Grover et al.1993; Alshawi et al.1992; Karlsson et al.1994; Group1995] are mostly hand-crafted and are designed to be domain-independent. They are usually based on a particular grammar formalism, such as CFG, Dependency Grammar or LTAG. Statistically induced grammars (SIGs) [Jelinek et al.1994; Magerman1995; Mori and Nagao1995] on the other hand, are trained on specific domains using a manually annotated corpus of parsed sentences from the given domain.

Aside from the methodological differences in grammar construction, the two approaches differ in the richness of information contained in the output parse structure. Wide-coverage grammars generally provide a far more detailed parse than that output by a statistically induced grammar. Also, the linguistic knowledge which is overt in the rules of hand-crafted grammars is hidden in the statistics derived by probabilistic methods, which means that generalizations are also hidden and the full training process must be repeated for each domain.

Ideally, one would like to combine the strengths of both approaches, by having a rule-based grammar with a heuristic component which is domain independent but whose contribution is tuned to particular domains. One way to capture the strengths of SIGS of associating preferences with parses in a WCG, is to introduce heuristics to rank parses. Both domain independent and domain dependent heuristics can be included in WCGs. In contrast, a SIG does not distinguish between grammatical knowledge, knowledge of domain independent and domain dependent preference.

Researchers [Hobbs and Bear1994; McCord1993] have suggested domain-independent heuristics in the context of CFGs. However, in recent grammar formalisms the lexicon has come to play a more central role than it does in non-lexicalized CFGs. These grammars are 'lexicalized' in that each elementary structure of the grammar is associated with at least one lexical item. These structures provide a domain over which syntactic and semantic constraints can be specified by the lexical item that selects them. It is precisely the power of these "extended domains of locality" which makes lexicalized grammars so well-suited to the specification of natural language grammars. In this paper, we show that the lexicalized grammars with extended domains of locality provide a unique opportunity to apply known disambiguation

*The authors would like to thank Aravind Joshi, Anoop Sarkar, Joseph Rosenzweig, Ted Briscoe and Henry Thompson for their valuable comments and assistance with this work. The work was partially supported by ARO grant DAAL03-89-0031, ARPA grant N00014-90-J-1863, NSF STC grant DIR-8920230, and Ben Franklin Partnership Program (PA) grant 93S.3078C-6.

techniques in an efficient manner, and in particular to exploit lexically sensitive heuristics that cannot be stated easily on a non-lexicalized grammar.

In this paper we discuss a three-stage approach to disambiguation in parsing, with particular emphasis on lexicalized grammars. The first stage is part-of-speech tagging on the input sentence. By ascertaining the part-of-speech of each word, one is able to greatly reduce the number of structures (trees) to be considered in the parsing process itself. The second opportunity for disambiguation comes when a set of possible structures has been selected for each lexical item. At this point certain structures can be filtered out, using statistical methods and heuristics which take advantage of the shapes of the structures. Finally, any remaining ambiguous parses are ranked using further heuristics.

We then discuss a method for combining the heuristics and automatically training their weights. We use hand-bracketed treebank parses for the training process. These trained heuristics are then used to evaluate the performance of our system against the IBM probabilistic grammar (on IBM-manual data). While it may appear that disambiguation and evaluation are disparate issues, we believe that it is useful to consider them together insofar as the results of the disambiguation step are the input to any evaluative processes.

The particular system we are using for our experiments is a wide-coverage, domain-independent system based on lexicalized Tree Adjoining Grammar (LTAG). We will assume that readers are familiar with the formalism. See [Joshi et al.1975], [Vijay-Shanker1987], [Vijay-Shanker and Joshi1991] and [Schabes et al.1988] for description of TAG and lexicalized TAG. A summary of each component of the XTAG system [Group1995] is presented in Table 1.

# 2 Disambiguation Techniques

There are a number of stages where syntactic ambiguity – lexical and structural – can be reduced in parsing. We will discuss: (1) part-of-speech tagging prior to parsing, (2) tree/subcat[1] filtering and weighting techniques, and (3) heuristics to rank the generated parses. The first and last are general techniques which are applicable to all types of grammars; tree filtering and tree weighting take advantage of the particular properties of lexicalized grammars. The combination of these three techniques has proven extremely effective in attacking the problem of ambiguity while simultaneously improving the efficiency of the parser.

## 2.1 Part-of-speech Tagging

It is well known that lexical ambiguity with regard to part-of-speech (POS) is one of the greatest sources of overall ambiguity. This is particularly important in a lexicalized grammar, where each word is associated with multiple structures for each POS it may have. In our grammar, for example, the word *try* selects 59 verb trees and 17 noun trees; simply by identifying its POS we substantially reduce the number of trees it contributes to the parsing process. When this is done for each word in a sentence, the reduction in number of trees selected is enormous. Consider two examples: the NP *the act of allowing fresh air into a room* receives 26 parses untagged, and POS tagging reduces that number to 4; the sentence *the second part is the name of your personal computer* receives 32 parses without POS tagging, and only 8 parses with tagging.[2]

## 2.2 Tree Filtering and Tree Weighting Techniques

**Filtering** The second opportunity for disambiguation comes after trees have been selected for each word of the input sentence, in the first pass of the parser. Structural properties of trees such as the span of the tree and the position of the anchor in the tree are used to weed out unsuitable trees. This has the effect of eliminating, for example, a noun tree with a determiner position when there is no determiner to the left

[1] Henceforth, we will just say "trees" but the reader should bear in mind that these techniques are equally applicable to other lexicalized grammars, whatever the structure they associate with each lexical item.

[2] The first example is from the Alvey test sentences and the second from the IBM Manual Corpus.

| Component | Details |
|---|---|
| Morphological Analyzer and Morph Database | Consists of approximately 317,000 inflected items.<br>Thirteen parts of speech are differentiated.<br>Entries are indexed on the inflected form and return the root form, POS, and inflectional information.<br>Database does not address derivational morphology. |
| POS Tagger and Lex Prob Database | Wall Street Journal-trained trigram tagger [Church1988]<br>Decreases the time to parse a sentence by an average of 93%. |
| POS Blender | Combines information from the Morphology and the POS tagger<br>Outputs N-best POS sequences with morphological information for each word of the input sentence. |
| Tree Database | 566 trees, divided into 40 tree families and 62 individual trees.<br>Tree families represent subcategorization frames.<br>E.g., the intransitive tree family contains the following trees: indicative, wh-question, relative clause, imperative and gerund. Individual trees are generally anchored ($\diamond$) by non-verbal lexical items that substitute or adjoin into the clausal trees.<br>Feature values may be specified within a tree or may be derived from the syntactic database. |
| Syntactic Database and Statistical Database | Associates lexical items with the appropriate trees and tree families based on subcategorization information.<br>Extracted from OALD and ODCIE and contains more than 105,000 entries.<br>Each entry consists of: the uninflected form of the word, its POS, the list of trees or tree-families associated with the word, and a list of feature equations that capture lexical idiosyncrasies. |
| X-Interface | Menu-based facility for creating and modifying tree files.<br>User controlled parser parameters: parser's start category, enable/disable/retry on failure for POS tagger.<br>Storage/retrieval facilities for elementary and parsed trees as text and postscript files.<br>Graphical displays of tree and feature data structures.<br>Hand combination of trees by adjunction or substitution for diagnosing grammar problems. |

Table 1: **XTAG System Summary**

of the noun in the input string. At this stage we have eliminated trees which would never be used in any parse of the input string, thus reducing the initial search space as early as possible for the parser. This does not however reduce the number of parses produced.

Furthermore, statistical information about the usage frequency of trees has been collected by parsing corpora. This information has been compiled into a database that is used by the parser. The database contains tree unigram frequencies of trees collected by parsing the Wall Street Journal, IBM manual, and ATIS corpora. The top 15 trees for each of these corpora are shown in Table 2. It is interesting to note that the *PP_Attaches_to_NP* structure ranks second in ATIS and ranks lower in IBM Manual and WSJ corpora. The parser, augmented with the statistical database, assigns each word of the input sentence the three most frequently used trees for that word. Statistical filtering removes trees which might ultimately participate in a less likely parse. Note that this is different from inducing the grammar from a particular corpus, in that the database contains a composite of frequencies. On failure the parser retries using all the trees suggested by the syntactic database for each word. [3] The augmented parser succeeds in parsing 50% of input sentences using only the top three trees for each word.

| # | ATIS | Prob. | IBM Manual | Prob. | WSJ | Prob. |
|---|---|---|---|---|---|---|
| 1 | Noun_Phrase | (0.260) | Determiner | (0.175) | Noun_Phrase | (0.184) |
| 2 | PP_Attaches_to_NP | (0.142) | Noun_with_Det | (0.174) | Noun_Mods_Noun | (0.151) |
| 3 | Noun_Mods_Noun | (0.105) | Noun_Mods_Noun | (0.112) | Determiner | (0.089) |
| 4 | Noun_with_Det | (0.094) | Aux_Verb | (0.095) | Noun_with_Det | (0.079) |
| 5 | Determiner | (0.092) | Noun_Phrase | (0.073) | Aux_Verb | (0.074) |
| 6 | Imperative_Double_Obj | (0.056) | Adjective | (0.044) | Adjective | (0.055) |
| 7 | Aux_Verb | (0.050) | PP_Attaches_to_VP | (0.041) | PP_Attaches_to_VP | (0.038) |
| 8 | Adjective | (0.045) | Passive_Trans | (0.037) | PP_Attaches_to_NP | (0.027) |
| 9 | Inverted_Aux | (0.044) | Indic_Transitive | (0.035) | PRO | (0.025) |
| 10 | Extracted_Predicative | (0.030) | PP_Attaches_to_NP | (0.033) | Pre-VP_Adverb | (0.021) |
| 11 | Imperative_Transitive | (0.021) | Imperative_Transitive | (0.015) | Indic_Transitive | (0.021) |
| 12 | Stacked_Det | (0.011) | PRO | (0.012) | Indic_Scomp | (0.014) |
| 13 | Obj_Extr_Trans | (0.008) | VP_Negation | (0.009) | Rel_Cl_Transitive | (0.011) |
| 14 | PP_Attaches_to_VP | (0.007) | Indic_Intrans | (0.008) | Sent_Adv | (0.010) |
| 15 | Comp_Extr_SComp | (0.004) | Post-VP_Adverb | (0.007) | Post-VP_Adv | (0.010) |

Table 2: **The Unigram probabilities of top 15 trees from ATIS, IBM Manual and WSJ Corpora**

Continuing with an example discussed above, *the second part is the name of your personal computer*, the number of parses is reduced from 8 to 3 by the tree-filtering techniques. In addition, these methods speed the overall runtime by a factor of 8.

**Local Weighting** Having selected the top three trees for each word in the sentence, we add general (dis)preference weightings for clause types. Some sample preference heuristics are:

1. Disprefer relative clause trees.
2. Disprefer topicalization.
3. Disprefer predicative trees.

We also include certain lexical preferences for highly ambiguous function words, such as:

1. Prefer *of* as NP over VP modifier.
2. Prefer *this* as Determiner over Noun.
3. Prefer *to* as Verb over Preposition.
4. Prefer *that* as Complementizer over Determiner.
5. Prefer *which* as Complementizer over Noun.

The four parses remaining for our example after tree filtering are now passed onto the next stage of processing with their associated weights.

[3] We are currently working on an agenda-based parser, which would prefer higher ranked trees and would not require such a fall-back strategy.

## 2.3 Global Heuristics for Ranking the Generated Parses

Any parses which survive POS tagging and tree-filtering are generated by the second stage of the parser in ranked order. This ranking is determined using heuristics based on structural preferences in the derivation. These preferences include:

1. Prefer argument positions to adjunct positions (here, this amounts to preferring fewer adjunction operations).
2. Prefer low-attached PPs.
3. Prefer high-attached Adjectives.

These heuristics differ in scope from the more local preferences described in the previous section, which are applied in the first stage of the parser.

For our example sentence, the heuristics rank highest the same sentence which human judges also selected as the correct parse (parse #3). The rule governing PP attachment is clearly the most important for this sentence. The three ranked sentences are shown schematically in Figure 1. The ranking of parses facilitates selection of a number of parses to be passed on to further levels of processing. For purposes of evaluation, we can select the number of parses equal to the number considered in the system being evaluated against. In applications emphasizing speed, only the highest-ranked parse will be considered. In applications emphasizing coverage, the top $n$ parses can be considered.

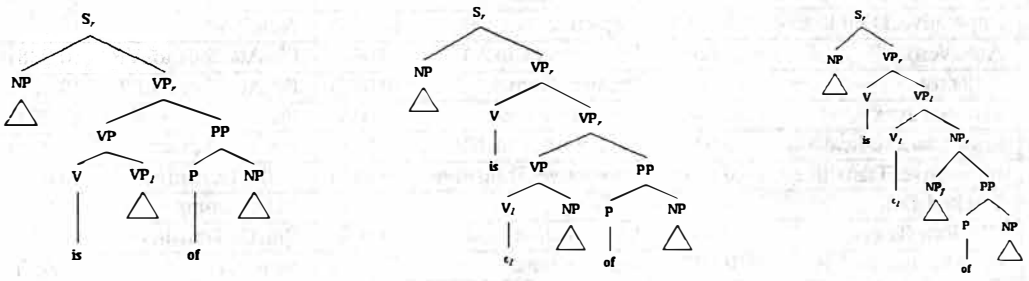



Figure 1: **Ranked Parses Generated For Sample Sentence**
*The second part is the name of your personal computer*

## 2.4 Discussion of Heuristics

Similar heuristics have been used by (non-lexicalized) CFG-based parsers (e.g. [Hobbs and Bear1994], [McCord1993], and [Nagao1994]). In this section we highlight the differences between lexicalized grammars and CFGs in terms of the methodology and applicability of the heuristics.

Local heuristics which refer to single lexical items or entire clausal constructions are easily stated within a lexicalized grammar, where they refer to a single constituent. Also lexical sensitivity can be exploited quite elegantly in the rules of a lexicalized grammar where the entire argument structure of a given lexical item is localized within an single rule. In a CFG, for example, the argument structure would be dispersed across a number of rules and would be more difficult to identify.

The weights of both sets of heuristics are combined by a linear function that is at first hand-set and then altered by the heuristics training described below. As we will discuss in the experimental section, we have developed a means of automatically setting these weights using a training algorithm. This technique give us the advantages of statistical methods in identifying structural generalizations in texts, while retaining the linguistically transparency and ease of portability of a rule-based system.

The heuristics themselves are entirely domain independent; only the weights are affected by training. When working with a particular domain, one could easily augment these general heuristics with more specialized ones. In this way, the heuristics have an advantage over purely statistical approaches which

must be completely retrained for each new genre. Hobbs and Bear [1994] have proposed two general heuristics but do not suggest of a way to combine or evaluate them in any detail. McCord's [1993] heuristics are domain independent but the contributions of the heuristics are hand-tuned for the computer-manual domain.

Alshawi and Carter [1994], working with a non-lexicalized CFG-based large scale grammar, specialize their grammar to the ATIS domain using heuristics whose contributions are trained on hand-picked correct Quasi-Logical Forms. They find that heuristics based on lexical semantic collocations dominate other heuristics in selecting the correct parse. However, these heuristics are domain dependent and need to be computed for each domain. Based on their results, we expect that adding semantic collocational information to our heuristic set would improve our results. Like our current heuristics, we will only add collocations which are stable across several genre of text.

# 3 Evaluation

As discussed above, the ranking of parses for each sentence is crucial to evaluating the grammar against other systems. XTAG has recently been used to parse Wall Street Journal[4], IBM manual, and ATIS corpora as a means of evaluating the coverage and correctness of XTAG parses.

## 3.1 Coverage

To evaluate the coverage of our grammar, a sentence is considered to have parsed if XTAG produces any parses. Verifying the presence of the correct parse among the generated parses is done manually at present by random sampling. Results without the use of parse ranking are shown in Table 3.[5] It is worth emphasizing that the XTAG grammar is truly wide-coverage and has not been fine-tuned to any particular genre, unlike many other grammars.

| Corpus | # of Sentences | % Parsed | Av. # of Parses/Sent |
|---|---|---|---|
| WSJ | 18,730 | 41.22 % | 7.46 |
| IBM Manual | 2040 | 75.42% | 6.12 |
| ATIS | 524 | 88.35% | 6.0 |

Table 3: **Performance of XTAG on various corpora**

Performance on the WSJ corpus is lower relative to IBM and ATIS due to the wide-variety of syntactic constructions present. Even grammars induced on the partially bracketed WSJ corpus have fairly low performance (e.g. 57.1% sentence accuracy for [Schabes et al.1993]).

## 3.2 Correctness

The second aspect of evaluating our grammar is determining whether we obtain the correct parse, as defined by a hand-bracketed corpus. We use a crossing bracket measure for evaluation. Crossing brackets is the percentage of sentences with no pairs of brackets crossing the treebank bracketing (i.e. ( ( a b ) c ) has a crossing bracket measure of one if compared to ( a ( b c ) ) ). Recall is the ratio of the number of constituents in the XTAG parse to the number of constituents in the corresponding Treebank sentence. Precision is the ratio of the number of correct constituents to the total number of constituents in the XTAG parse.

**Experiment 1: Equally Weighted Heuristics** A more detailed experiment to measure the crossing bracket accuracy of the XTAG-parsed IBM-manual sentences has been performed; performance results

[4] Sentences of length $\leq$ 15 words.

[5] This result was previously reported in [Doran et al.1994].

from other parsers on WSJ or ATIS are not available. In this experiment, XTAG-parses of 1100 IBM-manual sentences have been ranked using the heuristics discussed in this paper, with all heuristics weighted equally. The ranked parses have been compared[6] against the bracketing given in the Lancaster Treebank of IBM-manual sentences[7]. Table 4 shows the results of XTAG obtained in this experiment, which used the highest ranked parse for each system. It also shows the results of the latest IBM statistical grammar [Jelinek et al.1994] on the same genre of sentences. Only the highest-ranked parse of both systems was used for this evaluation.

| System | # of sentences | Zero Crossing Bracket % | Recall % | Precision % |
|---|---|---|---|---|
| XTAG | 1100 | 81.29% | 82.34% | 55.37% |
| IBM Statistical grammar | 1100 | 86.20% | 86.00% | 85.00% |

Table 4: **Performance of XTAG on IBM-manual sentences**

As can be seen from Table 4, the precision figure for the XTAG system is considerably lower than that for IBM. For the purposes of comparative evaluation against other systems, we had to use the same crossing-brackets metric though we believe that the crossing-brackets measure is inadequate for evaluating a grammar like XTAG. There are two reasons for the inadequacy. First, the parse generated by XTAG is much richer in its representation of the internal structure of certain phrases than those present in manually created treebanks (e.g. IBM: [$_N$ your personal computer], XTAG: [$_{NP}$ [$_G$ your] [$_N$ [$_N$ personal] [$_N$ computer]]]). The detailed bracketing provided by XTAG make the parse structure more informative. This is reflected in the number of constituents per sentence, shown in the last column of Table 5. We are aware of the fact that increasing the number of constituents also increases the recall percentage.

We measured the number of constituents per parse with the internal structure of NPs and VPs removed to more closely correspond to the Treebank parse. Precision improved to 66.64% while recall dropped slightly to 80.20%. We believe that the precision percentage can be further improved by flattening all adjunction structures (such as PPs, Adverbs and other modifiers) since each adjunction in LTAG adds an extra level of structure.

| System | Sent. Length | # of sent | Av. # of words/sent | Av. # of Constituents/sent |
|---|---|---|---|---|
| XTAG | 1-10 | 654 | 7.45 | 22.03 |
| | 1-15 | 978 | 9.13 | 30.56 |
| IBM Stat. Grammar | 1-10 | 447 | 7.50 | 4.60 |
| | 1-15 | 883 | 10.30 | 6.40 |

Table 5: **Constituents in XTAG parse and IBM parse**

A second reason for considering the crossing bracket measure inadequate for evaluating XTAG is that the primary structure in XTAG is the derivation tree. Two identical bracketings for a sentence can have completely different derivation trees (e.g. *kick the bucket* as an idiom vs. a compositional use). A more direct measure of the performance of XTAG would evaluate the derivation structure, which captures the dependencies between words.

**Experiment 2: Weighted Heuristics** For Experiment 2, an iterative process is used to train the heuristics. This is similar in spirit to the the work presented in [Alshawi and Carter1994], although the details of the training algorithm differ. In addition, the training algorithm in [Alshawi and Carter1994] used semantic representation for sentences restricted from ATIS domain as the "gold" corpus to train on. We experiment with phrase-structure parses of sentences from IBM-manual data and our training algorithm uses hand-bracketed Lancaster Treebank parses as the "gold" corpus.

[6] We used the Parseval program written by Phil Harison (phil@atc.boeing.com).

[7] The Treebank was obtained through Salim Roukos (roukos@watson.ibm.com) at IBM.

An IBM-manual corpus of 931 sentences was randomly split into three groups: TRAIN - 626 sentences, HELD-OUT - 205 sentences, and TEST - 100 sentences. TRAIN is used to train new heuristic weights, while HELD-OUT is used as a control to prevent overtraining on TRAIN. TEST is set aside to be used as a test for the final weightings. We use the same gold standard as in experiment 1. Each iteration chooses, at random, one of the heuristics to adjust, and a random amount to adjust it by. All of the sentences in TRAIN are then ranked using the adjusted heuristic (with the other heuristics unchanged).

The top six parses by this ranking are then compared with the parse from the gold standard. The crossing bracket, recall, and precision measures computed by Parseval are all taken equally into account when determining whether a heuristic change resulted in an improvement. If the result of this comparison (for all the sentences in TRAIN as a whole) is an improvement, then the random change is kept. Sentence group HELD-OUT is used to determine when this iterative process ends. Each time a heuristic change results in an improvement, all of the sentences in HELD-OUT are ranked using the new heuristic settings, and the overall result is compared to the last score for HELD-OUT. The iteration terminates when there has been no improvement three consecutive times.

Table 6 shows the results of this training process. The sentence group column indicates the set of sentences (HELD-OUT/TEST) on which the performance was measured. The Experiment column indicates the three sets of experiments performed on each of the sentence sets. The performance in all the experiments has been measured using the crossing bracket metric on the top six parses. The metric crossing bracket accuracy measures the percentage number of sentences with no crossed brackets. Crossing bracket average gives the average number of crossed bracket errors per parse.

| Sentence Group | Experiment | Zero Crossing Bracket % | Crossing Bracket Average | Recall % | Precision % |
|---|---|---|---|---|---|
| HELD-OUT | No heuristics | 77.56 | 1.15 | 81.57 | 54.01 |
| | No preference | 81.46 | 1.08 | 82.42 | 54.50 |
| | Preferences Trained I | 83.41 | 1.05 | 83.46 | 55.17 |
| | Preferences Trained II | 83.90 | 1.03 | 83.71 | 55.33 |
| TEST | No heuristics | 85.00 | 1.03 | 80.87 | 54.45 |
| | No preference | 87.00 | 0.97 | 82.11 | 55.26 |
| | Preferences Trained I | 88.00 | 0.96 | 82.61 | 55.59 |
| | Preferences Trained II | 89.00 | 0.94 | 82.98 | 55.83 |

Table 6: **Performance Results on the Crossing Bracket Measure for the HELD-OUT and TEST set without any heuristics, with heuristics but with no preference weights, and with heuristics weighted by weights set by the training process.**

1. No heuristics: This experiment refers to the baseline performance of the system in which all the parses were ranked equally and in case there are more than six parses, the first six on the list of parses were chosen for measuring the performance. The results are shown in the first and the fourth rows for the HELD-OUT and the TEST set respectively.
2. No preference: In this experiment, parses were ranked using the heuristics, however, all the heuristics had equal contributions to the rank of a parse. The results are shown in the second and the fifth rows for the HELD-OUT and the TEST set respectively. Including heuristics has improved the performance when compared to the performance with no heuristics.
3. Preferences Trained: In this experiment performance was measured on the top six parses that were ranked using the heuristics weighted according to the weights set by the training process. In Table 6 we show the results of the performance using the weights resulting from two training runs.

## 3.3 Discussion of the Experiments

As can be seen from the "Preferences Trained II" line in the "Test" data from Table 6, using our automatic training method we outperform the IBM statistical grammar result, given in Table 4. The training algorithm

improves the performance of the heuristics slightly. We believe that this is due to the domain-independent nature of the heuristics which makes the training algorithm less crucial than it might be if the heuristics included were domain-dependent. We are currently investigating this issue in the context of WSJ data.

We are collecting a hand-checked corpus of XTAG parsed sentences that includes the derivation trees which we will be able to use as a "gold" standard for future experiments. This will enable us to replace the crossing bracket measure with an exact match metric for the training procedure.

These experiments show that with domain independent heuristics the performance of a wide-coverage grammar can equal or better the performance of a statistically induced grammar. There is further opportunity for improvement by adding domain-dependent heuristics, in particular semantic collocations. However, in the interest of portability, we will not include such information in the present system.

# 4 Future Work

The heuristics presented in this paper do not completely exploit the lexical sensitivity provided by the LTAG representation. Towards incorporating more lexical information, we intend to use lexical collocation information as one of the factors for disambiguation. Weare in the process of collecting lexical collocation information that is reasonably domain independent from various corpora. We expect that including the such information will improve the results significantly.

We will continue to add and test the effectiveness of new heuristics, and to continue to refine our training algorithm. In addition, we plan to evaluate our grammar using the same techniques on other genre of text, in particular Wall Street Journal data.

As mentioned earlier, we feel that crossing bracket metric is not an adequate for measuring the performance of the grammar formalisms such as LTAG. Hence it serves as a poor objective function to improve the performance of such grammars. We are forced to using the crossing bracket measure since the only corpora available at present that serve as gold standard are ones that are constituent-bracketed. To overcome this problem, we are collecting an LTAG-parsed corpus with each sentence annotated with its correct derivation tree. This resource will prove invaluable for future research in training disambiguation heuristics.

# 5 Conclusion

In this paper, we have discussed a three-stage approach to disambiguation in the context of a lexicalized grammar, using a variety of domain independent statistical techniques. We have presented a training algorithm which uses hand-bracketed treebank parses to set the weights of these heuristics. We show that the performance of our grammar is comparable to the performance of the IBM statistical grammar, using both untrained and trained weights for heuristics.

# References


[Alshawi and Carter1994] Hiyan Alshawi and David Carter. 1994. Training and scaling preference functions for disambiguation. *Computational Linguistics*, 20(4).

[Alshawi et al.1992] Hiyan Alshawi, David Carter, Richard Crouch, Steve Pullman, Manny Rayner, and Arnold Smith. 1992. *CLARE – A Contextual Reasoning and Cooperative Response Framework for the Core Language Engine.* SRI International, Cambridge, England.

[Brill1994] Eric Brill. 1994. Some advances in transformation-based part of speech tagging. In *Proceedings of AAAI94.*

[Church1988] Kenneth Ward Church. 1988. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *2nd Applied Natural Language Processing Conference*, Austin, Texas.

[Doran et al.1994] Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. 1994. XTAG System - A Wide Coverage Grammar for English. In *Proceedings of the* $17^{th}$ *International Conference on Computational Linguistics (COLING '94)*, Kyoto, Japan, August.

[Group1995] The XTAG Research Group. 1995. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS 95-03, University of Pennsylvania.

[Grover et al.1993] Claire Grover, John Carroll, and Ted Briscoe, 1993. *The Alvey Natural Language Tools Grammar*, 4th release edition.

[Harris1962] Zelig Harris. 1962. *String Analysis of Language Structure*. Mouton and Co., The Hague.

[Hobbs and Bear1994] Jerry R. Hobbs and John Bear. 1994. Two Principles of Parse Preference. In *Current Issues in Natural Language Processing: In Honour of Don Walker*. Giardini with Kluwer.

[Jelinek et al.1994] F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, A. Ratnaparkhi, and S. Roukos. 1994. Decision Tree Parsing using a Hidden Derivation Model. *ARPA Workshop on Human Language Technology*, pages 260–265.

[Joshi and Srinivas1994] Aravind K. Joshi and B. Srinivas. 1994. Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing. In *Proceedings of the* $17^{th}$ *International Conference on Computational Linguistics (COLING '94)*, Kyoto, Japan, August.

[Joshi et al.1975] Aravind K. Joshi, L. Levy, and M. Takahashi. 1975. Tree Adjunct Grammars. *Journal of Computer and System Sciences.*

[Karlsson et al.1994] Karlsson, Voutilainen, Heikkilä, and Anttila. 1994. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text.* Mourton de Gruyter, Berlin and New York.

[Klein and Simmons1963] S. Klein and R. Simmons. 1963. A computational approach to grammatical coding of english words. *Journal of Computing Machinery*, 10.

[Magerman1995] David M. Magerman. 1995. Statistical Decision-Tree Models for Parsing. In *Proceedings of the* $33^{rd}$ *Annual Meeting of the Association for Computational Linguistics.*

[McCord1993] Michael C. McCord. 1993. Heuristics for Broad-Coverage Natural Language Processing. In *DARPA Human Language Technology Workshop*, March, 1993.

[Mori and Nagao1995] Shinsuke Mori and Makoto Nagao. 1995. Parsing Without Grammar. In *Proceedings of the* $4^{th}$ *Annual International Workshop of Parsing Technologies*, Prague.

[Nagao1994] Makoto Nagao. 1994. Varieties of Heuristics in Sentence Processing. In *Current Issues in Natural Language Processing: In Honour of Don Walker*. Giardini with Kluwer.

[Schabes et al.1988] Yves Schabes, Anne Abeillé, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. In *Proceedings of the* $12^{th}$ *International Conference on Computational Linguistics (COLING'88)*, Budapest, Hungary, August.

[Schabes et al.1993] Y. Schabes, M. Roth, and R. Osborne. 1993. Parsing the Wall Street Journal with the Inside-Outside Algorithm. In *Proceedings of the European ACL.*

[Soong and Huang1990] Frank K. Soong and Eng-Fong Huang. 1990. Fast Tree-Trellis Search for Finding the N-Best Sentence Hypothesis in Continuous Speech Recognition. *Journal of Acoustic Society, AM.*, May.

[Vijay-Shanker and Joshi1991] K. Vijay-Shanker and Aravind K. Joshi. 1991. Unification Based Tree Adjoining Grammars. In J. Wedekind, editor, *Unification-based Grammars*. MIT Press, Cambridge, Massachusetts.

[Vijay-Shanker1987] K. Vijay-Shanker. 1987. *A Study of Tree Adjoining Grammars*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

# STOCHASTIC PARSE-TREE RECOGNITION BY A PUSHDOWN AUTOMATON

Frédéric TENDEAU
INRIA-Rocquencourt [1], BP 105, 78153 Le Chesnay CEDEX, FRANCE
E-mail : Frederic.Tendeau@inria.fr

**Abstract**

We present the stochastic generalization of what is usually called correctness theorems : we guarantee that the probabilities computed operationally by the parsing algorithms are the same as those defined denotationally on the trees and forests defined by the grammar.

The main idea of the paper is to precisely relate the parsing strategy with a parse-tree exploration strategy : a computational path of a parsing algorithm simply performs an exploration of a parse-tree for the input portion already parsed. This approach is applied in particular to Earley and Left-Corner parsing algorithms. Probability computations follow parsing operations : looping problems (in rule prediction and subtree recognition) are solved by introducing probability variables (which may not be immediately evaluated). Convergence is ensured by the syntactic construction that leads to stochastic equations systems, which are solved as soon as possible.

Our algorithms accept any (probabilistic) CF grammar. No restrictions are made such as prescribing normal form, proscribing empty rules or cyclic grammars.

*Keywords* : pushdown automaton, dynamic programming, stochastic context-free grammar.

## 1 Introduction.

Stochastic context-free grammars are extensively used in natural language parsing and have been recently applied to genome analysis. They define probabilities on rules, hence on derivations and finally on the language described. These probabilities may be computed to rank the different derivations of an ambiguous sentence or to estimate the probability of sentences with a given prefix. In general, the relevant literature does not give the formal framework that could permit to guarantee the correctness of computations. [Booth-Thompson 73] gives such a framework, to study the problem of adequacy between probabilities in the grammar and in the language.

Our aim is to present a Left Corner (LC) algorithm that recognizes stochastic parse forests and computes prefix and subtree probabilities, improving [Stolcke 93] which is based on Earley's algorithm. We also attempt to address more systematically the issue of the correctness of the stochastic parser w.r.t. the denotational definition of sentence probabilities by the grammar.

After the presentation of the formalism, we progress towards our goal step by step, developing each time one of the following directions :

**parsing strategy :** before the LC strategy, we describe Earley's, which is in fact the same strategy without grammar compilation. The parsing strategies are expressed in term of pushdown automata (PDA).

**decoration level :** before the probabilistic computations, we give a purely syntactic version.

**number of considered trees :** before considering shared forests, we present the problem on a single tree. The shared forest is produced by a dynamic programming interpretation of the PDA and is expressed in term of a pushdown transducer (PDT).

Analyzing the various combinations may be viewed as exploring the eight corners of a cube. This separation of issues allows a uniform presentation of all variants. It also makes the algorithms easier to understand, and to prove correct.

[1] this work was partially supported by the grant 95-B030 from the Centre National d'Études des Télécommunications.

## 2 Analysis Material.

### 2.1 Grammars.

**Definition 1.** A *context-free grammar* (CFG) is a 4-tuple $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where $\Sigma$ is the set of terminal symbols, $\mathcal{N}$ that of non-terminals ($\Sigma \cap \mathcal{N} = \emptyset$), $\mathcal{R}$ that of rules, and $S \in \mathcal{R}$ is the axiom. The set $\mathcal{V} = \Sigma \cup \mathcal{N}$ is called the vocabulary.

We use usual notations: early Latin lowercase ($a, b...$) for terminals, late ones ($x, y$) for terminal strings, early Latin uppercase alphabet (A,B...) for the non-terminals, late ones (X,Y) for vocabulary symbols, and early Greek letters ($\alpha, \beta...$) for the vocabulary strings.

The grammar rules are also called productions. They are noted $A \rightarrow \alpha$, with $A \in \mathcal{N}$ called the left hand side (LHS) and $\alpha \in \mathcal{V}^*$ called right hand side (RHS). A rule having A as LHS is an A-rule. The collection of CFGs is noted $\mathcal{G}$.

From now on we implicitly consider a CFG $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$.

### 2.2 Derivation and Language.

Usual definitions.

– The relation *derive* is $\Longrightarrow = \{(\alpha A \beta, \alpha\gamma\beta) \mid A \in \mathcal{N}, \alpha, \beta, \gamma \in \mathcal{V}^* \text{and } A \rightarrow \gamma \in \mathcal{R}\}$

The reflexive transitive closure is noted $\overset{*}{\Longrightarrow}$. Deriving $n$ times is noted $\overset{n}{\Longrightarrow}$.

– A *derivation* is a sequence $(\alpha_i)_{i \geq 0}$ of $\mathcal{V}^*$ s.t. for each $i$, $\alpha_i \Longrightarrow \alpha_{i+1}$.

By definition, for each $0 \leq i \leq j$, $\alpha_i \overset{*}{\Longrightarrow} \alpha_j$: each sub-sequence is a derivation. We say that $(\alpha_k)_{k \in [i,j]}$ is a derivation of $\alpha_i$ into $\alpha_j$. A derivation is *elementary* if it is a pair of which first element is a non-terminal: it is noted like a grammar rule.

– A non-terminal symbol A is *productive* iff exists a terminal string $x$ s.t. $A \overset{*}{\Longrightarrow} x$.

– A non-terminal symbol A is *accessible* iff $S \overset{*}{\Longrightarrow} \alpha A \beta$.

– A vocabulary string $\alpha \in \mathcal{V}^*$ is a *sentential form* iff $S \overset{*}{\Longrightarrow} \alpha$.

– A terminal chain $x \in \Sigma^*$ is a *sentence* iff $S \overset{*}{\Longrightarrow} x$. (a sentence is a sentential form)

– The *language defined by G* is $\mathcal{L}(G) = \{x \in \Sigma^* \mid S \overset{*}{\Longrightarrow} x\}$.

– The relation *leftmost derive* is $\underset{\ell}{\Longrightarrow} = \{(xA\omega, x\eta\omega) \mid xA\omega \Longrightarrow x\eta\omega\}$

*Leftmost derivations* (sequences) and *leftmost sentential forms* are naturally derived. The set of all leftmost derivations of $\alpha$ into $\beta$ is noted $\alpha \underset{\ell}{\rightsquigarrow} \beta$, and $\alpha \overset{n}{\underset{\ell}{\rightsquigarrow}} \beta = \{d \in \alpha \underset{\ell}{\rightsquigarrow} \beta \mid \text{length of } d \text{ is } n\}$.

– A *sentential form* $\alpha$ is *ambiguous* iff $S \underset{\ell}{\rightsquigarrow} \alpha$ contains at least two elements.

– A *grammar* G is *ambiguous* iff an ambiguous sentence can be derived from it.

**Proposition 2.** *Classical results.*

(*i*) If $A \overset{*}{\Longrightarrow} x$ then $A \overset{*}{\underset{\ell}{\Longrightarrow}} x$.

(*ii*) $\mathcal{L}(G) = \{x \in \Sigma^* \mid S \overset{*}{\underset{\ell}{\Longrightarrow}} x\}$.

**Definition 3.** The *set of leftmost derivations* is

$$\Delta_\ell = \{\delta_0, \delta_1\} \cup \bigcup_{\alpha, \beta \in \mathcal{V}^*} \alpha \underset{\ell}{\rightsquigarrow} \beta$$

where $\delta_1$ (the unit derivation) and $\delta_0$ (the inconsistent derivation) are defined w.r.t. the composition properties.

**Definition 4.** The *composition of leftmost derivations* is the function $\textcircled{\ell}$: $\Delta_\ell \times \Delta_\ell \longrightarrow \Delta_\ell$ s.t.

$\forall\, d \in \Delta_\ell$, $d \,\textcircled{\ell}\, \delta_1 = \delta_1 \,\textcircled{\ell}\, d = d$ and $d \,\textcircled{\ell}\, \delta_0 = \delta_0 \,\textcircled{\ell}\, d = \delta_0$.

$\forall\, (d)_{i \in [0,l]} \in \alpha \underset{\ell}{\rightsquigarrow} x\beta\gamma$ and $(d')_{i \in [0,l']} \in \beta \underset{\ell}{\rightsquigarrow} \nu$, $(d'')_{i \in [0,l'']} = d \,\textcircled{\ell}\, d' \in \alpha \underset{\ell}{\rightsquigarrow} x\nu\gamma$ is s.t. $l'' = l + l'$, $(d'')_{i \in [0,l]} = (d)_{i \in [0,l]}$, and for $i > l$, $d''_i = x d'_{i-l} \gamma$.

### 2.3 Trees and Forests.

We use the approach of [Lang 89] to define trees as grammars: $(\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{s})$ is a derivation tree from $(\Sigma, \mathcal{N}, \mathcal{R}, S)$.

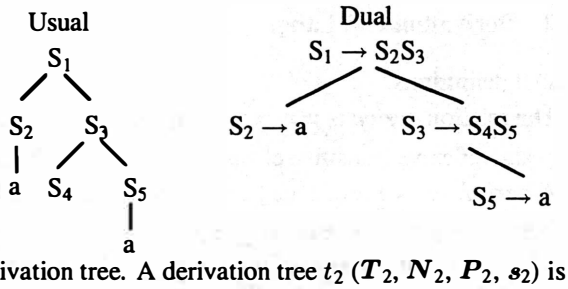
**Definition 5.** A *derivation tree* is the inconsistent tree $\tau_0$, or the unit tree $\tau_1$ or a CFG $(\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{s})$ verifying

1. for each $A \in \boldsymbol{N}$, A is accessible ; A is the LHS of at most one rule ; A appears at most once in at most one RHS.
2. $\boldsymbol{s}$ occurs in no RHS.
3. there exists a labelling function *lab*, from $(\boldsymbol{T} \cup \boldsymbol{N})^* \cup \boldsymbol{P}$ to $\mathcal{V}^* \cup \mathcal{R}$ s.t. $lab(\boldsymbol{N}) \subset \mathcal{N}$, $lab(\boldsymbol{P}) \subset \mathcal{R}$, $lab(a) = a$, $lab(\epsilon) = \epsilon$, $lab(X\alpha) = lab(X)lab(\alpha)$, $lab(A \rightarrow \alpha) = lab(A) \rightarrow lab(\alpha)$.

The set of derivation trees is $\mathcal{T}$. The vocabulary $\boldsymbol{V} = \boldsymbol{T} \cup \boldsymbol{N}$. Productions are called nodes, the $\boldsymbol{s}$-rule is called the root.

**Example 6.**

Given $\mathcal{R} = \{S \rightarrow SS, S \rightarrow a\}$, a possible derivation tree is s.t. $\boldsymbol{P} = \{S_1 \rightarrow S_2S_3, S_3 \rightarrow S_4S_5, S_2 \rightarrow a, S_5 \rightarrow a\}$ and $\forall i\ lab(S_i) = S$. This tree is shown as a dual form of the usual view.




**Definition 7.** Let $t_1$ $(\boldsymbol{T}_1, \boldsymbol{N}_1, \boldsymbol{P}_1, \boldsymbol{s}_1)$ be a derivation tree. A derivation tree $t_2$ $(\boldsymbol{T}_2, \boldsymbol{N}_2, \boldsymbol{P}_2, \boldsymbol{s}_2)$ is a *derivation subtree* of $t_1$ iff $\boldsymbol{P}_2 \subset \boldsymbol{P}_1$.

**Definition 8.** A parse tree $t$ is a *derivation tree* $(\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{s})$ of which non-terminals are all productive, such that $lab(\boldsymbol{s}) = S$.

$t$ is a parse tree of $x$ iff $\mathcal{L}(t) = \{x\}$, alternately: $t$ spans $x$.

During parsing, we need usual notion on trees (child, left and right) so we introduce orders :

– *parental or vertical order.* Given a derivation tree $t$, $\succeq \subset \boldsymbol{N} \times \boldsymbol{N}$ is the reflexive transitive closure of $\succ$ : $A \succ B$ iff $t$ has a rule $A \rightarrow \alpha B \beta$.

When $A \succeq B$, A is ancestor of B and B descendant of A — if $A \succ B$ then A is parent of B and B is a child of A.

– *horizontal order.* Let $t = (\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{s})$ be a tree, $\ll$ is a partial order defined on $\boldsymbol{N}$, and a relation on $\Sigma$: let X and Y be in $\boldsymbol{V}$, $X \ll Y$ iff $\boldsymbol{s} \stackrel{*}{\Longrightarrow} \alpha X \beta Y \gamma$.

"$X \ll Y$" is said "X *on the left of* Y".

**Definition 9.** The *composition of derivation trees* is the function $\otimes : \mathcal{T} \times \mathcal{T} \longrightarrow \mathcal{T}$ s.t.
$\forall\, t \in \mathcal{T}$, $t \otimes \tau_1 = \tau_1 \otimes t = t$ and $t \otimes \tau_0 = \tau_0 \otimes t = \tau_0$.
$\forall\, t_1 = (\boldsymbol{T}_1, \boldsymbol{N}_1, \boldsymbol{P}_1, \boldsymbol{s}_1)$ and $t_2 = (\boldsymbol{T}_2, \boldsymbol{N}_2, \boldsymbol{P}_2, \boldsymbol{s}_2)$, $t_1 \otimes t_2 = (\boldsymbol{T}_1 \cup \boldsymbol{T}_2, \boldsymbol{N}_1 \cup \boldsymbol{N}_2, \boldsymbol{P}_1 \cup \boldsymbol{P}_2, \boldsymbol{s}_1)$, if this grammar is a derivation tree and if $\boldsymbol{s}_2$ is the leftmost $t_1$ non-terminal without a child [2], else $t_1 \otimes t_2 = \tau_0$.

A derivation tree $(\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{s})$ is elementary iff $\boldsymbol{P}$ is a singleton. Derivations trees $\otimes$-composed with only elementary trees are called elementarily $\otimes$-composed. There exists a correspondence between leftmost derivations and elementary $\otimes$-composed derivation trees. From derivation trees to leftmost derivation, one has just to replace $\otimes$ by $\textcircled{\ell}$, and the nodes by their labels. From derivations to trees, $\textcircled{\ell}$ is replaced by $\otimes$ and a labelling function must be given.

#### 2.3.1 Ambiguity.

**Definition 10.** A *parse forest* is a set of parse trees.

Let $x$ be in $\mathcal{L}(G)$. The set of all the parse trees of $x$ is called the parse forest of $x$. The composition $\textcircled{\ell}$ is naturally extended on $\Delta_\ell$ subsets.

[2] this last condition is not necessary, *i.e.* $\otimes$ corresponds to the derivation, but we want $\otimes$ corresponds to the *leftmost* derivation.

### 2.4 Probabilistic Grammars and Forests.

Probabilities are defined on a set of events. What we call an event is the derivation of a non-terminal A using some grammar rule: an event requires a non-terminal A, we speak of an A-event. The set of all possible A-events is the set of ways to derive A, *i.e.* the set of A-rules. A probability is assigned to each rule: the probabilities over $\mathcal{R}$ are well defined iff the sum of all A-rules probability is 1. The probability domain is $\mathcal{D}(P)$.

**Definition 11.** A *probabilistic, or stochastic, CFG* is a 5-tuple $(\Sigma, \mathcal{N}, \mathcal{R}, S, P)$, s.t. $(\Sigma, \mathcal{N}, \mathcal{R}, S)$ is a CFG in which all non terminals are accessible, and $P$ is a function: $\mathcal{R} \longrightarrow \mathcal{D}(P)$, that associates with each production $A \rightarrow \omega$ a probability $P(A \rightarrow \omega)$, and s.t. for each $A \in \mathcal{N}$, $\sum_{A \rightarrow \alpha \in \mathcal{R}} P(A \rightarrow \alpha) = 1$.

$P$ is extended over derivations. A leftmost derivation is either $\delta_0$, or $\delta_1$, or an elementary derivation, or a composition of derivations: $P(\delta_0) = 0$ ; $P(\delta_1) = 1$ ; $P(A \overset{1}{\underset{\ell}{\rightsquigarrow}} \alpha) = P(A \rightarrow \alpha)$ ; for $d_1, d_2 \in \Delta_\ell$ s.t. $d_1 \textcircled{\ell} d_2 \neq \delta_0$, $P(d_1 \textcircled{\ell} d_2) = P(d_1) \times P(d_2)$.

$P(d)$ is the probability that an arbitrary derivation d', differing from a prefix of d by the last element of d', or equal to d, is actually d.

The equivalence between elementarily $\otimes$-composed derivation trees and leftmost derivations gives: for $t = (T, N, P, s)$, a derivation tree, $P(t) = \prod_{r \in P} P(lab(r))$. This is operationally used in our stochastic computations.

$P$ is extended on sets of derivations. For $E \subset \Delta_\ell$, $P(E) = \sum_{d \in E} P(d)$.

Application to leftmost sentential forms: for $\alpha$ s.t. $S \overset{*}{\underset{\ell}{\Longrightarrow}} \alpha$, $P(\alpha) = P(S \overset{*}{\underset{\ell}{\rightsquigarrow}} \alpha)$.

**Definition 12.** *Consistency condition.* A probabilistic grammar G is consistent iff

$$\lim_{n \to \infty} \sum_{x A \beta \text{ s.t. } S \overset{n}{\underset{\ell}{\Longrightarrow}} x A \beta} P(x A \beta) = 0$$

**Proposition 13.** If G is consistent then $P(\mathcal{L}(G)) = 1$.

The proof is in [Booth-Thompson 73], the idea is : starting from all the S-rules (probability = 1) and, for each, making all possible choices to derive the leftmost non-terminal (say A) at each step (choosing all the A-rules) keeps the sum of probabilities of all leftmost sentential forms equal to 1. When n grows to infinity there are more and more sentences but $P(\mathcal{L}(G)) = 1$ iff the sentential forms all tend to be sentences, *i.e.* the probability of the set of sentential forms that still contain a non-terminal tends to 0.

From now on our probabilistic grammars will be assumed consistent.

Corollary: this result states that the set of leftmost derivations from S to terminal strings can be considered as a set of independent events. The accessibility of each non terminal gives $(i)$ for each $\alpha \in \mathcal{V}^*$, $\sum_x P(\alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x) = 1$. $(ii)$ given a leftmost sentential form $\alpha$, $P(S \overset{*}{\underset{\ell}{\rightsquigarrow}} \alpha) = P(S \overset{*}{\underset{\ell}{\rightsquigarrow}} \alpha) \sum_x P(\alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x)$ $= \sum_x P(S \overset{*}{\underset{\ell}{\rightsquigarrow}} \alpha \textcircled{\ell} \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x)$ — the probability of the leftmost derivations from S to terminal strings, that contain $\alpha$: $S \overset{*}{\underset{\ell}{\rightsquigarrow}} \alpha \textcircled{\ell} \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x \subset S \overset{*}{\underset{\ell}{\rightsquigarrow}} x$.

### 2.5 The Dotted Rules.

The dotted rules are used to describe the parsing process. We show they are a mean to describe positions in a parse-tree.

**Definition 14.** A *dotted rule* is a couple $(A \rightarrow \alpha, index)$, usually noted $A \rightarrow \omega \bullet \delta$ (with $|\omega| = index$), s.t. $0 \leq index \leq |\omega\delta|$.

*index* indicates how many symbols have been recognized in RHS.

$A \rightarrow \bullet\alpha$ and $A \rightarrow \alpha\bullet$ are respectively an initial and a final (or complete) dotted rule.

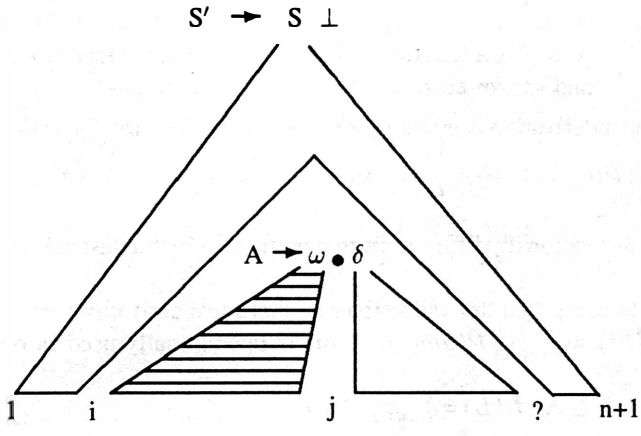
### 2.6 Parsing Conventions.

1. The grammar is augmented: a new symbol is added to $\mathcal{N}$: S' ; a new symbol is added to $\Sigma$: $\perp$ ; a new rule is added to $\mathcal{R}$: $S' \rightarrow S\perp$, s.t. $P(S' \rightarrow S\perp) = 1$.
2. For $w \in \Sigma^*$, $|w| = n$ and $1 \leq i \leq j \leq n$, $w_{i..j}$ stands for the string $w$ index range $[i, j[$.

3. From now on we will consider an input string: $x_{1\cdots n+1}$, its length is n, its first token, $x_1$, and its last, $x_n$. The token $x_{n+1}$ is $\perp$, which is not recognized by the non augmented grammar.

### 2.7 The Earley-Items.

**Definition 15.** An *Earley-item* is a triple noted $[A \rightarrow \omega \bullet \delta, i, j]$ s.t. $\omega \overset{*}{\Longrightarrow} x_{i\cdots j}$.




$[A \rightarrow \omega \bullet \delta, i, j]$ can be seen both as a sub-parse-tree and as a position on the node that dominates it: the subtree is the one that spans $x_{i\cdots j}$ (the one with horizontal lines), its root is labelled by $A \rightarrow \omega\delta$ and the dot • points the position between $\omega$ and $\delta$. The indexes are between the leaves (the question mark states that the index of the position is unknown).

Initial and final dotted rules correspond to initial and final positions on a node.
The collection of Earley-items on G is $\mathcal{I}(G)$.

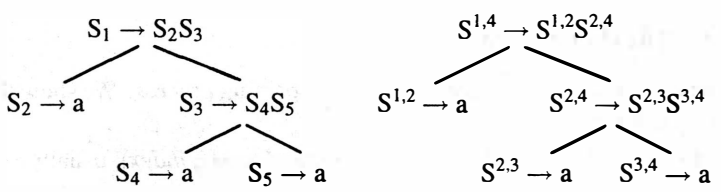
### 2.8 Shared Forest.

A function $\mathcal{G}$ will serve to define shared forests, using the Earley-items associated with each node of the parse trees. The set of natural integers is noted $\mathbb{N}$.

**Definition 16.** The *renaming parse-tree function* $\mathcal{G} : \mathcal{T} \longrightarrow \mathcal{G}$.
Consider a parse-tree $t = (T, N, P, s)$, and the sets $\mathcal{M} = \mathcal{N} \times \mathbb{N} \times \mathbb{N}$ and $\mathcal{W} = \mathcal{M} \cup \Sigma$. [3]
$\mathcal{G}(t) = (\Sigma, \mathcal{N}', \mathcal{R}', S_o)$, with $\mathcal{N}' \subset \mathcal{M}$ and $\mathcal{R}' \subset \mathcal{M} \times \mathcal{W}^*$.
$\mathcal{R}'$: let $r$ be in $P$ s.t. $lab(r) = A \rightarrow X_1, \ldots, X_k \in \mathcal{R}$. Let us consider the sequence of Earley-items on this node: $( [A \rightarrow X_1 \ldots X_m \bullet X_{m+1} \ldots X_k, i, j_m] )_{m \in [0,k]}$, (recall that $j_0 = i$).
Then $A^{i,j_k} \rightarrow Y_1, \ldots, Y_k \in \mathcal{R}'$ is built s.t. if $X_m \in \Sigma$ then $Y_m = X_m$, else $Y_m = X_m{}^{j_{m-1}, j_m}$.
$\mathcal{N}'$ elements are inferred from $\mathcal{R}'$.

A renamed parse-tree is no longer a parse-tree because a non-terminal can appear more than once in RHS, due to $A \overset{+}{\Longrightarrow} A$ cycles: note that two parse trees differing only by such a cycle have the same $\mathcal{G}$ image. However, the other conditions are respected. *e.g.* G' spans $x$ (by definition of Earley-items) and the labelling function is s.t. $lab(A^{i,j}) = A$.

**Example 17.**
Opposite, we show the derivation tree of example 6., completed with a new rule $(S_4 \rightarrow a)$ to make a parse tree $t$, and next, $\mathcal{G}(t)$.




**Definition 18.** *Union of grammars.* $\cup : \mathcal{G} \times \mathcal{G} \longrightarrow \mathcal{G}$
$(\Sigma, \mathcal{N}_1, \mathcal{R}_1, S_o) \cup (\Sigma, \mathcal{N}_2, \mathcal{R}_2, S_o) = (\Sigma, \mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{R}_1 \cup \mathcal{R}_2, S_o)$.

**Definition 19.** The *shared forest* of a sentence $x \in \mathcal{L}(G)$ is $\bigcup_{t \text{ parse-tree of } x} \mathcal{G}(t)$.

The number of parse trees for $x$ may be infinite. However the sets of non-terminals and rules in the infinite union are all finite because the number of Earley-items for $x$ is finite. Therefore the shared forest is a grammar.

[3] $\mathcal{W}$ will be used while describing dynamic programming interpretations.

In general, this grammar is not a parse-tree because a non-terminal can be in different LHSs (due to ambiguity and causing subtree sharing) or can appear several times in RHS (causing context sharing, see [Lang 89]).

In the shared forest context, the Earley-items $[A \rightarrow \omega \bullet \delta, i, j]$ can be seen as the sub-(shared) forest that spans $x_{i..j}$ and has a root labelled by $A \rightarrow \omega\delta$. It corresponds to this position in all the parse-trees.

### 2.9 Probabilities and Earley-Item.

**Definition 20.** The 5-tuple $< A \rightarrow \alpha \bullet \beta, i, j, \mathbb{P}, \mathbb{R} >$ is a *probabilistic item* iff $I = [A \rightarrow \alpha \bullet \beta, i, j]$ is an Earley-item s.t. $p_r(I) = \mathbb{R}$ and $p_p(I) = \mathbb{P}$, where $p_r$ and $p_p$ are defined below.

#### 2.9.1 Recognition Probability.

As a corollary of the grammar consistency (proposition 13.), one proves that $p_r$ is the probability of all the sub-parse trees rooted in a node labelled by $A \rightarrow \alpha\beta$, s.t. $\alpha$ spans $x_{i..j}$.

We define $p_r([A \rightarrow \alpha \bullet \beta, i, j]) = p(A \rightarrow \alpha\beta \,@\, \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j})$

#### 2.9.2 Prediction Probability.

As above, one proves that $p_p$ is the probability of an Earley-item, seen as a set of leftmost derivation trees corresponding to some single sentential form.

$p_p([A \rightarrow \alpha \bullet \beta, i, j]) = p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega \,@\, A \rightarrow \alpha\beta \,@\, \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j})$

**Property 21.** $p_p([A \rightarrow \alpha \bullet \beta, i, j]) = p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega).\ \times p_r([A \rightarrow \alpha \bullet \beta, i, j])$

Proof: $p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega \,@\, A \rightarrow \alpha\beta \,@\, \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j}) = p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega) \times p(A \rightarrow \alpha\beta \,@\, \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j})$.

**Property 22.** $p_p([S' \rightarrow S^{1,n+1} \bullet \bot, 1, n+1]) = p_r([S' \rightarrow S^{1,n+1} \bullet \bot, 1, n+1]) = p(x_{1..n+1})$.

Proof: $p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..1} S') = p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} S') = p(\delta_1) = 1$

therefore, using the previous property, $p_p([S' \rightarrow S^{1,n+1} \bullet \bot, 1, n+1]) = p_r([S' \rightarrow S^{1,n+1} \bullet \bot, 1, n+1])$.

Now, $p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} S\bot) = 1$ and $p(S \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..n+1}) = p(x_{1..n+1})$ give the second equality.

## 3 Pushdown Automata and their Interpretation.

We distinguish a (non-deterministic) PDA, that describes the parsing strategy, from its dynamic programming interpretation, presented in a pushdown transducer formalism.

### 3.1 Pushdown Automaton.

**Definition 23.** A PDA is a 4-tuple $(\Sigma_\pi, \pi^i, \pi^f, T)$ s.t. $\Sigma_\pi$ is the stack alphabet ; $\pi^i$ is the initial stack ; $\pi^f$ is the final stack ; $T$ is the set of transitions on the stack.

Stacks are noted $[\text{head}|\text{tail}]$.

Transitions, so : top1, input $\mapsto$ top2, where

– top1 is the stack 1 or 2 top element(s) before the transition ;

– input is the current token of the input : noted $a$ if it is scanned, $(a)$ if it is only read and () if the transition is input independent.

– top2 is the stack 1 or 2 top element(s) after the transition.

Without considering the input, there are four kinds of transitions :

**SWAP** : $e_1, input \mapsto e'_1$. The top element is consulted and popped, another is pushed.

**PUSH** : $e_2, input \mapsto e_1, e_2$. The top element is consulted and another is pushed.

**POP** : $e_1, e_2, input \mapsto e'_2$. The top element and the one beneath are consulted and popped, another is pushed.

**SWAP2** : $e_1, e_2, input \mapsto e'_1, e_2$. The top element and the one beneath are consulted, only the first is popped, another is pushed.

A complete computation ends with the final stack.

### 3.2 The Interpretations.

An interpretation is an execution model of the automaton which manages the non-determinism with dynamic programming technic. We use [VdlC 93] terminology and the so-called $S^1$ interpretation.

$S^1$ destroys the stack structure but stores the elements in a set called 1-items set : $\mathcal{E}$. The transitions re-build some parts of the stack.

For example the 1-items $[a]$ and $[b]$, permit to build 2 stacks : $p = [a, b]$ and $q = [b, a]$. But a POP requiring the top (a,b) considers only $p$.

$S^1$ interpretation is very compact, and allows very good sharing between the stacks. The form of the elements (Earley-items) will guarantee the correctness of the execution.

**Definition 23.** A *pushdown transducer* is a 5-tuple $(\Sigma_\pi, \pi^i, \pi^f, T, \Sigma_\sigma)$ s.t. $(\Sigma_\pi, \pi^i, \pi^f, T)$ is a PDA and $\Sigma_\sigma$ is the output alphabet.

PDTs will be used for dynamic programming interpretations of PDA. The stack of our PDTs will be the set of 1-items $\mathcal{E}$. The transitions are the same than for PDA, excepted that they have an extra parameter for the output. They have the form : top1, input $\mapsto$ top2, output, with $\varepsilon$ meaning no output. Stacks elements on the left (top1) are chosen in $\mathcal{E}$, and top2 is a single element, added to $\mathcal{E}$. This makes PUSH look like SWAP, and POP like SWAP2.

The execution starts with the top element of $\pi^i$ in $\mathcal{E}$, it loops until no more 1-item can be added to $\mathcal{E}$ or no more token is to be read.

The execution output is the shared-parse-forest of the input.

## 4 Earley.

In [Earley 70] a CFG recognizer and parser are given. The recognizer uses dynamic programming, what leads to a cubic time and space complexity. The parsing strategy it uses is not separated from its dynamic programming interpretation. We do such a separation.

### 4.1 Earley's PDA.

From the augmented grammar $(\Sigma \cup \{\bot\}, \mathcal{N} \cup \{S'\}, \mathcal{R} \cup \{S' \rightarrow S\bot\}, S')$ and $x_{1..n+1}$, a PDA $(\Sigma_\pi, \pi^i, \pi^f, T)$ is built s.t.

**Stack alphabet :** $\Sigma_\pi = \mathcal{I}(G) \cup \{\bot\}$.

**Initial stack :** $\pi^i = \left[[S' \rightarrow \bullet S\bot, 1, 1] \middle| \bot\right]$

**Final stack :** $\pi^f = \left[[S' \rightarrow S^{1,n+1} \bullet \bot, 1, n+1] \middle| \bot\right]$

**Transitions :** $T$ = PUSH $\cup$ SWAP $\cup$ POP.

PUSH : set of *prediction* transitions, of the form
$[A \rightarrow \alpha \bullet B\beta, i, j], () \mapsto [B \rightarrow \bullet\gamma, j, j], [A \rightarrow \alpha \bullet B\beta, i, j]$

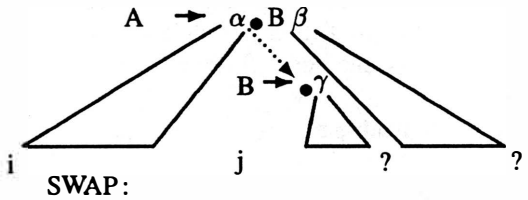
SWAP : set of *scan* transitions, of the form
$[A \rightarrow \alpha \bullet a\beta, i, j], a \mapsto [A \rightarrow \alpha a \bullet \beta, i, j+1]$

POP : set of *completion* transitions, of the form
$[A \rightarrow \alpha \bullet, i, j], [B \rightarrow \eta \bullet A\beta, k, i], () \mapsto [B \rightarrow \eta A \bullet \beta, k, j]$

### 4.2 Proof Indications.

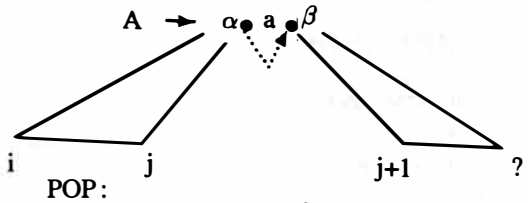
For correction: first, the initial Earley-item corresponds to top-leftmost position on any parse-tree. Afterwards, assuming we have a deterministic complete computation, the figures below show that each computation step consists in a left-to-right depth-first exploration (LRDFE) step of a $x_{1..n+1}$ parse-tree.

PUSH:



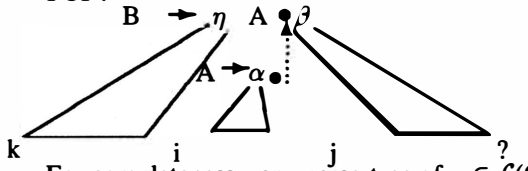

The current position is on a node labelled by $A \rightarrow \alpha B \beta$, after $\alpha$ and in front of B, the computation pushes the initial position on a node labeled by $B \rightarrow \gamma$. It corresponds to going down to the current node next child to the right: a LRDFE step.

SWAP:




Here, the computation corresponds to staying in the same node and moving to the next position from left to right, a token being recognized.

POP:




The current node N is labeled by $A \rightarrow \alpha$ and the position is final, a whole sub-parse tree has been recognized. The computation corresponds to the come back to its parent, on N right.

For completeness: any parse-tree of $x \in \mathcal{L}(G)$ can be explored LRDF. Each movement is either a left to right step on a node, or a descent to a child, or a come back up to a parent, to which corresponds respectively a SWAP, PUSH or POP transition, and only one.

Note that the extended version of this paper, with all the complete proofs, is to appear in [Tendeau 95].

### 4.3 Earley-Item and Left Derivation.

A LRDFE, with an action executed on each node the first time it is met, is a prefix LRDFE. Considering the output of the rule that labels a node as an action, a prefix LRDFE of a given parse-tree yields a left derivations of that tree.

We used to see Earley-items as positions in a parse-tree, the Earley parsing strategy permits to consider them also as a step within a (prefix) LRDFE, *i.e.* within a left derivation process: $I = [A \rightarrow \alpha \bullet \beta, i, j]$ identifies $S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega \; \textcircled{\ell} A \rightarrow \alpha\beta \; \textcircled{\ell} \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j}$. [4]

Hence $p_p(I)$ is the probability of the left derivation trees of $S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega \; \textcircled{\ell} A \rightarrow \alpha\beta \; \textcircled{\ell} \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j}$.

This permits to present the intuition of the proofs: a parsing algorithm performs a LRDFE and the probability of explored (sub-)trees are incrementally computed.

### 4.4 Probabilistic PDA.

Grammar rules are now decorated by probabilities. Trees and automata are also decorated.

**Stack alphabet** $\mathcal{I}(G) \times \mathcal{D}(p) \times \mathcal{D}(p) \cup \{\bot\}$.

**Initial stack** $[\langle S' \rightarrow \bullet S \bot, 1, 1, 1, 1 \rangle | \bot]$

**Final stack** $[\langle S' \rightarrow S \bullet \bot, 1, n+1, \Pi, \Pi \rangle | \bot]$

**PUSH**

$\langle A \rightarrow \alpha \bullet B\delta, i, j, \mathbf{P}, \mathbf{R} \rangle, () \mapsto \langle B \rightarrow \bullet\eta, j, j, \mathbf{P}p(B \rightarrow \eta), P(B \rightarrow \eta) \rangle, \langle A \rightarrow \alpha \bullet B\delta, i, j, \mathbf{P}, \mathbf{R} \rangle$

**SWAP**

$\langle A \rightarrow \alpha \bullet a\delta, i, j, \mathbf{P}, \mathbf{R} \rangle, a \mapsto \langle A \rightarrow \alpha a \bullet \delta, i, j+1, \mathbf{P}, \mathbf{R} \rangle$

[4] when considering a deterministic computation this set is reduced to a singleton.

**POP**

$$<A\rightarrow\alpha\bullet, i, j, \mathbf{P}'', \mathbf{R}''>, <B\rightarrow\eta\bullet A\beta, k, i, \mathbf{P}, \mathbf{R}>, () \mapsto <B\rightarrow\eta A\bullet\beta, k, j, \mathbf{PR}'', \mathbf{RR}''>$$

### 4.5 Proof indications.

SWAP : the position progresses but in the same node, hence the probabilities remain unchanged.

PUSH : a node N, that has never been visited, is reached. The prediction probability is multiplied by $P(lab(N))$. The subtree identified by the pushed item has only one node, N, then $P(lab(N))$ is its probability.

POP : I,J $\mapsto$ K. K identifies J derivation tree composed with I sub-parse tree. Then K probabilities are J ones multiplied by $P_p(I)$.

## 5 Dynamic Programming Interpretation of Earley's Automata.

To produce the shared forest, we rename (using $g$) the sub-parse trees incrementally. When a non-terminal $N \in \mathcal{N}$ is shifted in a completion, it is immediately renamed by $N' \in \mathcal{M}$ : the following Earley-items $[A\rightarrow\alpha\bullet\beta, i, j]$ are s.t. $\alpha \in \mathcal{W}^*$, and $\beta \in \mathcal{V}^*$, their set is $\mathcal{I}(G)^I$.

### 5.1 Earley PDA Interpretation.

$\mathcal{E}$ domain : $\mathcal{I}(G)^I$

Initial stack,: $\left[[S'\rightarrow\bullet S\perp, 1, 1]\middle|\perp\right]$

Final stack,: $\left[[S'\rightarrow S^{1,n+1}\bullet\perp, 1, n+1]\middle|\blacktriangle\right]$

Transitions :

PUSH : $[A\rightarrow\alpha\bullet B\beta, i, j], () \mapsto [B\rightarrow\bullet\eta, j, j], \varepsilon$

SWAP : $[A\rightarrow\alpha\bullet a\beta, i, j], a \mapsto [A\rightarrow\alpha a\bullet\beta, i, j+1], \varepsilon$

POP : $[A\rightarrow\alpha\bullet, i, j], [B\rightarrow\eta\bullet A\beta, k, i], () \mapsto [B\rightarrow\eta A^{i,j}\bullet\beta, k, j], A^{i,j}\rightarrow\alpha$

### 5.2 Earley PPDA Interpretation.

#### 5.2.1 Synchronization Problem.

The absence of probabilities hides a problem that raises when computing them : as soon as a node is recognized, its parent can pursue its recognition, but computing the parent probabilities requires the probabilities of all its ambiguous children.

To continue not to take care of the computation order, symbolic probabilities can be introduced (representing probabilities of ambiguous children). The system is stable at the end of a complete computation, and can be solved then. But this would forbid to base a parsing decision on probabilities.

To permit this, we choose a intermediate solution : $\mathcal{E}$ is stratified and its computation is synchronized with the tokens. $\mathcal{E} = \bigcup_{j\in[0,n]} \mathcal{E}_j$, with $\mathcal{E}_j$ being the set of the Earley-items having j as second index. The level j+1 is computed after the level j is completed.

#### 5.2.2 The PPDT Computation Process.

The synchronization splits the transition set in two parts : the intra-level transitions (PUSH and POP) and the inter-level ones (SWAP). The computation process is slightly modified.

Instead of having a single loop on the transitions set, there are two : one for inter-level (PUSH-POP), one for intra-level (SWAP). Each is executed until no new 1-items are produced and then the other starts a new cycle, until no transition is applicable or $\perp$ is reached.

PUSH and POP are applied as usual : the 1-items on the left (of the transition) are chosen in $\mathcal{E}$ and the one on the right is added to $\mathcal{E}$.

For SWAP, there is a difference : the 1-item on the right is added in set : $\mathcal{E}_{j+1}$, while $\mathcal{E} = \bigcup_{k=0}^{k=j} \mathcal{E}_k$ ; the 1-items on the left are chosen in $\mathcal{E}$.

When the inter-level loop does not add any new 1-items in $\mathcal{E}_{j+1}$, j is incremented.

### 5.2.3 Symbolic Probabilities.

$p(\bullet B_j)$ is introduced because the initial Earley-items with a B in LHS are produced by all the Earley-items having a dot in front of B in level j : there is such a symbol for each B and each j.

$$p(\bullet B_j) = \sum_{<A\to\alpha\bullet B\beta,i,j,\mathbf{P},\mathbf{R}>\in\mathcal{E}} \mathbf{P}$$

$p(A_{ij}\bullet)$ is introduced because a subtree rooted in $A^{i,j}$ and spanning $x_{i..j}$ may be produced more than once. Such a symbol may exist for each (A, i, j).

$$p(A_{ij}\bullet) = \sum_{<A\to\alpha\bullet,i,j,\mathbf{P},\mathbf{R}>\in\mathcal{E}} \mathbf{R}$$

By definition, these variables can be evaluated if the level j is complete, they form an invertible linear system. This is easy to prove with G consistency : let N be the vector of all non-terminals: $N_l \in \mathcal{N}$. The $p(\bullet B_j)$ solving problem amounts to finding the $p(\bullet N)$ vector s.t.

$$p(\bullet N)_l = \sum_{<N_r\to\alpha\bullet N_l\beta,i,j,\mathbf{P},\mathbf{R}>\in\mathcal{E}} \mathbf{P}$$

$$= \sum_{<N_r\to\alpha\bullet N_l\beta,i,j,\mathbf{P},\mathbf{R}>\in\mathcal{E},s.t.i\neq j} \mathbf{P} + \sum_{<N_r\to\alpha\bullet N_l\beta,j,j,\mathbf{P},\mathbf{R}>\in\mathcal{E}} \mathbf{P}$$

$= K_l + \sum_r L_{l,r}$ where $K_l$ is the constant vector $K$ $l^{\text{th}}$ element and
$L_{l,r} = p_p([N_r\to\alpha\bullet N_l\beta, j, j]) = p_p([N_r\to\bullet\alpha N_l\beta, j, j])\, p(\alpha \overset{}{\rightsquigarrow}_l \epsilon) = p(\bullet N)_r\, p(N_r\to\alpha N_l\beta)\, p(\alpha \overset{}{\rightsquigarrow}_l \epsilon)$.

Thus $p(\bullet N) = M\, p(\bullet N) + K$, where $M_{l,r} = p(N_r \overset{1}{\rightsquigarrow}_l \alpha N_l\beta)\, p(\alpha \overset{}{\rightsquigarrow}_l \epsilon)$.

The equality $(M^p)_{l,r} = p(N_r \overset{p}{\rightsquigarrow}_l \alpha N_l\beta)\, p(\alpha \overset{}{\rightsquigarrow}_l \epsilon)$ is easy to verify. Hence, the consistency condition leads to: $\lim_{p\to\infty} (M^p)_{l,r} = 0$, therefore, there exists a $\kappa$ s.t. for each $q \geq \kappa$, $\sum_l (M^q)_{l,r} < 1$, then $||M^q||_1 < 1$ [5]. Hence, the series $\sum_{i=0}^{\infty} M^i$ converges, *i.e.* $(I - M)$ is invertible, $p(\bullet N) = (I - M)^{-1} K$ and $(I - M)^{-1}_{l,r} = p(N_r \overset{}{\rightsquigarrow}_l \alpha N_l\beta)\, p(\alpha \overset{}{\rightsquigarrow}_l \epsilon) = p(N_r \overset{}{\rightsquigarrow}_l N_l\beta)$.

The $p(A_{ij}\bullet)$ system is similar, the corresponding matrix $Q$ being s.t. $(I - Q)^{-1}_{l,r} = p(N_r \overset{}{\rightsquigarrow}_l N_l)$.

*Important note*: $M$ and $Q$ matrices can be statically inverted since they depend only on G.

### 5.2.4 Description of the PPDT.

$\mathcal{E}$ domain : $\mathcal{I}(G)^I \times \mathcal{D}(p) \times \mathcal{D}(p)$.

Initial stack : $[<S'\to\bullet S\perp, 1, 1, 1, 1> \mid \perp]$

Final stack : $[<S'\to S\bullet\perp, 1, n+1, \Pi, \Pi> \mid \perp]$

Intra-level :

**PUSH**

$<A\to\alpha\bullet B\delta, i, j, \mathbf{P}, \mathbf{R}>, () \mapsto <B\to\bullet\eta, j, j, p(\bullet B_j)\, P(B\to\eta), P(B\to\eta)>, \varepsilon$

**POP**

$<A\to\alpha\bullet, i, j, \mathbf{P}'', \mathbf{R}''>, <B\to\eta\bullet A\beta, k, i, \mathbf{P}, \mathbf{R}>, () \mapsto <B\to\eta A^{i,j}\bullet\beta, k, j, \mathbf{P}p(A_{ij}\bullet), \mathbf{R}p(A_{ij}\bullet)>, (A^{i,j}\to\alpha, \mathbf{R}'')$

Inter-level : **SWAP**

$<A\to\alpha\bullet a\beta, i, j, \mathbf{P}, \mathbf{R}>, a \mapsto <A\to\alpha a\bullet\beta, i, j+1, \mathbf{P}, \mathbf{R}>, \varepsilon$

[5] $||M||_1$ is then 1-norm defined by $\max_r \sum_l |M_{l,r}|$.

### 5.3 Proof indications.

PUSH prediction: a node N, that has never been visited, is reached. Contrarily to the deterministic computation, N can be reached from several positions: all the ones that have (at level j) a dot in front of B. That is why $p_p$ is $p(lab(N))$ multiplied by $p(\bullet B_j)$.

POP: I,J $\mapsto$ K. K probabilities are still based on J ones, yet they do not only depend on $p_r(I)$ but on the sum of the probabilities of all the subtrees rooted in $A^{i,j}$: $p(A_{ij}\bullet)$.

## 6 Left Corner.

Earley's algorithm is completely dynamic: nothing is statically produced using the grammar before parsing. [Stolcke 93] gives a stochastic parse-tree recognition algorithm based on Earley's, but which avoids the dynamic resolution of our systems by solving them statically. After [Leermakers 89], we propose to integrate this optimization in a process which performs syntactic static computation: Left Corner (LC).

The LC strategy, seen as a tree exploration, is the same as Earley's. But all the prediction calls are statically computed: a finite automaton is compiled using the grammar.

### 6.1 Construction of the Underlying Finite Automaton.

Considering the augmented grammar $(\Sigma\cup\{\bot\}, \mathcal{N}\cup\{S'\}, \mathcal{R}\cup\{S'\rightarrow S\bot\}, S')$.

Given a non-initial dotted rule, we want all the initial dotted rules produced by an Earley prediction. For this we use the LC relation.

**Definition 24.** *Left Corner relation.*
$\angle = \{(A, B) \in \mathcal{N}^2 \mid B\rightarrow A\alpha \in \mathcal{R}\}$

**Definition 25.** *Left Corner state.*
s is a LC state iff $s = \{k(s)\} \cup nk(s)$ s.t. if $s = s_0$, the initial state, then $k(s_0) = S'\rightarrow\bullet S\bot$, else k(s) is a non initial dotted rule, and $nk(s) = \{C\rightarrow\bullet\alpha \mid k(s) = A\rightarrow X\beta\bullet B\gamma$ and $C \angle^* B\}$.
k(s) is the kernel of s, nk(s), its non-kernel set.

The set of LC states is $\mathcal{S}^{comp}$. It contains a single final state: $s_f = \{S'\rightarrow S\bullet\bot\}$.

Transitions are summarized in 3 tables. $s \in \mathcal{S}^{comp}$, $B \in \mathcal{N}$.

1. $GOTO_k(s, B) = r$ s.t. $k(s) = A\rightarrow\omega\bullet B\delta$ and $k(r) = A\rightarrow\omega B\bullet\delta$.
2. $GOTO_{nk}(s, B) = \{r \in \mathcal{S}^{comp} \mid A\rightarrow\bullet B\omega \in s$ and $k(r) = A\rightarrow B\bullet\omega\}$.
3. $ACTION(s, a) \in \mathcal{A}$, with $s \in \mathcal{S}^{comp}$, $a \in \Sigma$ and $\mathcal{A}$ is the set of actions.
   - ksc(r) (kernel scan) iff $k(s) = A\rightarrow\omega\bullet a\delta$ and $k(r) = A\rightarrow\omega a\bullet\delta$.
   - nksc(r) (non-kernel scan) iff $A\rightarrow\bullet a\omega \in s$ and $k(r) = A\rightarrow a\bullet\omega$
   - red($A\rightarrow\omega$) (reduction of $A\rightarrow\omega$) iff $A\rightarrow\omega\bullet \in s$.

By definition, a LC state contains all the possible paths from the kernel position to the leaves of all the trees where the kernel position can appear. During the parse, when moving from, say, s to r, one path is pursued among all the potential paths in s: until a transition is executed from a state, its real position is given by its kernel, the non-kernel set contains virtual positions.

The LC PDA does not need any prediction transition since it is already within the LC states, but scan and completion transitions have to manage the virtual paths becoming real: this explains the splitting kernel/non-kernel, corresponding to Earley-like/prediction management.

### 6.2 The LC PDA.

As a state is associated to the dotted rule in its kernel, (s, i, j) is associated to the corresponding Earley-item.

**Stack alphabet** $\mathcal{S}^{comp}\times[0, n]\times[0, n]\cup\{\bot\}$.

**Initial stack** : $[(s_0, 1, 1)|\bot]$

**Final stack** : $[(s_f, 1, n+1)|\bot]$

**Scans** :

- KERNEL SCAN.
  if ksc(r) $\in$ ACTION(s, a), apply
  SWAP (s, i, j), a $\mapsto$ (r, i, j+1).
- NON-KERNEL SCAN.
  if nksc(r) $\in$ ACTION(s, a), apply
  PUSH (s, i, j), a $\mapsto$ (r, j, j+1), (s, i, j).

**Reductions** :

- KERNEL REDUCE.
  - if red(A$\rightarrow$X$\omega$) $\in$ ACTION(s, a) and r $\in$ GOTO$_k$(s', A), apply
    POP (s, i, j), (s', k, i), (a) $\mapsto$ (r, k, j).
  - if red(A$\rightarrow\epsilon$) $\in$ ACTION(s, a) and r $\in$ GOTO$_k$(s, A), apply
    SWAP (s, i, j), (a) $\mapsto$ (r, i, j).
- NON-KERNEL REDUCE.
  - red(A$\rightarrow$X$\omega$) $\in$ ACTION(s, a) and r $\in$ GOTO$_{nk}$(s', A), apply
    SWAP2 (s, i, j), (s', k, i), (a) $\mapsto$ (r, i, j), (s', k, i).
  - if red(A$\rightarrow\epsilon$) $\in$ ACTION(s, a) and r $\in$ GOTO$_{nk}$(s, A), apply
    PUSH (s, i, j), (a) $\mapsto$ (r, j, j), (s, i, j).

### 6.3 Proof indications.

Replacing states by their kernels make the kernel transitions appear like in Earley PDA.

Non-kernel transitions correspond to taking in account the static predictions while shifting a symbol — terminal for scans, non-terminal for reductions. Each non-kernel transition nk-T deals with the same action than the kernel one k-T, but manages the realization of the prediction: nk-T shifts a symbol X in a non-kernel rule, the corresponding move in the tree is (1) getting down from the kernel, by performing a PUSH, and (2) shifting X, by SWAPing the pushed position by the shifted one. The nk-T directly pushes a shifted position.

A second splitting occurs: $\epsilon$-reduction or not. Contrarily to a non-$\epsilon$ reduction the complete dotted rule is not a whole state that must be popped: the transition on the LHS is done without popping because its parent is in the same state.

### 6.4 LC PPDA.

LC parsing computes no prediction dynamically. It is so with probabilities. Each predicted position is statically computed with its (predicted) probability. As a LC state contains a part of shared forest, it will not be possible (or completely uninteresting) to extract the prediction probability of one tree. Hence, prediction probabilities will be computed only during the dynamic programming interpretation.

#### 6.4.1 The Underlying Automaton.

Each time a rule appears with its first symbol recognized (*i.e.* in non-kernel transitions and $\epsilon$-reductions) its probability must be communicated to the parser to compute the recognition probability. This justifies the following tables modifications:

- GOTO$_{nk}$(s, X) = {(r, p) | A$\rightarrow\bullet$X$\omega$ $\in$ s, A$\rightarrow$X$\bullet\omega$ $\in$ r and p = $p$(A$\rightarrow$X$\omega$)}.
- Non-kernel scans: nksc(r, p) s.t. A$\rightarrow$X$\bullet\omega$ $\in$ r and p = $p$(A$\rightarrow$X$\omega$)
- red(A$\rightarrow\epsilon$, $p_\epsilon$): $p_\epsilon$ = $p$(A$\rightarrow\epsilon$).

### 6.4.2 The Automaton.

**Stack alphabet :** $\mathcal{S}^{comp} \times [0, n] \times [0, n] \times \mathcal{D}(\mathcal{P}) \cup \blacktriangle$

**Initial stack** $[(s_0, 1, 1, 1) | \bot]$

**Final stack** $[(s_f, 1, n+1, \Pi) | \bot]$

KERNEL SCAN : if ksc(r) $\in$ ACTION(s, a), apply
SWAP (s, i, j, $\mathbf{R}$), a $\mapsto$ (r, i, j+1, $\mathbf{R}$)

NON-KERNEL SCAN : if nksc(r, p) $\in$ ACTION(s, a), apply
PUSH (s, i, j, $\mathbf{R}$), a $\mapsto$ (r, j, j+1, p), (s, i, j, $\mathbf{R}$)

KERNEL REDUCE :

- if red(A$\rightarrow$X$\omega$) $\in$ ACTION(s'', a) and r $\in$ GOTO$_k$(s, A), apply
  POP (s'', i, j, $\mathbf{R}''$), (s, k, i, $\mathbf{R}$), (a) $\mapsto$ (r, k, j, $\mathbf{R}\mathbf{R}''$)
- if red(A$\rightarrow\epsilon$, $p_\epsilon$) $\in$ ACTION(s, a) and r $\in$ GOTO$_k$(s, A), apply
  SWAP (s, i, j, $\mathbf{R}$), (a) $\mapsto$ (r, i, j, $\mathbf{R}p_\epsilon$)

NON-KERNEL REDUCE :

- if red(A$\rightarrow$X$\omega$) $\in$ ACTION(s'', a) and (r, p) $\in$ GOTO$_{nk}$(s, A), apply
  SWAP2 (s'', i, j, $\mathbf{R}''$), (s, k, i, $\mathbf{R}$), (a) $\mapsto$ (r, i, j, p$\mathbf{R}''$), (s, k, i, $\mathbf{R}$)
- if red(A$\rightarrow\epsilon$, $p_\epsilon$) $\in$ ACTION(s, a) and (r, p) $\in$ GOTO$_{nk}$(s, A), apply
  PUSH (s, i, j, $\mathbf{R}$), (a) $\mapsto$ (r, j, j, p$p_\epsilon$), (s, i, j, $\mathbf{R}$)

### 6.5 Proof indications.

As for the PDA, only non-kernel transitions and $\epsilon$-reductions are to be verified — the others being the same as Earley PPDA.

The non-kernel scan corresponds to recognizing a terminal from an initial position I : the prediction probability is $\underset{p}{P}(I) = p$, given by nksc(r,p).

For reductions, the probability of the recognized subtree $t$ must be multiplied to the previous position, as usual. In case of $\epsilon$-reduction, $\underset{r}{P}(t) = p_\epsilon$, given by ACTION, otherwise by the first popped 1-item ($\mathbf{R}''$). In case of non-kernel reduction, the recognition probability of the previous position is given by GOTO$_{nk}$ (p), otherwise by the second popped 1-item ($\mathbf{R}$).

## 7 LC Dynamic Programming Interpretation.

### 7.1 LC PDT.

To produce the shared forest, we need to introduce an extra parameter in the 1-items : the decorated shifted part of the kernel RHS.

**$\mathcal{E}$ domain :** $\mathcal{S}^{comp} \times [0, n] \times [0, n] \times \mathcal{W}^*$

**Initial stack :** $[(s_0, 1, 1, \epsilon) | \bot]$

**Final stack :** $[(s_f, 1, n+1, S^{1,n+1}) | \bot]$

**Transitions :**

**Scans :**

- KERNEL SCAN. if ksc(r) $\in$ ACTION(s, a), apply
  SWAP : (s, i, j, $\alpha$), a $\mapsto$ (r, i, j+1, $\alpha$a), $\varepsilon$.
- NON-KERNEL SCAN. if nksc(r) $\in$ ACTION(s, a), apply
  PUSH : (s, i, j, $\alpha$), a $\mapsto$ (r, j, j+1, a), $\varepsilon$.

**Reductions**

- KERNEL REDUCE.
  - if red(A→Xω) ∈ ACTION(s, a) and r ∈ $GOTO_k$(s′, A), apply POP : (s, i, j, α), (s′, k, i, β), (a) ↦ (r, k, j, $\beta A^{i,j}$), $A^{i,j}$→α.
  - if red(A→ϵ) ∈ ACTION(s, a) and r ∈ $GOTO_k$(s, A), apply SWAP : (s, i, j, α), (a) ↦ (r, i, j, $\alpha A^{j,j}$), $A^{j,j}$→ϵ.
- NON-KERNEL REDUCE.
  - if red(A→Xω) ∈ ACTION(s, a) and r ∈ $GOTO_{nk}$(s′, A), apply SWAP2 : (s, i, j, α), (s′, k, i, β), (a) ↦ (r, i, j, $A^{i,j}$), $A^{i,j}$→α
  - if red(A→ϵ) ∈ ACTION(s, a) and r ∈ $GOTO_{nk}$(s, A), apply PUSH : (s, i, j, α), (a) ↦ (r, j, j, $A^{j,j}$), $A^{j,j}$→ϵ.

### 7.2 PPDA Interpretation.

#### 7.2.1 The Underlying Automaton.

**Definition 26.** *Dotted rule prediction probability.*
Let $s$ be a LC state, $p_p$(A→α•β | $s$) = $p_p$([A→α•β, i, j]) / $p_p$(k($s$), i, j)).

Hence, $p_p$(k($s$) | $s$) = 1 and $p_p$(A→•Bα | $s$) = $p$(A→Bα) $p_s$(•B),
where $p_s$(•B) = $\sum_{C\to\omega\bullet B\delta\in s} p_p(C\to\omega\bullet B\delta|s)$. [6]

As for $p$(•$B_j$), the $p_s$(•B) form a invertible linear system, that is then statically inverted.

Only non-kernel transitions are changed : they have an extra parameter to take in account the statically computed prediction probabilities.

Let (r, p, $\pi$) be in a non-kernel transition from s on X. Then A→•Xω ∈ s, k(r) = A→X•ω and $\pi$ = $p_p$(A→•Xω | s).

#### 7.2.2 The Symbolic Probabilities.

For Earley strategy, we introduced two kinds of variable symbol : $p$(•$B_j$) for prediction, $p$($A_{ij}$•) for completion. The last one is simply adapted to LC states, the prediction adaptation is less immediate because predictions are made during non-kernel transitions.

- $p$(•$r_j$) is the prediction probability of r kernel rule in its initial position at level j.

$$p(\bullet r_j) = \sum_{\substack{(s,i,j,\mathbf{P},\mathbf{R},\alpha)\in\mathcal{E}\\ s.t.\\ (r,p,\pi)\in GOTO_{nk}(s,A)\\ or\\ nksc(r,p,\pi)\in ACTION(s,x_j)}} \mathbf{P}\pi$$

- The ϵ-reductions force us to make a particular case.

  - i ≠ j :

$$p(A_{ij}\bullet) = \sum_{\substack{(s'',i,j,\mathbf{P}'',\mathbf{R}'',\alpha)\in\mathcal{E}\\ s.t.\ red(A\to\omega)\in ACTION(s'',x_j)}} \mathbf{R}''$$

  -

$$p(A_{jj}\bullet) = \sum_{\substack{(s,i,j,\mathbf{P},\mathbf{R},\alpha)\in\mathcal{E}\\ s.t.\ red(A\to\epsilon,p_\epsilon)\in ACTION(s,x_j)}} p_\epsilon$$

Without ϵ-rules, values are just incrementally accumulated, without needing any matrix inversion. Anyway we saw in the invertibility proof, section 5.2.3, that the inversions can be performed statically.

[6] $p_s$(•B) appears in [Wright&Wrigley 89] for a probabilistic LR parser, it is noted $P_B$.

### 7.2.3 LC PPDT.

**$\mathcal{E}$ domain:** $\mathcal{S}^{comp} \times [0, n] \times [0, n] \times \mathcal{D}(\mathcal{P}) \times \mathcal{D}(\mathcal{P}) \times \mathcal{W}^*$

**Initial stack:** $[(s_0, 1, 1, 1, 1, \epsilon) | \bot]$

**Final stack:** $[(s_f, 1, n+1, \Pi, \Pi, S^{1,n+1}) | \blacktriangle]$

**Inter-level, scans:**

- KERNEL SCAN: if ksc(r) $\in$ ACTION(s, a), apply
  SWAP (s, i, j, $\mathbf{P}$, $\mathbf{R}$, $\alpha$), a $\mapsto$ (r, i, j+1, $\mathbf{P}$, $\mathbf{R}$, $\alpha$a), $\varepsilon$
- NON-KERNEL SCAN: if nksc(r, p, $\pi$) $\in$ ACTION(s, a), apply
  PUSH (s, i, j, $\mathbf{P}$, $\mathbf{R}$, $\alpha$), a $\mapsto$ (r, j, j+1, $\mathcal{P}(\bullet r_j)$, p, a), $\varepsilon$

**Intra-level, reductions:**

- KERNEL REDUCE:
  - if red(A$\rightarrow X\omega$) $\in$ ACTION(s'', a) and r $\in$ GOTO$_k$(s, A), apply
    POP (s'', i, j, $\mathbf{P}''$, $\mathbf{R}''$, $\alpha$), (s, k, i, $\mathbf{P}$, $\mathbf{R}$, $\beta$), (a) $\mapsto$ (r, k, j, $\mathbf{P}\mathcal{P}(A_{ij}\bullet)$, $\mathbf{R}\mathcal{P}(A_{ij}\bullet)$, $\beta A^{i,j}$), ($A^{i,j}\rightarrow\alpha$, $\mathbf{R}''$).
  - if red(A$\rightarrow\epsilon$, $p_\epsilon$) $\in$ ACTION(s, a) and r $\in$ GOTO$_k$(s, A), apply
    SWAP (s, i, j, $\mathbf{P}$, $\mathbf{R}$, $\alpha$), (a) $\mapsto$ (r, i, j, $\mathbf{P}\mathcal{P}(A_{jj}\bullet)$,$\mathbf{R}\mathcal{P}(A_{jj}\bullet)$, $\alpha A^{j,j}$), ($A^{j,j}\rightarrow\epsilon$, $p_\epsilon$).
- NON-KERNEL REDUCE:
  - if red(A$\rightarrow$X$\omega$) $\in$ ACTION(s'', a) and (r, p, $\pi$) $\in$ GOTO$_{nk}$(s', A), apply
    SWAP2 (s'', i, j, $\mathbf{P}''$, $\mathbf{R}''$, $\alpha$), (s, k, i, $\mathbf{P}$, $\mathbf{R}$, $\beta$), (a) $\mapsto$ (r, i, j, $\mathcal{P}(\bullet r_i)\mathcal{P}(A_{ij}\bullet)$, $p\mathcal{P}(A_{ij}\bullet)$, $A^{i,j}$), ($A^{i,j}\rightarrow\alpha$, $\mathbf{R}''$).
  - if red(A$\rightarrow\epsilon$, $p_\epsilon$) $\in$ ACTION(s, a) and (r, p, $\pi$) $\in$ GOTO$_{nk}$(s, A), apply
    PUSH (s, i, j, $\mathbf{P}$, $\mathbf{R}$, $\alpha$), (a) $\mapsto$ (r, j, j, $\mathcal{P}(\bullet r_j)\mathcal{P}(A_{jj}\bullet)$, $p\mathcal{P}(A_{jj}\bullet)$, $A^{j,j}$), ($A^{j,j}\rightarrow\epsilon$, $p_\epsilon$).

### 7.3 Proof indications.

Non-kernel scan: the prediction probability is exactly $\mathcal{P}(\bullet r_j)$ definition. Because of r kernel, the subtree identified by (r, j, j+1) has a single node, and p is its probability.

Kernel-$\epsilon$-reduction: w.r.t. the PPDA, $p_\epsilon$ is replaced by $\mathcal{P}(A_{jj}\bullet)$, which sums the probability of all the subtrees rooted in $A^{j,j}$.

Non-kernel non-$\epsilon$-reductions: in a similar way $\mathcal{P}(A_{ij}\bullet)$ replaces the prediction probability of the reduced 1-item. W.r.t. Earley, $\mathcal{P}(\bullet r_j)$ summarizes the prediction probabilities of the Earley-items of the form [A$\rightarrow\bullet$B$\beta$, i, j].

## 8 Conclusion.

Like [Stolcke 93], we have presented an Earley-like stochastic parser that computes prefix and sub-parse-trees probabilities. The stochastic shared forest is produced with the subtree probability on each node.

Under Stolcke's first assumption (no $\epsilon$-rules), our LC algorithm improves his, since the LC states contain all the necessary prediction probabilities. The improvement is less important without this assumption, but [Schabes 91] ideas on $\epsilon$-rules optimization give back its interest to LC strategy (w.r.t. Earley): it consists in computing the $\epsilon$-reductions statically, within the LC states (probabilities just follow) hence no prediction loops remain to be managed during parsing.

The most likely (Viterbi) parse can be easily obtained by replacing the + operator, in the probability domain, by *max* (loops are not to be followed since they lower the probability).

More formally, we have showed that Earley and LC strategies amount to a left-to-right depth-first exploration of the parse trees. It justifies the prediction probability definition as a left derivation probability, what permits to see it as a prefix probability.

More generally, we adopted the formal languages and automata theory point of view. Algorithms are presented within a framework that naturally generalizes the purely syntactic ones and their computations are proved correct w.r.t. the definitions.

## References


[Booth-Thompson 73] Taylor L. Booth, Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Transaction on Computers*, C-22(5) pp 442–450, 1973.

[Earley 70] Jay Earley. An efficient context-free parsing algorithm. Communications of the ACM 6. pp 451–455, 1970.

[Lang 74] Bernard Lang. Deterministic techniques for efficient non-deterministic parsers. Editor J. Loeckx. *Procedings of the Second Colloquium on Automata, Languages and Programming*, vol. 14 of *Lectures Notes in Computer Science*, pp 255–269. Springer-Verlag, 1974.

[Lang 89] Bernard Lang. Towards a uniform formal framework for parsing. Current issues in parsing technology, M.Tomita ed., Kluver Academic Publisher. 1991.

[Leermakers 89] R. Leermakers. How to cover a grammar. $27^{th}$ *meeting of the association for computational linguistics*, Vancouver, 1989.

[Schabes 91] Yves Schabes. Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. $29^{th}$ meeting of the association for computational linguistics, Berkeley, 1991.

[Stolcke 93] Andreas Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. TR-93-065, Berkeley, 1993.

[Tendeau 95] Frédéric Tendeau. Stochastic parse-tree recognition by a pushdown automaton. To appear as an INRIA-Rocquencourt research report, 1995.

[VdlC 93] Éric Villemonte de la Clergerie. Automates à pile et programmation dynamique - DyALog : une application à la programmation en logique. *PhD thesis, Université Paris VII*, 1993.

[Wright&Wrigley 89] J.H Wright, E.N. Wrigley. Probabilistic LR parsing for speech recognition. *Proceedings IWPT*, pp 105–114, 1989.

# An HPSG-based Parser for Automatic Knowledge Acquisition

Kentaro Torisawa† Jun'ichi Tsujii‡

†‡Department of Information Science
University of Tokyo
Hongo 7-3-1, Bunkyo-ku Tokyo, 113, Japan
{torisawa,tsujii}@is.s.u-tokyo.ac.jp

‡CCL, UMIST
PO BOX 88, Manchester M60 1QD U.K.
tsujii@ccl.umist.ac.uk

## 1 Introduction

Our aim is to build an HPSG [Pollard, 1993] based parser that can be used as a component of a knowledge acquisition(KA) system from unrestricted text[Horiguchi, 1995]. KA proceeds by using *underspecified lexical entry templates* given to each part of speech for words. By filling out the underspecified parts of them through unification, knowledge is acquired.

Our contention is that we cannot give an exhaustive set of specific *CFG skeletons* to the parser prior KA, in order to obtain a wide coverage required for handling corpora. In our parser, rules with *CFG skeletons*, which are widely used in HPSG implementations such as [Carpenter, 1994], are replaced with a few *rule schemata* and principles, whose examples are shown in Fig. 1 and 2. They do not specify particular syntactic categories and can cover most of the linguistic constructions in corpora by relying on lexicalization and augmentation with definite clause programs. However, this replacement prevents us from using optimization techniques for conventional unification-based parsers.

Our parser adopts a two-phased architecture. Phase 1 is a bottom-up parsing with compiled object-oriented code realizing only part of constraints in a full grammar. A full grammar is applied to completed parse trees in Phase 2.

Applicaton of rule schemata and their principles is monotone because of monotonicity of unification. (i.e. for any feature structures $F_0$,$F_1$, $F_0'$, $F_1'$, if $F_0 \sqsubseteq F_1$, $F_0' \sqsubseteq F_1'$ $F_0 \sqcup F_1 \sqsubseteq F_0' \sqcup F_1'$.). If a sign $S$ subsumes a sign $S'$ and the application of principles or rule schemata to $S'$ succeeds, the application to $S$ also succeeds and the results reserves their daughters' subsumption relation. Our basic

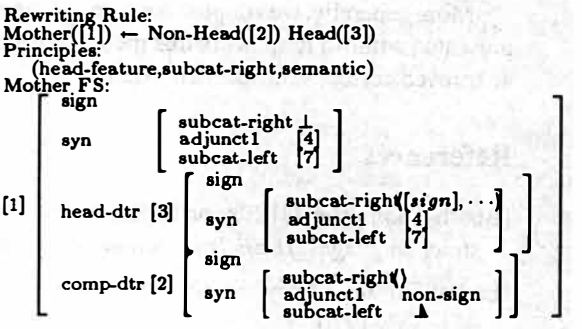



Figure 1: An example of a rule schema.

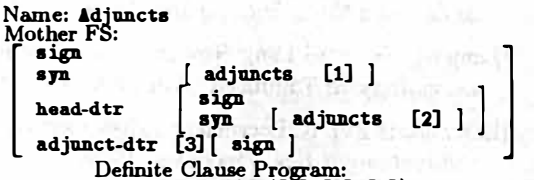



Figure 2: Adjuncts Principle

idea is that we can systematically *weaken* lexical entries and other grammar components by eliminating certain constraints in them so that they are compiled to simple objects and cheaper procedures without losing the ability of a full grammar. Although the compiled *grammar* overgenerates, illegitimate signs are removed in Phase 2.

## 2 Compilation

Our compiler produces two items, *Sign Objects*, which are objects corresponding to signs, and *Rule Methods*, which play roles of rule schemata and principles. Both items are directly executed in Common Lisp Object System. Rule methods take sign objects representing daughter signs as input and produce sign objects corresponding to mothers. Sign objects have slots corresponding to only part of feature structures. This *reduction* is justified by monotonicity of unification.

Each slot of a sign object contains a fragment of the feature structure or other Lisp objects converted from the feature structure, such as symbols representing types. Fig. 3

```
(SLOT-NAME    SLOT-VALUE)
(SUBCAT-RIGHT ((E-LIST NIL #(FT @ #x921a82))))
(SUBCAT-LEFT      ((NIL      NON-OBLIGATORY-
SIGN #(FT @ #x921a9a))))
(ADJUNCTS2 (SIGN
        #(FT @ #x92170a)
        ((ADJUNCT1 NON-SIGN :SINGLETON))
        (SELECTING-FEATURE-SHARING
                SUBCAT-LEFT
                SUBCAT-LEFT)))
```

Figure 3: A compiled lexical entry

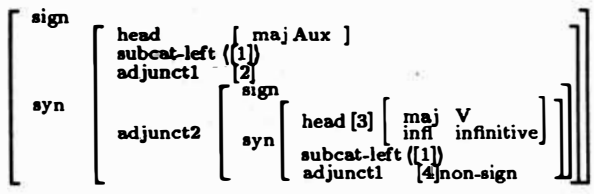



Figure 4: An original lexical entry

shows the sign object compiled from the lexical entry in Fig. 4. In this sign object, the **ADJUNCTS2** slot contains the type **sign** and a head-feature **#(FT @ #x92170a)**, which represents the feature structure denoted by the tag **[3]** in Fig. 4. The third element represents the constraint that the **ADJUNCTS2** value of a selected sign must be **non-sign**. This also corresponds to the feature structure tagged as **[4]** in Fig. 4. The fourth element is a command to transfer the **subcat-left** value of a selected sign to the mother's same slot. This transfer is represented by the structure sharing **[1]** in the original lexical entry.

A rule method contains only part of the constraints realized in principles and rule schemata. This reduction of constraints corresponds to the elimination of feature structures, structure sharings and part of a definite clause program in a rule method or its principles. The soundness of this reduction can be proven by monotonicity of unification. Furthermore, some unification evoked by structure sharing is replaced by simple assignements of slot values. For example, most feature *raising* is performed by assignments. This replacement does not affect the soundness of our compilation because any two feature structures always subsume their unified one. If a sign object is created by the code containing assignement instead of unification, it subsumes the sign object created as the result of unification a unification. Thus, a sign object with rule methods always subsume the sign to be created by the original grammar.

Rule schemata with their principles are categorized as 1) rule schemata to *use* selecting features, such as **SUBCAT**, which are feature structures to be unified with another sign. and 2) rule schemata to transfer selecting features such as a rule schema augmented with a trace principle. The first category is divided further into three according to the type of the selecting features (singleton, list or set)in a rule schema.

The compiler generates rule methods by filling out a template which is prepared for each type of rule schemata with references to a rule schema and its principles. The differences among code templates reflect the differences of the definite clause programs to be evoked in application of each type of rule schemata. For example, **Cancel-a-member** in Fig. 2 is the program for *using* selecting features of set type. The *behaviour* of such important parts of the definite clause programs are reflected directly in the templates. The following is the template for a rule schema for the list type selecting features.

```
(lambda (selector selectee)
  (if (and (selecting-feature-unifiable?
             (<selecting-feature>
                              selector)
             selectee)
           <other-unifiability-checking>)
      (let ((mother-sign (create-sign)))
        <feature-raising>
        <evaluate-structure-sharing-commands>
        mother-sign)
    nil))
```

The rule method takes daughter sign objects, which are bound to the variables **selector** and **selectee** in the argument list, and produces their mother sign object bound to the variable **mother-sign**.

# 3 Conclusion

For the rule schemata presented in Fig. 1, the compiled code is about 43 times as fast as the application of the rule schemata and its principles. For a 25 word sentence, bottom-up parsing with the compiled code followed by the applications of the original grammar to the completed parse trees was 3.1 times as fast as the parsing with only the original grammar. The required storage was 370% less than that of the original.

Linguistically well-defined grammar formalisms such as HPSG have been regarded as inappropriate for dealing with unrestricted real-world text. In order to build feasible systems, researchers have relied on more procedural grammar whose well-defined-ness is difficult to show. However, by using our compilation technique, we will be able to develop a robust and efficient HPSG-based parser which can be a component of a practical system.

# References

[Carpenter, 1994] Carpenter, B. and Penn, G. (1994). *The Attibute Logic Engine User's Guide.* Carnegie Mellon University.

[Horiguchi, 1995] Horiguchi, K., Torisawa, K., and Tsujii, J. (1995). Automatic acquisition of content words using an HPSG-based parser. to be submitted.

[Pollard, 1993] Pollard, C. and Sag, I. A. (1993). *Head-Driven Phrase Structure Grammar.* CSLI Publications.

# PARSING D-TREE GRAMMARS

K. Vijay-Shanker
Computer & Information Science
University of Delaware
vijay@udel.edu

David Weir
Cognitive & Computing Sciences
University of Sussex
david.weir@cogs.susx.ac.uk

Owen Rambow
CoGenTex, Inc.
Ithaca, NY 14850
owen@cogentex.com

## 1 Motivation

D-Tree Grammars (DTG) (Rambow et al., 1995) arise from work on Lexicalized Tree-Adjoining Grammars (LTAG) (Joshi & Schabes, 1991). A salient feature of LTAG is the extended domain of locality it provides. Each elementary structure can be associated with a lexical item and properties related to the lexical item (such as subcategorization) can be expressed within the elementary structure. In addition, LTAG remain tractable, yet their generative capacity is sufficient to account for certain syntactic phenomena that, it has been argued, lie beyond Context-Free Grammars (CFG) (Shieber, 1985). LTAG, however, has two limitations: the LTAG operations of substitution and adjunction do not map cleanly onto the relations of complementation and modification; and they cannot provide analyses for certain syntactic phenomena. We briefly discuss the first issue here, and refer to Rambow et al. (1995) for a broader discussion of both issues.

In LTAG, the operations of substitution and adjunction relate two lexical items, establishing either the linguistic relation of complementation (predicate-argument relation) or modification between them. These relations represent important linguistic intuition, they provide a uniform interface to semantics, and they are, as Schabes & Shieber (1994) argue, important in order to support statistical parameters in stochastic frameworks and appropriate adjunction constraints in LTAG.

However, the LTAG composition operations are not used uniformly: while substitution is used only to add a (nominal) complement, adjunction is used both for modification and (clausal) complementation. Clausal complementation could not be handled uniformly by substitution because of the existence of syntactic phenomena such as long-distance *wh*-movement in English. Furthermore, there is an inconsistency in the directionality of the operations used for complementation in LTAG: nominal complements are substituted into their governing verb's tree, while the governing verb's tree is adjoined into its own clausal complement. The fact that adjunction and substitution are used in a linguistically heterogeneous manner means that (standard) LTAG derivation trees do not provide a good representation of the dependencies between the words of the sentence, i.e., of the predicate-argument and modification structure.

In defining DTG we have attempted to resolve this problem with the use of a single operation (that we call subsertion) for handling all complementation and a second operation (called sister-adjunction) for modification. However, the definition remains faithful to what we see as the key advantages of LTAG (in particular, its enlarged domain of locality)[1].

## 2 Definition of DTG

A **d-tree** is a tree with two types of edges: domination edges (**d-edges**) and immediate domination edges (**i-edges**). For each internal node, either all of its daughters are linked by i-edges or it has a single daughter that is linked to it by a d-edge. Each node is labelled with a terminal symbol, a nonterminal symbol or the empty string. A d-tree containing $n$ d-edges can be decomposed into $n + 1$ **components** containing only (0 or more) i-edges, with the root of

[1] Related formalisms can be found in Becker et al. (1991) and Rambow (1994a).

all components other than the topmost one connected to a node in a component above by a d-edge. In the d-trees $\alpha$ and $\beta$ shown in Figure 1, these components are shown as triangles.

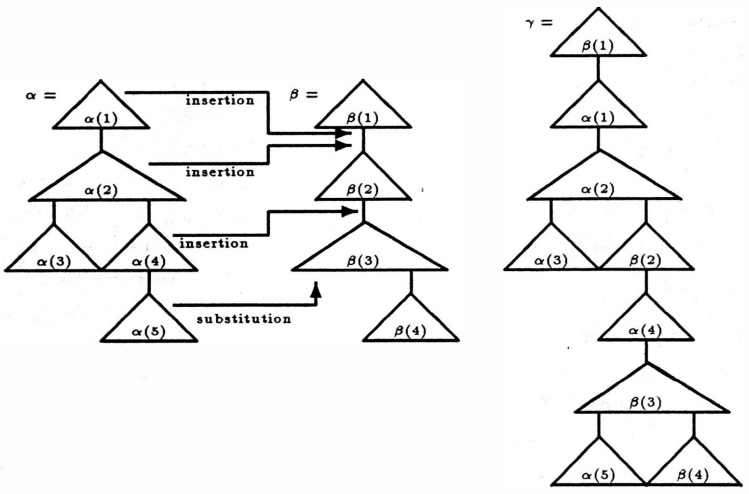



Figure 1: Subsertion

*Subsertion* is an operation on d-trees that is a generalization of tree substitution. When a d-tree $\alpha$ is subserted into another d-tree $\beta$, a component of $\alpha$ is substituted at a frontier nonterminal node (a **substitution node**) of $\beta$ and all components of $\alpha$ that are above the substituted component are inserted into d-edges in $\beta$ above the substituted node or placed above the root node of $\beta$. For example, Figure 1 illustrates a possible subsertion of $\alpha$ in $\beta$ with the component $\alpha(5)$ being substituted at a substitution node in $\beta$. In the composed d-tree, $\gamma$, the components, $\alpha(1)$, $\alpha(2)$, and $\alpha(4)$ of $\alpha$ above $\alpha(5)$ are inserted in the path in $\beta$ between the root and the substitution node. In general, some of these components could have been placed above the root of $\beta$ as well. When a component $\alpha(i)$ of some d-tree $\alpha$ is inserted into a d-edge between nodes $\eta_1$ and $\eta_2$ in $\beta$ two new d-edges are created, the first of which relates $\eta_1$ and the root node of $\alpha(i)$, and the second of which relates the frontier node of $\alpha(i)$ that dominates the substituted component to $\eta_2$. It is possible for components above the substituted node to drift arbitrarily far up in any way that is compatible with the domination relationships present in the substituted d-tree. DTG provide a mechanism called **subsertion-insertion constraints** to control what can appear within d-edges (see below).

The second composition operation involving d-trees is called sister-adjunction. The target node of sister-adjunction must be a node at the top of an i-edge. When a d-tree $\alpha$ is sister-adjoined at a node $\eta$ in a d-tree $\beta$ the composed d-tree $\gamma$ results from the addition to $\beta$ of $\alpha$ as a new leftmost or rightmost sub-d-tree below $\eta$. Note that sister-adjunction involves the addition of exactly one new immediate domination edge and that several sister-adjunctions can occur at the same node. **Sister-adjoining constraints** specify where d-trees can be sister-adjoined and whether they will be right- or left-sister-adjoined (see below).

A DTG is a four tuple $G = (V_N, V_T, S, D)$ where $V_N$ and $V_T$ are the usual nonterminal and terminal alphabets, $S \in V_N$ is a distinguished nonterminal and $D$ is a finite set of **elementary** d-trees. A DTG is said to be **lexicalized** if each d-tree in the grammar has at least one terminal node. The elementary d-trees of a grammar $G$ have two additional annotations: subsertion-insertion constraints and sister-adjoining constraints. These will be described below. We first discuss DTG derivations and subsertion-adjoining trees (SA-trees), which are partial derivation structures that represent dependency information, the importance of which was stressed in the introduction. SA-trees contain information regarding the substitutions and sister-adjunctions that take place in a derivation, capturing the linguistic dependency relations of complementation and modification.

Consider a DTG $G = (V_N, V_T, S, D)$. We assume some naming convention for the elementary d-trees in $D$ and some consistent ordering on the components and nodes of elementary d-trees in $D$. Any elementary d-tree is also considered a derived d-tree. The components of the elementary d-tree are considered the **substitutable**[2] components of this tree considered as a derived d-tree. Derivations involve sister-adjoining or subsertions. If $\alpha$ is subserted in $\beta$, only substitutable components of $\alpha$ may be substituted (at a substitution node in $\beta$). In the resulting d-tree, say $\gamma$, the components of $\gamma$ that came from $\beta$ are now the substitutable components of $\gamma$ and are ordered (or numbered) in the same manner as in $\beta$.

The **tree set** $T(G)$ generated by $G$ is defined as the set of trees $\gamma$ such that there is a derived d-tree, $\gamma'$ rooted with the nonterminal $S$, whose frontier is a string in $V_T^*$; and $\gamma$ results from the removal of all d-edges from $\gamma'$. A d-edge is removed by merging the nodes at either end of the edge as long as they are labelled by the same symbol. The **string language** $L(G)$ associated with $G$ is the set of terminal strings appearing on the frontier of trees in $T(G)$.

As in derivation trees of a LTAG, the nodes in SA-trees are labeled by the names of the elementary d-trees. The edges depict the composition operations used. In the case of sister-adjoining, an edge labelled $(d, m)$ between nodes labelled by two elementary d-trees, say from $\alpha$ to $\beta$, represents sister adjoining of $\beta$ at node in $\alpha$ addressed $m$. $d \in \{\text{left}, \text{right}\}$ indicates whether left- or right-sister-adjunction took place. For subsertion, an edge labelled $(l, m)$ encodes the substitution of the $l^{th}$ component of $\beta$ at the substitution node addressed $m$ in $\alpha$.

As indicated earlier, an SA-tree only partially captures the derivation of a d-tree, and does not specify the placement of the inserted components during subsertions. A derivation graph for $\gamma \in T(G)$ results from the addition of insertion edges to a SA-tree $\tau$ for $\gamma$. The location in $\gamma$ of an inserted elementary component $\alpha(i)$ can be unambiguously determined by identifying the source of the node (say the node with address $n$ in the elementary d-tree $\alpha'$) with which the root of this occurrence of $\alpha(i)$ is merged with when d-edges are removed. The insertion edge will relate the two (not necessarily distinct) nodes corresponding to appropriate occurrences of $\alpha$ and $\alpha'$ and will be labelled by the pair $(i, n)$.

Each d-edge in elementary d-trees has an associated subsertion-insertion constraint (SIC). A SIC is a finite set of elementary node addresses (ENAs). An ENA $\eta$ specifies some elementary d-tree $\alpha \in D$, a component of $\alpha$ and the address of a node within that component of $\alpha$. If an ENA $\eta$ is in the SIC associated with a d-edge between $\eta_1$ and $\eta_2$ in an elementary d-tree $\alpha$ then $\eta$ cannot appear properly within the path that appears from $\eta_1$ to $\eta_2$ in the derived tree $\gamma \in T(G)$.

Each node at the top of an i-edge of elementary d-trees has an associated sister-adjunction constraint (SAC). A SAC is a finite set of pairs, each pair identifying a direction, $d \in \{\text{left}, \text{right}\}$, and an elementary d-tree, $\alpha$. A SAC gives a complete specification of what can be sister-adjoined at a node. If a node $\eta$ is associated with a SAC containing a pair $(d, \alpha)$ then the d-tree $\alpha$ can be $d$-sister-adjoined at $\eta$. Further details of the motivation and definition of DTG, example d-trees and derivations, and usage of SIC and SAC may be found in Rambow et al. (1995).

# 3 Parsing Framework

Our approach to DTG parsing is inspired by a methodology introduced by Lang (Billot & Lang, 1989; Lang, 1991) for CFG. This involves dividing the parsing process into two phases. For example, given a CFG, an equivalent PDA is constructed. The transition moves of this (non-deterministic) PDA represent the primitive steps of the recognition process. Dynamic programming techniques then serve to determinize the recognition process. This approach is particularly well-suited to formalisms such as LTAG and DTG that have an enlarged domain of locality, where individual structures can span noncontiguous substrings. Adjunction of two structures does not correspond to a single step in the recognition process but has to be decomposed into several more primitive steps in recognizers that only consider contiguous

[2] The notion of substitutability is used to ensure the SA-tree is a tree. That is, an elementary structure cannot be subserted into more than one structure since this would be counter to our motivations for using subsertion for complementation.

substrings. This decomposition corresponds to the conversion from grammar to automaton in Lang's framework.

We adopt a variant of this approach using a second grammar formalism rather than an automaton model. In the case of LTAG, the LTAG-equivalent formalisms of LIG or HG are suitable candidates, since unlike LTAG, in both formalisms, each rewriting step corresponds directly to the basic steps of recognition. Indeed, parsing algorithms for LTAG have been developed by adapting the algorithms for LIG or HG (Vijay-Shanker & Weir, 1993a)[3].

The elementary structures of DTG, like LTAG, have an enlarged domain of locality. A single subsertion involves one substitution and possibly several insertions. The recognition process corresponding to a single subsertion is comprised of several more primitive steps. As with LTAG, it is valuable to use a formalism with a smaller domain of locality. For this reason, we introduce the Linear Prioritized Multiset Grammar (LPMG) formalism. LPMG is similar to LIG differing in its use of multisets in place of stacks.

We provide a translation from DTG to LPMG that captures the primitive steps of DTG recognition. Specialization of the recognition grammar (the LPMG obtained) for an input serves to encode the set of all parses for that input. Clearly, one could develop an algorithm for DTG directly and have the translation to LPMG be implicit. However, we make the translation explicit so that we can systematically consider different dynamic programming methods to obtain different DTG recognition algorithms. For space reasons, we consider only one Earley-style method in this paper. It would be equally straightforward to develop a CKY-style algorithm from the LPMG grammar.

# 4 Definition of LPMG

We now define LPMG, which is a variant of {}-LIG (Rambow, 1994b) obtained by providing an additional mechanism for restricting the manipulation of the members of the multiset. A multiset over a finite alphabet $V$ is a function $m : V \rightarrow \mathsf{N}$ where $\mathsf{N}$ is the natural numbers. The set of all multisets over $V$ is denoted $\mathsf{N}^V$. For two multisets $m_1$ and $m_2$ over $V$, multiset addition and subtraction are defined such that for all $v \in V$: $(m_1 \oplus m_2)(v) = m_1(v) + m_2(v)$ and $(m_1 \ominus m_2)(v) = max\,(m_1(v) - m_2(v), 0)$. When context permits we will abuse this notation and not distinguish between a symbol and the multiset that contains only that symbol. An ordering on multisets is defined as: $m_1 \sqsubseteq m_2$ iff $m_1(v) \leq m_2(v)$ for all $v \in V$. The empty multiset is denoted $\phi$. The number of elements in the multiset $m$ is denoted $size\,(m)$.

A LPMG is a six tuple $G = (V_N, V_T, V_M, S, \sigma_0, P, \rho)$ where $V_N$, $V_T$ and $V_M$ are the nonterminal, terminal and multiset alphabets, respectively; $S \in V_N$ and $\sigma_0 \in V_M$ are the initial nonterminal and multiset symbols, respectively; $\rho \subseteq V_M \times V_M$ is the priority relation and $P$ is a finite set of productions having the form $A[\sigma] \rightarrow x$ where $A \in V_N$ and $x \in V_T \cup \{\epsilon\}$; or $A[\cdots \sigma \cdots] \rightarrow X_1 \ldots X_k$ where $k \geq 1$ and for $1 \leq i \leq k$, $X_i \in \{ A[\sigma_1, \ldots, \sigma_k] \mid A \in V_N,\ k \geq 1$ and $\sigma_1, \ldots, \sigma_k \in V_M \} \cup \{ A[\cdots \sigma_1, \ldots, \sigma_k \cdots] \mid A \in V_N,\ k \geq 0$ and $\sigma_1, \ldots, \sigma_k \in V_M \}$

Given a priority relation $\rho$ we define $\rho$-selectable as follows: $\sigma$ is $\rho$-selectable from a multiset $m$ iff $m(\sigma) \geq 1$ and for all $\sigma'$ such that $\langle \sigma', \sigma \rangle \in \rho$ we have $m(\sigma') = 0$. In derivations sentential forms are strings of objects taken from $V_T \cup \mathsf{M}(V_N, V_M)$ where $\mathsf{M}(V_N, V_M) = \{ A[m] \mid A \in V_N$ and $m \in \mathsf{N}^{V_M} \}$. For all $\Upsilon_1, \Upsilon_2 \in (\mathsf{M}(V_N, V_M) \cup V_T)^*$: (1) if $A[\sigma] \rightarrow x \in P$ then $\Upsilon_1 A[\sigma] \Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 x \Upsilon_2$; and (2) if $A[\cdots \sigma \cdots] \rightarrow X_1 \ldots X_k \in P$, $\sigma$ is $\rho$-selectable from $m$, $m \ominus \sigma = m'_1 \oplus \ldots \oplus m'_k$, $X_i = A_i[\sigma_{i,1}, \ldots, \sigma_{i,n_i}]$ or $X_i = A_i[\cdots \sigma_{i,1}, \ldots, \sigma_{i,n_i} \cdots]$ for each $i$ $(1 \leq i \leq k)$, $m_i = m'_i \oplus \sigma_{i,1} \oplus \ldots \oplus \sigma_{i,n_i}$ for each $i$ $(1 \leq i \leq k)$ and $m'_i = \phi$ if $X_i = A_i[\sigma_{i,1}, \ldots, \sigma_{i,n_i}]$ for each $i$ $(1 \leq i \leq k)$ then $\Upsilon_1 A[m] \Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 A_1[m_1] \ldots A_k[m_k] \Upsilon_2$.

In this latter derivation step, we rewrite $A[m]$ by using a production with $A[\cdots \sigma \cdots]$ in its left-hand-side. This is possible only when $\sigma \in m$ and the contents of $m$ and the priority relation

[3]The use of grammars rather than automata ties in with Lang's later work (for instance, see Lang (1994)) where grammars can be specialized for a specific input to represent the shared forest of derivation trees for that input. In particular, Vijay-Shanker & Weir (1993b), show that for LTAG an equivalent HG or LIG can be used in the grammar specialization process, rather than the object (LTAG) grammar.

allows the rewriting which is verified by the definition of $\rho$-selectivity. The application of the rule will remove the $\sigma$ from $m$ and the remaining multiset elements are distributed amongst the $k$ elements in the right-hand-side of the rule. The multisets inherited by the $i^{th}$ element is indicated as $m'_i$. Thus, $m'_1 \oplus \cdots \oplus m'_k$ must be equal to $m \ominus \sigma$. If the $i^{th}$ element is $A_i[\sigma_{i_1}, \ldots, \sigma_{i,k_i}]$ then the derivation from it should only have the multiset $m_i = \phi \oplus \sigma_{i_1} \oplus \ldots \oplus \sigma_{i,k_i}$. That is, it does not inherit any multiset elements. This is indicated above by stating that $m'_i = \phi$ in this case. On the other hand, if the $i^{th}$ element is $A_i[\cdots \sigma_{i_1}, \ldots, \sigma_{i,k_i} \cdots]$ then the derivation from it should not only have the multiset elements $\sigma_{i_1}, \ldots, \sigma_{i,k_i}$ associated with it, but also $m'_i$, that part of the multiset $m$ that is inherited by the $X_i$. The language $L(G)$ generated by $G$ is the set of terminal strings derived from $S[\sigma_0]$.

# 5 DTG to LPMG Conversion

The construction described here is similar to the LTAG to LIG conversion described by Vijay-Shanker & Weir (1994) that has been used in the development of LTAG parsing algorithms (Vijay-Shanker & Weir, 1993a). Adjunction has the effect of embedding one elementary tree within another and the LIGs stack is used to control the unbounded nesting of elementary trees that occur in LTAG derivations. Following the UVG-DL to {}-LIG conversion described by Rambow (1994a), the DTG to LPMG conversion described below is similar to the LTAG to LIG conversion except that multisets rather than stacks are used to control the embedding of d-trees. This is because there is limited control over the relative positioning of the inserted components of two subserted d-trees.

Embedding context are multisets of ENAs (elementary node addresses). We consider embedding of d-trees only at nodes at the top of d-edges. When this happens the multiset stores the ENA at the bottom of the d-edge and when this node is reached in the derivation, this ENA is removed from the multiset. Thus, *open* d-edges are represented by elements of the multiset (corresponding to nodes at the bottom of the d-edge) and removal of these elements from the multiset corresponds to *closing* the corresponding d-edge. The multiset can be seen as a record of the elementary nodes that are still to be visited in a top-down traversal of the derived d-tree.

LPMG nonterminals are used to encode the current ENA and the productions for each ENA are determined by the context of ENA in its elementary d-tree. That is, the productions depend on whether the node is at the top of a d-edge, top of an i-edge, or a substitution node. Productions correspond to inserting or not inserting within a d-edge, substituting a component, or sister-adjoining a d-tree. When we apply a production corresponding to the insertion of some ENA we must check that the ENA does not appear in the SIC that is associated with some open d-edge. As every open d-edge is represented in the multiset by the ENA of the node at the bottom of the d-edge, SICs can be checked as follows. First we define the priority relation so that whenever the multiset contains an ENA at the bottom of some open edge it is not possible to select from that multiset an ENA that is in that d-edge's SIC. Second, not only is the current ENA encoded by the nonterminal symbol but we also store it in the multiset. Whenever a production for some ENA is applied we also specify that the corresponding ENA must be removed from the multiset. Thus, it is not possible to use a production for an ENA whose positioning at that point in the derivation violates a SIC. This explains the apparent redundancy in productions where an ENA is encoded both in the multiset and in the nonterminal. While LPMG nonterminals encode the nodes being visited, their productions insist that their ENAs are present in the multiset to ensure that they can be visited in the derivation.

From a DTG $G = (V_N, V_T, S, D)$ we construct a LPMG $G' = (V'_N, V_T, V_M, S, \sigma_0, P, \prec)$. Let $V_E$ be the set of ENAs of trees in $D$ whose members will be denoted $\eta$ with or without subscripts and primes. Let $V'_N = V_E \cup \{ S' \}$ and $V_M = V_E \cup \{ \sigma_0 \}$. $\rho$ is defined such that if $\eta_1$ is the d-edge daughter of some elementary node and $\eta_2$ is in the SIC associated with this d-edge then the pair $\langle \eta_1, \eta_2 \rangle$ is included in $\rho$. $P$ is defined as follows.

**Case 1:** As the root of a derived tree can correspond to the root of any elementary d-tree that is labelled by $S$, for each $\eta_r$ labelled by $S$ that is the root of some d-tree in $D$ let $S[\cdots \sigma_0 \cdots] \rightarrow \eta_r[\cdots \eta_r \cdots] \in P$.

**Case 2:** For each terminal node $\eta$ that is labelled $x$ let $\eta[\eta] \rightarrow x \in P$. This production ensures that when a terminal node is visited the only ENA in the multiset that can be present at this point must be that of the terminal node.

**Case 3:** For each $\eta$ and $\eta_r$ such that $\eta_r$ is the root of an elementary d-tree that (according to the SAC at $\eta$) can be left-sister-adjoined at $\eta$ let $\eta[\cdot\cdot\ \eta\ \cdot\cdot] \rightarrow \eta_r[\eta_r]\eta[\cdot\cdot\ \eta\ \cdot\cdot] \in P$. For each $\eta$ and $\eta_r$ such that $\eta_r$ is the root of an elementary d-tree that (according to the SAC at $\eta$) can be right-sister-adjoined at $\eta$ let $\eta[\cdot\cdot\ \eta\ \cdot\cdot] \rightarrow \eta[\cdot\cdot\ \eta\ \cdot\cdot]\eta_r[\eta_r] \in P$. The multiset associated with $\eta$ is not distributed to the element in the right-hand-side of the rule that corresponds to the d-tree being sister-adjoined.

**Case 4:** For each node $\eta$ in some d-tree in $D$ with i-edge daughters $\eta_1, \ldots, \eta_k$, where $k \geq 1$, let $\eta[\cdot\cdot\ \eta\ \cdot\cdot] \rightarrow \eta_1[\cdot\cdot\ \eta_1\ \cdot\cdot] \ldots \eta_k[\cdot\cdot\ \eta_k\ \cdot\cdot] \in P$. Here the multiset is to be distributed (in any manner) amongst the children, indicating that the open edges can be closed in the subtree below any of them.

**Case 5:** Suppose that $\eta_t$ is a node in some d-tree in $D$ with d-edge daughter $\eta_b$.

If $\eta_t$ and $\eta_b$ are labelled by the same symbol then let $\eta_t[\cdot\cdot\ \eta_t\ \cdot\cdot] \rightarrow \eta_b[\cdot\cdot\ \eta_b\ \cdot\cdot] \in P$.

For each $\eta$ that is labelled with the same symbol as $\eta_t$ and is the root of some elementary d-tree in $D$ let $\eta_t[\cdot\cdot\ \eta_t\ \cdot\cdot] \rightarrow \eta[\cdot\cdot\ \eta, \eta_b\ \cdot\cdot] \in P$.

For each $\eta$ that is labelled with the same symbol as $\eta_t$, is the root of some elementary component but is not the root of a d-tree let $\eta_t[\cdot\cdot\ \eta_t\ \cdot\cdot] \rightarrow \eta[\cdot\cdot\ \eta_b\ \cdot\cdot] \in P$.

The first production corresponds to the case where a component is not inserted within this d-edge. The latter two productions consider insertions at this d-edge. Note $\eta_b$ is added to the multiset at this point indicating that it will be the next node in its elementary d-tree that is to be visited. When the component (with root $\eta$) being inserted is not the topmost component of its elementary d-tree (third production) then $\eta$ must be found in the multiset and should not be added. Note that it will only be possible to apply a production for the nonterminal $\eta$ if $\eta$ is indeed in the multiset. On the other hand, when the component is the topmost component in its d-tree (second production) the multiset at this point in the LPMG derivation will not record this instance of this tree (as this is where we are considering the embedding of the d-tree for the first time), hence $\eta$ is also added to the multiset.

**Case 6:** Suppose that $\eta$ is a substitution node in some d-tree in $D$.

For each $\eta_r$ that is labelled with the same symbol as $\eta$ and is the root of an elementary d-tree in $D$ let $\eta[\cdot\cdot\ \eta\ \cdot\cdot] \rightarrow \eta_r[\cdot\cdot\ \eta_r\ \cdot\cdot] \in P$.

For each $\eta_r$ that is labelled with the same symbol as $\eta$, is the root of an elementary component but not the root of a d-tree in $D$ let $\eta[\cdot\cdot\ \eta\ \cdot\cdot] \rightarrow \eta_r[\cdot] \in P$. Any component (whether the topmost of an elementary d-tree or not) can be substituted at a substitution node provided their labels match. As in Case 5, we need to consider whether the component is the topmost component of its elementary d-tree.

The above construction has been oversimplified slightly. In order to incorporate the substitutability conditions described in the definition of DTG derivations we must check that each elementary d-tree is involved in only one subsertion. This can be done by using two forms of multiset symbols for each ENA: one that is used for nodes in d-trees above the node that will be substituted; and the other for nodes below the substituted node. The substitutability constraint can be enforced by allowing substitution only with nodes encoded by the first form of ENA and by changing from the first to the second form of ENA at this point. A second complication concerns the fact that the definition of SICs states that an ENA that is in a SIC at a d-edge between $\eta_t$ and $\eta_b$ cannot be place *properly* within the path from $\eta_t$ to $\eta_b$. With the use of extra nonterminals it is straightforward to capture this definition, but due to space restrictions we are unable to give details.

# 6 Earley-style DTG Parsing

We assume that the DTG from which LPMG $G = (V_N, V_T, V_M, S, \sigma_0, P, \rho)$ is constructed is lexicalized. Let the input string be $a_1 ... a_n$. The algorithm completes the $n+1$ item sets $I_0$, ..., $I_n$. Items in item sets have the form $((\omega_0 \rightarrow \omega_1 \cdot \omega_2, m_1), (m_2, i))$ where $\omega_0 \rightarrow \omega_1\omega_2$ is a LPMG production, $i$ is the index of the item set that introduced the production; and $m_1$ and $m_2$ are multisets. In predicting use of a LPMG production such as $A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow A_1[\cdot\cdot\ \sigma_1\ \cdot\cdot] \ldots A_k[\cdot\cdot\ \sigma_k\ \cdot\cdot]$, rather than considering all possible distributions of the multiset among the nonterminals on the right, we pass the entire multiset to the first subtree and propagate the multiset through the remaining subtrees in a top-down, left-to-right traversal of the derived tree. The underlying idea is that in an item $((\omega_0 \rightarrow \omega_1 \cdot \omega_2, m_1), (m_2, i)) \in I_j$ the multiset $m_2$ is the multiset at the time that we introduce the dotted rule $\omega_0 \rightarrow \cdot\omega_1\omega_2$ into item list $I_i$, and $m_1$ is the current value of the multiset that is the remainder to be passed onto subtrees not yet considered (corresponding to parts to the right of the dot in this dotted rule). The input is accepted if $((S[\cdot\cdot\ \sigma_0\ \cdot\cdot] \rightarrow \omega\cdot, \phi), (\sigma_0, 0)) \in I_n$.

**Initialization:** $((S[\cdot\cdot\ \sigma_0\ \cdot\cdot] \rightarrow \cdot\omega, \phi), (\sigma_0, 0)) \in I_0$
if $S[\cdot\cdot\ \sigma_0\ \cdot\cdot] \rightarrow \omega \in P$. Note, initially the multiset contains just $\sigma_0$. As the use of this rule will cause it to be removed, the empty multiset will be the current multiset that is passed to the descendant.

**Prediction (a):** $((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \cdot\omega, m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma), (m_1, i)) \in I_i$
if $((\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma_1, \ldots, \sigma_k\ \cdot\cdot]\omega_2, m_1), (m_2, j)) \in I_i$, $A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \omega \in P$, $\sigma$ is $\rho$-selectable from $m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ and $size(m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma) \leq n+1$. As indicated above, $m_1$ is the current value of the multiset, whereas the multiset to be passed onto the first descendant is obtained by adding $\sigma_1, \ldots, \sigma_k$ to $m_1$ and removing one occurrence of $\sigma$ from the resulting set. Note that the condition that $\sigma$ is $\rho$-selectable from $m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ ensures that $\sigma$ is in $m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$. Due to the assumption that the underlying DTG is lexicalized we limit the application of this predictive step to situations where $size(m_1 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma) \leq n+1$.

**Prediction (b):** $((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \cdot\omega, \phi \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma), (m_1, i)) \in I_i$
if $((\omega_0 \rightarrow \omega_1 \cdot A[\sigma_1, \ldots, \sigma_k]\omega_2, m_1), (m_2, j)) \in I_i$, $A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \omega \in P$, $\sigma$ is $\rho$-selectable from $\phi \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ and $k - 1 \leq n+1$. In this case we do not feed the entire multiset $m_1$ into the new production since the new production is "called" with a multiset containing just $\sigma_1, \ldots, \sigma_k$.

**Scanning:** $((\omega_0 \rightarrow \omega_1 A[\cdot\cdot\ \sigma\ \cdot\cdot] \cdot \omega_2, m_1), (m_2, j)) \in I_{i'}$ (resp. $((\omega_0 \rightarrow \omega_1 A[\sigma] \cdot \omega_2, m_1), (m_2, j)) \in I_{i'}$)
if $((\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma\ \cdot\cdot]\omega_2, m_1), (m_2, j)) \in I_i$ (resp. $((\omega_0 \rightarrow \omega_1 \cdot A[\sigma]\omega_2, m_1), (m_2, j)) \in I_i$), $A[\sigma] \rightarrow x \in P$ and $i = i'$ when $x = \epsilon$ and $i' = i+1$ when $x = a_{i+1}$.

**Completer (a):** $((\omega_0 \rightarrow \omega_1 A[\cdot\cdot\ \sigma_1, \ldots, \sigma_k\ \cdot\cdot] \cdot \omega_2, m_1), (m_3, l)) \in I_i$
if $((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \omega\cdot, m_1), (m_2, j)) \in I_i$, $m_1 \sqsubseteq m_2$, $((\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma_1, \ldots, \sigma_k\ \cdot\cdot]\omega_2, m_2), (m_3, l)) \in I_j$, $\sigma$ is $\rho$-selectable from $m_2 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ and $size(m_2 \oplus \sigma_1 \oplus \ldots \oplus \sigma_k \ominus \sigma) \leq n+1$. Before we considered the use of the production the multiset was $m_2$. Part of this has been used up in the derivation from $A$ and the remainder is $m_1$. This means we need to verify that $m_1 \sqsubseteq m_2$. Furthermore, because the use of this rule must have been predicted earlier (when we were considering $I_j$) then we must expect the presence of an item in $I_j$ containing a dotted rule of the form $\omega_0 \rightarrow \omega_1 \cdot A[\cdot\cdot\ \sigma_1, \ldots, \sigma_k\ \cdot\cdot]\omega_2$. In particular, the completed item expects that the current multiset in that item must be $m_2$.

**Completer (b):** $((\omega_0 \rightarrow \omega_1 A[\sigma_1, \ldots, \sigma_k] \cdot \omega_2, m_1), (m_2, l)) \in I_i$
if $((A[\cdot\cdot\ \sigma\ \cdot\cdot] \rightarrow \omega\cdot, \phi), (m_1, j)) \in I_i$, $((\omega_0 \rightarrow \omega_1 \cdot A[\sigma_1, \ldots, \sigma_k]\omega_2, m_1), (m_2, l)) \in I_j$, $\sigma$ is $\rho$-selectable from $\phi \oplus \sigma_1 \oplus \ldots \oplus \sigma_k$ and $k \leq n+1$. The dot is moved over a nonterminal that is associated with a fixed multiset. Thus, the multiset associated with the completed production must be empty and the multiset $m_1$ remains intact since it was not fed into the completed production (see the second case of the predictor step).

A LPMG parse forest can be extracted from the completed item sets in the usual way. Since there is a direct correspondence between a group of LPMG productions and DTG composition

operations, it is possible to recover the derivation graphs, and therefore SA-trees, of the underlying DTG from the corresponding derivation tree of the constructed LPMG grammar. Thus, the LPMG parse forest provides a reasonable encoding of the set of DTG derivations for the input string. In one respect, the LPMG parse forest is particularly compact since there is a one-to-many mapping from LPMG derivation trees to DTG derivation graphs. When reconstructing a DTG derivation graph from a LPMG derivation tree it is necessary to establish which occurrences of ENAs (occurring in the multisets at nodes of the LPMG derivation tree) should be associated with the same occurrence of a d-tree in the DTG derivation. Because no distinction is made between different occurrences of the same multiset symbol in a multiset, there may be several ways of associating occurrences of a multiset symbols at different nodes in the derivation tree. Thus, it is possible that for a given LPMG derivation tree there will be several ways of making the correspondence of occurrences of multiset symbols to occurrences of elementary d-trees. This is attractive because for every possible way of making the correspondence there will be a legal DTG derivation. Thus, a single LPMG derivation tree is compactly encoding a set of DTG derivations.

The item sets contain tuples of the form $((\omega_0 \rightarrow \omega_1 \cdot \omega_2, m_1), (m_2, i))$ where $0 \leq i \leq n$ and $n$ is input length. The number of items in a item set depends on the number of possible multisets. Since we assume that the underlying DTG grammar is lexicalized the number of open d-edges at any node in the derived tree is bounded by the length of the input string. The construction presented in Section 5 is such that the size of any multiset used in a derivation by a LPMG thus constructed is bounded by $n+1$. The number of such multisets is bounded by $O(n^k)$, where is $k$ is the total number of d-edges in the elementary d-trees of the grammar. Thus, the number of items in a item set is bounded by $O(n^{2k+1})$. The completer step dominates the running time of the algorithm since for each of the $O(n^{2k+1})$ items in $I_i$ we consider $O(n^{2k+1})$ items in $I_j$. Thus the running time of this step is $O(n^{4k+2})$. Since there are $n+1$ item sets the total worst-case running time of the algorithm is $O(n^{4k+3})$. Note that the running time of the recognizer is sensitive to the number of open d-edges that arise in derivations and that in some applications (such as with grammars of English) this number may be very small (perhaps as low as 1 or 2).

# References


T. Becker & O. Rambow. 1994. Parsing free word order languages. In *3rd TAG Workshop.*

T. Becker & O. Rambow. 1995. Parsing non-immediate dominance relations. IWPT'95.

T. Becker, A. Joshi, & O. Rambow. 1991. Long distance scrambling and Tree Adjoining Grammars. In EACL'91.

S. Billot & B. Lang. 1989. The structure of shared forests in ambiguous parsing. In ACL'89.

A. Joshi & Y. Schabes. 1991. Tree-Adjoining Grammars and lexicalized grammars. In M. Nivat & A. Podelski, eds., *Definability and Recognizability of Sets of Trees.*

B. Lang. 1991. A uniform framework for parsing. In M. Tomita, ed., *Current Issues in Parsing Technology.*

B. Lang. 1994. Recognition can be harder than parsing. *Computat. Intelligence.*, 10(4).

O. Rambow. 1994a. *Formal and Computational Aspects of Natural Language Syntax.* Ph.D. thesis, Dept. CIS, Univ. Penn.

O. Rambow, K. Vijay-Shanker, & D. Weir. 1995. D-Tree Grammars. In ACL'95.

O. Rambow. 1994b. Multiset-valued Linear Index Grammars. In ACL'94.

Y. Schabes & S. Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computat. Ling.*, 20(1).

S. Shieber. 1985. Evidence against the context-freeness of natural language. *Ling. & Philos.*, 8.

K. Vijay-Shanker & D. Weir. 1993a. Parsing some constrained grammar formalisms. *Computat. Ling.*, 19(4).

K. Vijay-Shanker & D. Weir. 1993b. The use of shared forests in TAG parsing. In EACL'93.

K. Vijay-Shanker & D. Weir. 1994. The equivalence of four extensions of Context-Free Grammars. *Math. Syst. Theory*, 27.

# THE INFLUENCE OF TAGGING ON THE RESULTS OF PARTIAL PARSING IN GERMAN CORPORA

Oliver Wauschkuhn
University of Stuttgart, Institut für Informatik
Breitwiesenstr. 20–22
D-70565 Stuttgart, Germany
e-mail: wauschkuhn@informatik.uni-stuttgart.de

## 1 Introduction

In the last two years good progress has been made in work on part-of-speech tagging for German, while for a long time it was focussed on the English language. Now, that German corpora can be tagged with quite a high accuracy (cf. [Schmid 95]), we have to think about for which applications tagging can be used, and how tagged corpora are to be used and processed.

One possibility is the use of a tagger as preprocessor for a partial syntactic analysis of textual corpora. Parsing a sentence results in many cases in a (more or less) great number of ambiguities, which are to be resolved in subsequent analysis steps. To reduce the number of result trees in an earlier stage a tagger could be used to disambiguate the morphological and morphosyntactic part-of-speech annotations of the input for the parser.

This paper describes an examination of the influence of tagging on subsequent partial parsing in German corpora with respect to the number of analysis trees. The underlying question is, to what extent tagging can serve as a preprocessor for a partial syntactic analysis to reduce the number of ambiguities in the parse results. For this, two small corpora have been partially syntactically analyzed, each in a tagged and an untagged form, and the results between these forms are compared with respect to their number of trees.

The term *tagged* corpus(-form)/sentence means in this context, that (most of) its part-of-speech tags are unambiguous. Words of a few classes constitute an exception being annotated with one additional tag if a *structural* analysis is needed to disambiguate them (for more details, see chapter 2). A corpus(-form)/sentence, on the other hand, is called *untagged*, if its words are annotated ambiguously with all tags that result from a morphological analysis without considering their context. This is equal to the input of a tagger.

The next chapter makes some more detailed remarks on the corpus material used for this work and on the part-of-speech annotation. In the third chapter the parsing method for partially analyzing the sentences is explained. In chapter 4 the quantitative results of the different analyses are presented and compared. The last chapter summarizes these results and draws a conclusion with respect to the influence of tagging on the parsing results.

## 2 The Corpus Material

The corpus material used for this evaluation is a small part of the German "Stuttgarter Zeitungs–Corpus", henceforth called StZ-Corpus. The whole corpus consists of the issues of two years of the German newspaper "Stuttgarter Zeitung" and contains ~36 million tokens, building ~1.8 million sentences. It has been tagged with the Xerox Part-of-Speech Tagger (cf. [Cutting et al. 92]) adapted to the German language (cf. [Schmid 93]).

One text for this work, consisting of 1097 sentences, is a hand-tagged part of the StZ-Corpus, which served as training- and test-corpus for the adaption of the tagger to German texts. Consequently, its tags are (nearly) 100% correct.

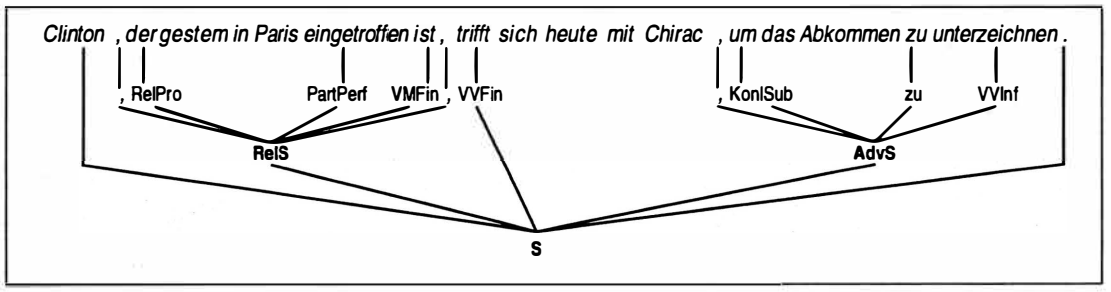



**Figure 1:** Example for the tree structure resulting from the clause-analysis step.

The other text with 3776 sentences is taken from the statistically tagged part of the StZ-Corpus. The tag error rate of the German version of the Xerox tagger can be taken from [Schmid 95] and [SchmidKempe 95] as between 3.5 and 4%. With respect to the whole corpus these values should be seen merely as a lower bound for the error rate, because the text the tagger was tested on was taken from the same corpus part as the training text, and has therefore a similar style of sentence constructions.

The tagset the StZ-Corpus is tagged with consists of 71 tags representing syntactic categories, which are grouped into the following 12 main categories: noun, adjective, cardinal number, verb, determiner, pronoun, adverb, conjunction, adposition (preposition, postposition, etc.), structural word, interjection, punctuation mark. For detailed information about this tagset, see [Schiller 94], and an overview is given in [SchillerThielen 95]. In addition to these syntactic tags, the hand tagged text is also annotated with morphosyntactic feature information, like case, gender, number, etc.

Auxiliaries and modal verbs are *never* annotated as main verbs in the tagged StZ-Corpus, even if they have such a function in the sentence. Generally, a bigram tagger, which considers only the left context of the word to be annotated, cannot decide between these two purposes of auxiliaries and modal verbs: in a German main clause construction, an auxiliary/modal verb and the corresponding main verb often do not appear adjacently—i.e. they build discontinuous constituents—, and the main verb stands in the right context of the corresponding auxiliary/modal verb.

As the grammars for the partial parsing process make use of the distinction between an auxiliary/modal use of an auxiliary/modal verb and the main verb use, all those verbs in the tagged corpora had to be annotated with an additional main verb tag before the partial syntactic analysis.

## 3 The Partial Parsing Process

### 3.1 Method

For the domain of a partial syntactic analysis of German textual corpora we developed a method that divides the parse process into two steps. The task of the first step is to detect all single clauses of the possibly complex input sentence, i.e. main clauses, subordinate clauses and infinitive constructions. Their hierarchical order (coordination, subordination) is here of little interest. This analysis step is henceforth also called *clause-analysis*. An example for the result structure of this step is shown in Figure 1 for the following sentence:[1]

> *Clinton, der gestern in Paris eingetroffen ist, trifft sich heute mit Chirac, um das Abkommen zu unterzeichnen.*
> *(Clinton, who arrived in Paris yesterday, meets Chirac today to sign the agreement.)*

It consists of a main clause (S) (*"Clinton trifft sich heute mit Chirac."*), a relative clause (RelS) (*", der gestern in Paris eingetroffen ist,"*) and an adverbial clause (AdvS) (*", um das Abkommen zu unterzeichnen"*).

[1]The translation is given in parentheses.

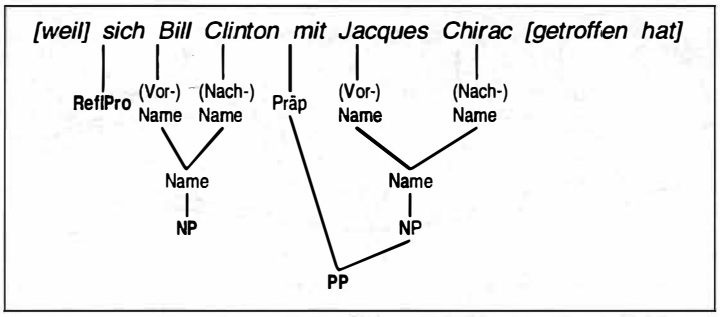



**Figure 2:** Example for the tree structure resulting from the NP-analysis step.

In the second step, the minimal NPs and PPs[2] within each single clause of the complete sentence are to be detected without making any decision about their hierarchical order (coordination, subordination). I.e., inside a single clause all NPs and PPs are on the same hierarchical level. Henceforth, this step is also referred to as *NP-detection* or *NP-analysis*. An example of the result of this parse step is given in Figure 2; the adverbial clause

*[weil] sich Bill Clinton mit Jacques Chirac [getroffen hat]*
*([because] Bill Clinton [has met] Jacques Chirac)*

consists of a reflexive pronoun (ReflPro) (*"sich"*), a noun phrase (NP) (*"Bill Clinton"*) and a prepositional phrase (PP) (*"mit Jacques Chirac"*).

To parse a sentence partially, these two steps are applied successively: First, the clause-analysis is done and then the NP-detection is applied to every single clause of the result of the first step. The second step is not carried out for sentences that do not yield a result in the clause-analysis step. The combination of the outcome of all these parses for a sentence builds the final result of the partial analysis.

The division of the parsing process into the clause- and the NP-analysis is especially adequate for German sentence constructions. Complex sentences are well structured into clauses by means of punctuation marks (especially commas) and structural words (especially different types of conjuctions), and simple minimal NPs do not extend clause limits. In addition, the independence between these parse steps makes the method more robust. A detailed description of this stepwise parsing strategy is given in [Wauschkuhn 94].

### 3.2 Realization

The parse process is realized by means of a chart parser (CHAPLIN)[3], which applies sets of syntactic rules (grammars) on annotated input sequences. The grammars are of a context-free type and their rules can additionally define morphosyntactic feature restrictions between the right-hand side categories. The input for an analysis is an ordered set of categorially and morphosyntactically annotated wordforms.

The development of the grammar for the first analysis step is in an advanced stadium, while the grammar of the NP-detection consists mainly of a set of "basic rules" and has to be further developed and refined.

The clause-analysis grammar consists of 229 phrase structure rules, the one for the NP-detection of 180.

## 4 Results from the Parsing Process

### 4.1 Description of the Experiment

Both texts described in chapter 2—the hand-tagged one, consisting of 1097 sentences, henceforth also referred to as *Corpus 1*, and the statistically tagged one with 3776 sentences, henceforth

[2] noun phrases and prepositional phrases

[3] The parser is implemented in COMMON LISP and described in detail in [BurkertLöthe 95].

also called *Corpus 2*—have been analyzed in the following way:

Each corpus was partially parsed twice along the two-step method, once in the tagged form[4] and once in the untagged form. The single words of the untagged corpus versions were analyzed with two German morphology systems (GERMMORPHAN, implemented by E. Lehmann, University of Stuttgart, and MORPH, implemented by G. Hanrieder, FORWISS[5], Erlangen) and then were annotated with the resulting categories and morphosyntactic features. So, the words of the untagged version of Corpus 2 bore a bit more information than in the tagged form, namely the morphosyntactic features.

Since (partial) syntactically (hand-)annotated *German* corpora—like the Penn Treebank for the English language—are not yet (freely) available, it was impossible to automatically evaluate parsing results for quality. Therefore, a quantitative evaluation was made considering the number of parse trees resulting from the syntactic analysis of the input sequences. The results for each corpus form (tagged and untagged) of both corpora are represented in detail within Tables 1a–4b in the appendix. Each table holds for the different numbers of resulting parse trees (column 1), the corresponding number of input sequences that yields this tree number (column 2), and its percentage with respect to the complete number of inputs (column 3).

In addition, it has been examined for the clause-analysis step, how many sentences yield the same quantitative non-ambiguous parse result (i.e. 0 and 1 tree, respectively) for *both* input forms, tagged and untagged. This amount compared to the number of sentences that yield the same result for their untagged form represents a measure for the disambiguating quality of the tagger for subsequent partial parsing: this proportion indicates the relative amount (percentage) of sentences that were annotated with such parts-of-speech by the tagger that they yield the same non-ambiguous parse result (0 or 1 tree) as in their untagged form. So, the same percentage for selecting the "correct" tags with respect to a subsequent syntactic analysis can be expected for the other cases where the parse process yields more than one tree. This measure of quality can be put into numbers through the quotient of the amount of sentences having the same quantitative analysis result for both corpus versions and the amount of sentences yielding that result for the untagged version. These values are represented in Table 6 in the appendix.

The input type for the clause-analysis is a sentence and the input for the second step is a sequence of words from a single clause that is limited to the left and to the right by elements of type "verb" or "clause boundary". Thus, in most cases for successfully clause-analyzed sentences, more than one NP-analysis is to be made.

Both analysis steps are evaluated seperately for this work. If a clause-analysis results in more than one parse tree, the second step is evaluated only for the simple clauses of *one* of the results of step 1. Thus each sentence part is NP-analyzed only one time and is not considered more than once in the results.

To guarantee that the parser won't get stuck while analyzing a word sequence and building the corresponding parse forest[6] for counting the number of result trees, two interruption criteria are defined: *a)* a time limit is defined at which the computation is interrupted, and *b)* for the untagged corpus versions input sequences of more than 50 elements for the first step and of more than 11 elements for the second step are ignored, i.e. not analyzed. Those cases are listed in the tables in the first row "interruption".

### 4.2 Results of each Analysis

Now that all conditions of the parse experiment have been explained, the results of the analyses for the different corpus types are to be presented and compared.

First, the results of the *clause-analyses* of the four corpus types (Corpus 1 and 2, each in the tagged and the untagged version) will be described and discussed and after that the results of the *NP-detection*. All quantitative results are summarized in Tables 1a–4b. The comparisons are made between tagged and untagged version of each corpus.

[4] As mentioned, the auxiliaries and modal verbs are co-tagged as main verbs in the tagged corpus versions.

[5] Bayerisches Forschungszentrum für Wissensbasierte Systeme

[6] The time- and space-critical phases of an analysis of "hard" input sequences is generally ***not*** the parse process itself, i.e. the filling of the chart, but the construction of the parse forest from the chart.

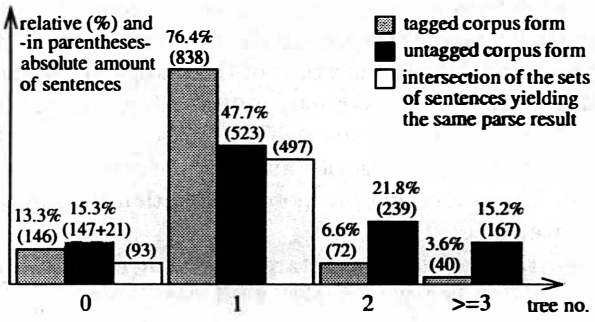



**Chart 1:** Amounts of sentences resulting in 0, 1, 2, and ≥3 parse trees in the clause-analysis step for Corpus 1.

The essential analysis results relevant to the following discussions are represented in several bar charts (Chart 1–5). In addition to the amounts of resulting parse trees, they show for the first analysis step the amounts of sentences that yield the same quantitative result for the tagged and the untagged corpus version. In the charts for the NP-analysis the absolute result values (tree numbers) are not contained, because the number of analyzed input sequences differs between the tagged and the untagged corpus versions. Thus, only the relative comparison is of interest.

### Clause-Analysis

*Corpus 1 — The Hand-Tagged Text*

All quantitative results of the clause-analysis of the hand-tagged corpus are given in detail in Tables 1a and 1b for the tagged and the untagged version respectively; Chart 1 gives an overview. Both corpus forms have nearly the same amount of sentences that are not covered by the grammar for this parse step, namely *13.3%* and *13.4%*. In addidion, for the untagged version the parse process was interrupted or not carried out for *1.9%* of the input[7], leading to a rate of *15.3%* for not successfully parsed sentences.

The intersection of the set of tagged sentences that lead to no parse result and the corresponding set of untagged sentences contains only *63.2%* of the untagged set of unparsable sentences (see also Table 5 in the appendix).

Actually, one would expect a much higher correspondence between these sets of input, especially, because they have nearly the same number of elements (146 and 147 respectively): If a parser finds a parse tree for an unambiguously annotated (i.e. tagged) sentence, it should also find one or more results for it in the ambiguously annotated (untagged) version, because the disambiguated tags should be contained in the non-disambiguated tags. And if the parser doesn't find a parse tree for an ambiguously annotated sentence, how should a result be found for the disambiguated form?

There are two reasons for this difference between the sets of unparsable sentences: First, the hand-tagging of the corpus was not done on the basis of the results of the morphology systems used to analyze the untagged version, and second, even the morphology systems make a few mistakes, eg. for some unknown verbforms.

Considering the successful analysis results presumes a great advantage of tagging a corpus before parsing it with respect to the number of results: *76.4%* of the sentences in the hand-tagged corpus form led to an unambiguous analysis result while the corresponding rate for the untagged version is only *47.7%*. But in this case, it must be taken into account that the tagged corpus form has an error rate of nearly 0% due to manual annotation.

In contrast to the unsuccessful parses, the comparison of the sets of sentences (tagged vs. untagged) yielding one analysis result confirms the expectation: *95.0%* of the untagged sentences led also in their tagged form to one unambiguous result (cf. also Table 5).

[7]This concerns sentences with more than 50 elements or needing a computation time of more than 5 minutes.

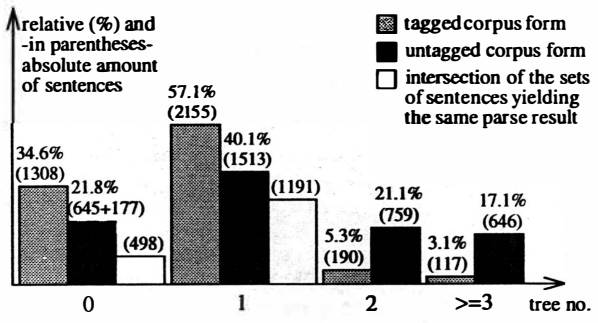



**Chart 2:** Amounts of sentences resulting in 0, 1, 2, and $\geq 3$ parse trees in the clause-analysis step for Corpus 2.

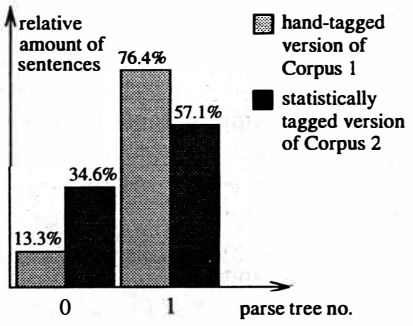



**Chart 3:** Comparison of quantitative analysis results between manually and statistically tagged sentences.

More than one parse tree was found for *10.2%* of the hand-tagged sentences (*6.6%* for 2 trees + *3.6%* for 3 or more) and for a relatively great part of the untagged sentences, namely *37.0%* (*21.8%* + *15.2%*).

*Corpus 2 — The Statistically Tagged Text*

Tables 2a and 2b hold detailled quantitative results for the clause-analysis of Corpus 2; see Chart 2 for an overview. *34.6%* of the statistically tagged sentences were not covered by the grammar for the first analysis step while this holds for only *17.1%* of the untagged sentences, plus *4.7%* for interrupted analyses. This great difference between the results of tagged and untagged input resulting in 0 parse trees didn't arise for the manually annotated Corpus 1 and is caused for a certain part by wrong tags produced by the statistical tagger (see below).

Comparing the sets of tagged and untagged sentences yielding an unsuccessful parse result shows that *77.2%* of the untagged sentences are also contained in the unparsable tagged sentence set (see Table 5).

Considering the unambiguous successful results (i.e., exactly one parse tree was found) also emphasizes the disambiguating quality of tagging with regard to the syntactic analysis, however, not to such an extent as in the hand-tagged case: *57.1%* of non-ambiguous results for the statistically tagged version versus *40.1%* for the untagged. Similary to the unsuccessful parses, *78.7%* of the unambiguously analyzed untagged sentences yielded also in the tagged form one parse tree.

The cases for 2 or more parse trees resemble to the results for Corpus 1.

*Hand-Tagged vs. Statistically Tagged Input*

A comparison between the clause-analysis results of the manually and the statistically tagged corpus versions (see Chart 3 for an overview and Tables 1a and 2a respectively for detailed results) shows that in the latter version *21.3%* more sentences resulted in 0 parse trees than in the former, while on the other hand the percentage of successfully unambiguously parsed sentences is less by a similar amount (*19.3%*).

This shift from successful unambiguous parse results to unsuccessful analyses between these corpus versions supports the statement from above that a certain part of the unsuccessfully parsed sentences is caused by wrong tags relevant to this analysis step. A closer look at the first 30 of all 1309 non-parsable sentences of the (statistically) tagged form of Corpus 2 led to the following result: for 21 sentences (*70.0%*) the failure was due to one wrong tag, 7 sentences (*23.3%*) were not covered by the grammar and 2 sentences (*6.7%*) had both shortcomings.

However, even if the rate is worse, statistical tagging helps to reduce the number of ambiguities in the results of subsequent partial parsing.

*Some Remarks on the Imbalance in the Results Between Tagged and Untagged Corpus Forms*

To explain the imbalance in the results between tagged and untagged input, one must have a look at the categories that are relevant to the first analysis step (i.e. for the detection of single clauses in complex sentences). The main group of these are verbs and function words. Many German function words have more than one possible syntactic category depending on their use in clauses/sentence constructions. Thus, the morphology produces more than one tag for them. Some examples:

*"zu"* → infinitive introductory structural word, preposition, separable verb prefix
*"und"* → phrase coordinating conjunction, clause coordinating conjunction
*"wie"* → preposition, comparative conjunction, clause subordin. conjunction, interrogative adverb

A tagger reduces these part-of-speech ambiguities by considering the near context of the words which in turn reduces the number of possible syntactic structures. On the other hand, the tagger is not able to decide generally about parts-of-speech that are based on long-distance dependencies, which can cause wrong parse results.

## NP-Analysis

As already mentioned above, the development of the grammar for the second analysis step is less advanced than for the first step, which can also be derived from the success rates as well as from the amount of the following quantitative parse results. Therefore, these shall be seen merely with regard to a *comparison* between tagged and untagged input.

*Corpus 1 — The Hand-Tagged Text*

Quantitative results for the NP-analysis of the hand-tagged corpus are given in Tables 3a for the tagged and in 3b for the untagged form; Chart 4 gives an overview. A striking feature of the tagged version is the high rate of *41.4%* of not successfully parsed input sequences, while for the untagged corpus form this rate is at *22.4%*, plus *8.2%* for interrupted parses.

This great amount of unparsable hand-tagged input is in parts caused by the bad tuning of the NP-grammar to the tagset the corpus is annotated with. The grammar is tuned to the morphology results used in the untagged input, which have for example a finer distinction in the noun categories than the tagset of the tagger. While in the latter there exists only the category noun, the morphological analysis distinguishes between uncountable nouns (eg. *rice, butter, water*), nouns representing a quantity unit (eg. *litre, million*), and other "normal" nouns. Thus, in the grammar, a number expression followed by a noun can only combine to a NP, if the noun is an uncountable noun, a quantity unit or in plural number; but that combination is not possible with a singular "normal" noun.

For the case of successfully unambiguously analyzed input sequences quite a high rate of *52.0%* could be found for the hand-tagged corpus form while the corresponding rate for untagged input is much less, namely *18.6%*. On the other hand, the great amount of *50.8%* of untagged input sequences yielded an ambiguous parse result; *39.6%* produced 3 or more trees and even *13.5%* 10 or more. This high ambiguity rate for untagged input is above all due to the various functions German determiner words can have (*"der", "die", "das"* → determiner, relative pronoun, demonstrative pronoun; *"eine", "einen"* → determiner, indefinite pronoun), and the fact that nouns—especially not so often used ones—are frequently classified also as names by the morphology. Such ambiguities produce a great variety of different potential chains of NPs for the input sequences.

Especially for these cases tagging is a useful tool to highly reduce the number of parse trees, because dependencies between categories relevant to the NP-analysis extend mainly over a short distance. However, these results still have to be examined with respect to quality, i.e. correctness of the annotations.

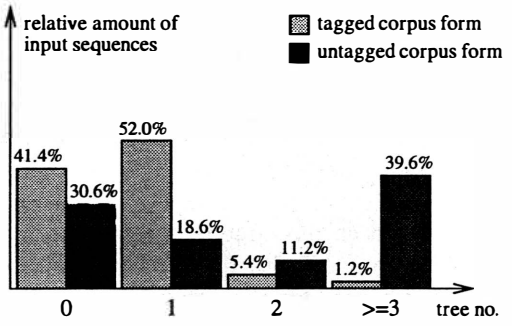



**Chart 4:** Amounts of sentences resulting in 0, 1, 2, and ≥3 parse trees in the NP-analysis step for Corpus 1.

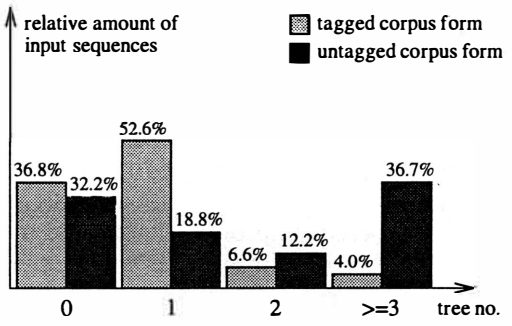



**Chart 5:** Amounts of sentences resulting in 0, 1, 2, and ≥3 parse trees in the NP-analysis step for Corpus 2.

*Corpus 2 — The Statistically Tagged Text*

Considering the results of the NP-analysis of the statistically tagged corpus in its tagged and untagged versions (Tables 4a and 4b respectively and Chart 5), a great similarity to the quantitative parse results of Corpus 1 can be noticed. The only difference that is worth mentioning is, that the amount of unsuccessfully parsed sentences (0 trees) is *4.6%* smaller for the statistically tagged corpus form than for the hand-tagged one (Corpus 1), and that this part has been shifted to the results of 2 or more analysis trees. This shift can be explained by the lack of morphosyntactic annotations in the statistically tagged corpus.[8] Due to this lack, the great number of feature restrictions in the NP-grammar is not considered in the parse process, so that more phrase structure rules are applied than actually match the syntactic constructions. This results in a greater number of syntax trees having a certain error rate.

Since all other results are similar to those of the NP-analyses of Corpus 1, nothing new can be said upon the comparison between the parse results of the tagged and the untagged corpus version.

## 5 Summary and Conclusion

A very conspicuous property among the quantitative results is that for each *tagged* corpus version in both analysis steps the amount of unambiguously parsed input sequences lies over *50%*—for the clause-analysis of hand-tagged input even over *75%*—, while on the other hand, the correponding rate for *untagged* input is about *40%* and *47%* for the clause-analysis and below *20%* for the NP-analysis.

Considering the amounts of unsuccessful parse results (i.e. 0 trees) shows in most cases a much higher rate for tagged than for untagged input.

A conclusion from these facts and numbers is that tagging helps to reduce the number of parse trees but at the cost of increasing the rate of unsuccessful parses.

For lack of manually syntactically annotated German corpora, parse results couldn't be proved for correctness on a large scale automatically. Therefore no details can be given about the *quality* of the disambiguating character of the tagger for a subsequent syntactic analysis, which in turn depends on the correctness of the part-of-speech annotations.

However, the syntactic results were examined "indirectly" at least for the clause-analysis step by comparing the sets of sentences that result in the same number of parse trees (for 0 and 1) between tagged and untagged input. For the case of Corpus 2 *78,7%* of sentences that yielded in the clause-analysis exactly one parse tree for their tagged form resulted in the same tree number for their statistically tagged version. For this amount of sentences the tagger has choosen the "correct"[9] parts-of-speech relevant to the first analysis step, and a similar

[8] Recall that the untagged version of Corpus 2 is in contrast morphosyntactically annotated for the parse process.

[9] "correct" only with respect to yield a parse result.

percentage of correctness can be expected for the other statistically annotated sentences.

Regarding the relatively high rate of unsuccessful parse results for tagged input, two reasons can be given: One problem is the error rate of the tagger of at least 3.5–4%, which has been confimed by the small examination of the tags of the non-clause-analysable sentences of the statistically tagged corpus, where in *70%* of the sentences a wrong tag caused the non-parsability. The second problem effecting the NP-analysis is the lacking adaption of the NP-grammar to the tags of the tagger.

In both cases improvements can be made: In the tagged corpus, frequently repeated tag errors must be recognized, and either the tagger is to be improved with respect to these cases or those tags are to be co-tagged with all possible parts-of-speech for the corresponding word. For the NP-grammar, refinements and extensions in the rule set are to be made.

To use a tagger as preprocessor for syntactical analyses, only such parts-of-speech should be disambiguated with it that can be decided on the basis of the considered context. For a bigram- or trigram-tagger this is the case for short-distance dependencies, and long-distance phenomena like combinations of an auxiliary with the corresponding participle are to be resolved by the parser. Therefore, a tagger is especially useful to disambiguate such words that are relevant to the partial NP-analysis: their parts-of-speech depend in many cases on adjacent words/categories, and furthermore, as mentioned above (4.2: NP-Analysis), these words can produce a great amount of syntactic ambiguities.

To conclude I want to give a proposal on how to use a tagger as a preprocessor for parsing a textual corpus:

1. For the case of a successful syntactic analysis of a tagged sentence, the analysis result should be accepted as a disambiguated parse forest.

2. Tagging errors increase the number of unsuccessful parses. In such a case, i.e., if the syntactic analysis of a tagged sentence results in 0 trees, this sequence should be reparsed in its untagged form.

This method makes use of a tagger as a tool for disambiguating syntactic results only in the advantageous cases. Otherwise the tagger results are ignored.

## References


[BurkertLöthe 95] Gerrit Burkert and Mathis Löthe. CHAPLIN – Ein Chart Parser für Linguistische Experimente. Technical Report, Institut für Informatik, University of Stuttgart, 1995, forthcoming.

[Cutting et al. 92] Doug Cutting and Julian Kupiec and Jan Pedersen and Penelope Sibun. A Practical Part-of-Speech Tagger. In *Proceedings of the third Conference on Applied Natural Language Processing*, Trento, Italy, April 1992.

[Schiller 94] Anne Schiller. Guidelines für das Tagging deutscher Textcorpora (Kleines und erweitertes Tagset). Technical Report, Institut für maschinelle Sprachverarbeitung (IMS), University of Stuttgart, 1994.

[SchillerThielen 95] Anne Schiller and Christine Thielen. Ein kleines und erweitertes Tagset fürs Deutsche. In *Tagungsberichte des Arbeitstreffens "Lexikon + Text", Schloß Hohentübingen, 1994.* Niemeyer, Tübingen, 1995.

[Schmid 93] Helmut Schmid. Tagging German with the Xerox-Tagger. Technical Report, Institut für maschinelle Sprachverarbeitung (IMS), University of Stuttgart, 1993.

[Schmid 95] Helmut Schmid. Improvements in Part-of-Speech Tagging with an Application to German. In *Proceedings of EACL SIGDAT-Workshop, Dublin, Ireland, 1995.*

[SchmidKempe 95] Helmut Schmid and André Kempe. Tagging von Corpora mit HMM, Entscheidungsbäumen und Neuronalen Netzen. In *Tagungsberichte des Arbeitstreffens "Lexikon + Text", Schloß Hohentübingen, 1994.* Niemeyer, Tübingen, 1995.

[Wauschkuhn 94] Oliver Wauschkuhn. Mehrstufiges Parsing zur syntaktischen Analyse von Textcorpora. Technical Report, Institut für Informatik, University of Stuttgart, 1994.

## Appendix: Tables

| *no. of parse trees* | *no. of sentences* | *relative amount of sentences* |
|---|---|---|
| interrupt. | 1 | 0.0% |
| 0 | 146 | 13.3% |
| 1 | 838 | 76.4% |
| 2 | 72 | 6.6% |
| 3 | 11 | 1.0% |
| 4 | 11 | 1.0% |
| 5 | 1 | 0.0% |
| 6–9 | 10 | 0.9% |
| 10–19 | 6 | 0.5% |
| 20–49 | 1 | 0.0% |
| ≥50 | 0 | 0.0% |
| *sum* | *1097* | |

**Table 1a:** Hand-tagged input.

| *no. of parse trees* | *no. of sentences* | *relative amount of sentences* |
|---|---|---|
| interrupt. | 21 | 1.9% |
| 0 | 147 | 13.4% |
| 1 | 523 | 47.7% |
| 2 | 239 | 21.8% |
| 3 | 41 | 3.7% |
| 4 | 46 | 4.2% |
| 5 | 3 | 0.3% |
| 6–9 | 42 | 3.8% |
| 10–19 | 27 | 2.5% |
| 20–49 | 8 | 0.7% |
| ≥50 | 0 | 0.0% |
| *sum* | *1097* | |

**Table 1b:** Untagged input.

**Absolute and relative amounts of sentences in relation to the number of parse trees they result in for the clause-analysis of Corpus 1.**

| *no. of parse trees* | *no. of sentences* | *relative amount of sentences* |
|---|---|---|
| interrupt. | 6 | 0.2% |
| 0 | 1308 | 34.6% |
| 1 | 2155 | 57.1% |
| 2 | 190 | 5.3% |
| 3 | 23 | 0.6% |
| 4 | 34 | 0.9% |
| 5 | 9 | 0.2% |
| 6–9 | 27 | 0.7% |
| 10–19 | 16 | 0.4% |
| 20–49 | 8 | 0.2% |
| ≥50 | 0 | 0.0% |
| *sum* | *3776* | |

**Table 2a:** Statistically tagged input.

| *no. of parse trees* | *no. of sentences* | *relative amount of sentences* |
|---|---|---|
| interrupt. | 177 | 4.7% |
| 0 | 645 | 17.1% |
| 1 | 1513 | 40.1% |
| 2 | 795 | 21.1% |
| 3 | 140 | 3.7% |
| 4 | 222 | 5.9% |
| 5 | 31 | 0.8% |
| 6–9 | 125 | 3.3% |
| 10–19 | 76 | 2.0% |
| 20–49 | 45 | 1.2% |
| ≥50 | 7 | 0.2% |
| *sum* | *3776* | |

**Table 2b:** Untagged input.

**Absolute and relative amounts of sentences in relation to the number of parse trees they result in for the clause-analysis of Corpus 2.**

| *no. of parse trees* | *no. of input sequ.* | *relative amount of input sequ.* |
|---|---|---|
| interrupt. | 0 | 0.0% |
| 0 | 1244 | 41.4% |
| 1 | 1562 | 52.0% |
| 2 | 162 | 5.4% |
| 3 | 8 | 0.3% |
| 4 | 17 | 0.6% |
| 5 | 1 | 0.0% |
| 6–9 | 6 | 0.2% |
| 10–19 | 3 | 0.1% |
| 20–49 | 1 | 0.0% |
| 50–99 | 0 | 0.0% |
| ≥100 | 0 | 0.0% |
| *sum* | *3004* | |

**Table 3a:** Hand-tagged input.

| *no. of parse trees* | *no. of input sequ.* | *relative amount of input sequ.* |
|---|---|---|
| interrupt. | 230 | 8.2% |
| 0 | 629 | 22.4% |
| 1 | 524 | 18.6% |
| 2 | 315 | 11.2% |
| 3 | 388 | 13.8% |
| 4 | 120 | 4.3% |
| 5 | 27 | 1.0% |
| 6–9 | 199 | 7.1% |
| 10–19 | 198 | 7.0% |
| 20–49 | 106 | 3.8% |
| 50–99 | 50 | 1.8% |
| ≥100 | 27 | 1.0% |
| *sum* | *2813* | |

**Table 3b:** Untagged input.

**Absolute and relative amounts of input sequences in relation to the number of parse trees they result in for the NP-analysis of Corpus 1.**

| *no. of parse trees* | *no. of input sequ.* | *relative amount of input sequ.* |
|---|---|---|
| interrupt. | 2 | 0.0% |
| 0 | 2804 | 36.8% |
| 1 | 4011 | 52.6% |
| 2 | 504 | 6.6% |
| 3 | 84 | 1.1% |
| 4 | 116 | 1.5% |
| 5 | 30 | 0.4% |
| 6–9 | 55 | 0.7% |
| 10–19 | 17 | 0.2% |
| 20–49 | 3 | 0.0% |
| 50–99 | 0 | 0.0% |
| ≥100 | 0 | 0.0% |
| *sum* | *7626* | |

**Table 4a:** Statistically tagged input.

| *no. of parse trees* | *no. of input sequ.* | *relative amount of input sequ.* |
|---|---|---|
| interrupt. | 802 | 8.5% |
| 0 | 2220 | 23.7% |
| 1 | 1765 | 18.8% |
| 2 | 1148 | 12.2% |
| 3 | 1170 | 12.5% |
| 4 | 356 | 3.8% |
| 5 | 82 | 0.9% |
| 6–9 | 684 | 7.3% |
| 10–19 | 528 | 5.6% |
| 20–49 | 337 | 3.6% |
| 50–99 | 162 | 1.7% |
| ≥100 | 129 | 1.4% |
| *sum* | *9383* | |

**Table 4b:** Untagged input.

**Absolute and relative amounts of input sequences in relation to the number of parse trees they result in for the NP-analysis of Corpus 2.**

| | *Corpus 1* | | *Corpus 2* | |
|---|---|---|---|---|
| | *0 trees* | *1 tree* | *0 trees* | *1 tree* |
| compared sentence numbers | 93 of 147 | 497 of 523 | 498 of 645 | 1191 of 1513 |
| relative amount by division | 63.2% | 95.0% | 77.2% | 78.7% |

**Table 5:** Amounts of sentences that yield the same number of parse trees (in the clause-analysis step) for both input forms, tagged and untagged (for 0 and 1 tree), compared to the amount of sentences that result in this tree number for the untagged form.

# Partitioning Grammars and Composing Parsers

**Fuliang Weng and Andreas Stolcke**[1]

**Speech Technology and Research Laboratory**
**SRI international, 333 Ravenswood Ave., Menlo Park, CA 94025**
**{fuliang,stolcke}@speech.sri.com**

Formal and natural languages may not be homogeneous in the sense that no single style parser can do well for any languages or sublanguages. Natural languages as a whole do not belong to any of the subclasses of CFG with efficient and compact parsers. On the other hand, natural language exhibits distinct sublanguages for various phrase types, idioms, etc., which can conveniently be described by specialized sub-grammars in various formalisms.

From a theoretical point of view, there are grammars $G$ that lead to LR(k) parsers with size exponential in the size of $G$, given by Earley, 1968 and Ukkonen, 1982.

$S \rightarrow A_i \quad (1 \le i \le n)$
$A_i \rightarrow a_j A_i \, (1 \le i \ne j \le n)$
$A_i \rightarrow a_i B_i | b_i \quad (1 \le i \le n)$
$B_i \rightarrow a_j B_i | b_i \, (1 \le i, j \le n)$

If we divide this grammar into $n$ sub-grammars based on $i$ and compile them separately, however, we can get LR(k) parsers with a polynomial size, though its recognition time complexity increases to $\sqrt{|G|}$ times of the original one.

From a practical point of view, in a large parser, sublanguages may be described by pre-existing sub-grammars, possibly using specialized parsing algorithms. Combining sub-grammars may also be needed to extend existing grammars in the future. Therefore, a framework for partitioning grammars and combining parsers would prove useful.

The main points in this paper are: a general schema for partitioning a grammar into sub-grammars, and the combination of parsers for sub-grammars into an overall parser that yields the same parses as one for the original full grammar. Formal definitions for grammar partitioning will be presented, along with a correctness theorem, and a parser composition in the context of GLR parsing framework is presented.

Given $G = (\Sigma, NT, P, S)$ is a CFG, $P = \cup P_i$ where $P_i \cap P_j = \phi, i \ne j$ and $p_S \in P_0$. Without loss of generality, we assume in the rest of the paper that there is only one production rule with $S$ as its LHS symbol.

Definition 1: $\{G_i = (\Sigma_i, NT_i, P_i, \nabla)\}_{i=0}^{n}$ is called a partition of $G$, where $NT_i = \{LHS(p) | p \in P_i\}$.

Definition 2: Let $P_A = \{p | p \in P \wedge LHS(p) = A\}$, $\{A | A \in NT, A \in RHS(p), p \in P_i$ and $P_A - P_i \ne \phi\}$ is called the *input* of $G_i$, $INPUT_{G_i}$, and $\{A | A \in LHS(p), p \in P_i, (\exists j) A \in INPUT_{G_j}\}$ is called the *output* of $G_i$, $OUTPUT_{G_i}$; for $G_0$, $OUTPUT_{G_0}$ has an additional element $S$. The *INPUT* is those NTs used by a sub-grammar that were previously parsed by other sub-grammars. The *OUTPUT* is those NTs that are the result of parsing by a sub-grammar and are handed on to other sub-grammars for further parsing.

Definition 3: Let $\{G_i = (\Sigma_i, NT_i, P_i, \nabla)\}_{i=0}^{n}$ be a partition of $G$, $\{G_i^X = (\Sigma_i^o, NT_i^o, P_i^o, X)\}_{i=0}^{n}$ is called the sub-grammar set of $G$ with respect to partition $\{G_i\}_{i=0}^{n}$, where $\Sigma_i^o = \Sigma_i \cup \{vtm_A | A \in INPUT_{G_i}\}$, $NT_i^o = NT_i \cup \{A | A \in INPUT_{G_i}\}$, $P_i^o = P_i \cup \{A \rightarrow vtm_A | A \in INPUT_{G_i}\}$[2], $X \in OUTPUT_{G_i}$. In particular, $G_0^S = (\Sigma_0^o, NT_0^o, P_0^o, S)$ is called the master sub-grammar. A directed calling graph for the sub-grammar set of $G$ is defined as $(V, E)$, where $V$ is the sub-grammar set and $E = (A, B)$, where the start symbol of sub-grammar $B$ is in the *INPUT* of sub-grammar $A$.

---

1. The authors would like to thank P. Price and M. Cohen of STAR lab/SRI for their support.

2. It is a modified version of the virtual terminal technique used by Korenjak, 1969 and Weng, 1993.

Definition 4: A derivation of the sub-grammar set $\{G_i^S\}_{i=0}^n$ of $G$ is one of the following:

1. $\alpha A \beta \Rightarrow \alpha\gamma\beta$ if $\exists i$ such that $A \rightarrow \gamma \in P_i^o$.
2. $\alpha vtm_A \beta \Rightarrow \alpha A \beta$ if $\exists i$ such that $A \in OUTPUT_{G_i}$.

Theorem 1: A string of terminals $\alpha$ is derived from $G$ iff it can be derived from the sub-grammar set $\{G_i^S\}_{i=0}^n$, starting from the master sub-grammar $G_0^S$.

For simplicity, we assume that $G$ is partitioned into two subsets, a master sub-grammar $G_S$ with the start symbol $S$ and a slave sub-grammar $G_A$ with the start symbol A. Any number of slave grammars can be accommodated with trivial modifications as long as the calling graph between sub-grammars is a DAG. Let $\{A\}$ be the output of $G_A$ and the input of $G_S$, and S the output of $G_S$. Compiling $G_A$ and $G_S$, by using a GLR(0) parser generator, we obtain four tables: a pair of Action and Goto tables for each of $G_A$ and $G_S$. Based on (Tomita, 1986), we modify the GLR parsing process as follows:

Starting $parser_{G_S}$ from its initial state and initial position of 1 in an input string of length $n$:

1. In $parser_{G_S}$ create register($i$) for the $i$-th word in the current input; initialize register(*) to $\phi$ at the beginning of parsing, that is, in $PARSE(G_S, a_1, \ldots, a_n)$ of Tomita's algorithm.

2. In the $PARSEWORD(i)$ process of Tomita's algorithm, instead of initializing $Q$ (a place to hold shift action) to $\phi$, assign to $Q$ the value of register($i$), that possibly stores node-state pairs $\langle v, s \rangle$ given by sub-parser(s) $parser_{G_A}$, where $v$ is the root node of a forest returned by $parser_{G_A}$ and $s$ is a state that $parser_{G_S}$ will go to after shifting in node $v$. In other words, $parser_{G_S}$ delayed this shift action until position $i$ when its sub-parser gets there.

3. When $parser_{G_S}$ reaches state $s$ and looks at terminal $t \in \Sigma$ at position $j$ of the current input, do normal GLR parsing using $G_S$.

4. When $parser_{G_S}$ reaches its state $s$ and scans position $j$ of the input, $Action_{G_S}(s, vtm_A) \neq \phi$:

(a) If $\langle reduce, i \rangle \in Action_{G_S}(s, vtm_A)$, do it as in the normal GLR parsing by using $G_S$.

(b) If $\langle shift, i \rangle \in Action_{G_S}(s, vtm_A)$, switch the control to $parser_{G_A}$ and start the parsing process of $G_A$ from the initial state of $G_A$ and position $j$ until *accept* is reached at position $k \leq n$ in the input. Hang the whole forest under $A$, which has $vtm_A$ as its son in the partially parsed forest created by $G_S$, and eliminate this $vtm_A$. Put $\langle shift, i \rangle$ in register($k$) and continue this process until $parser_{G_A}$ reaches the end of the input, then return the control to $parser_{G_S}$.

With a synchronization mechanism, the DAG restriction on the calling graph for the algorithm can be removed.

Compared with other work, the grammar partitioning presented here gives more freedom as to how the grammar is divided, and can be generalized to combine parsers of different styles. This is different from Korenjak's partitioning grammars and constructing LR(k) processors, and also from Abney's chunking parser.

**References**


S. Abney, *Parsing by Chunks*, In *Principle-Based Parsing: Computation and Psycholinguistics*, R. C. Berwick et al. (eds.), Kluwer Academic Publishers, 1991

J. Earley, *An Efficient Context-free Parsing Algorithm*, Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, 1968.

A. Korenjak, *A Practical Method for Constructing LR(k) Processors*, *CACM* 12 (11), 1969.

M. Tomita, *Efficient Parsing for Natural Language*, Kluwer Academic Publishers, 1986.

E. Ukkonen, *Lower Bounds on the Size of Deterministic Parsers*, *J. Computer and System Sciences* 26, 153-170, 1983.

F. Weng, *Handling Syntactic Extra-grammaticality*, In *Proceedings of the 3rd International Workshop on Parsing Technologies*, Tilburg, the Netherlands, 1993.

# PARSING WITH TYPED FEATURE STRUCTURES

Shuly Wintner

Nissim Francez

Computer Science
Technion, Israel Institute of Technology
32000 Haifa, Israel
{shuly,francez}@cs.technion.ac.il

**Abstract**

In this paper we provide for parsing with respect to grammars expressed in a general TFS-based formalism, a restriction of ALE ([2]). Our motivation being the design of an abstract (WAM-like) machine for the formalism ([14]), we consider parsing as a computational process and use it as an operational semantics to guide the design of the control structures for the abstract machine.

We emphasize the notion of **abstract typed feature structures** (AFSs) that encode the essential information of TFSs and define unification over AFSs rather than over TFSs. We then introduce an explicit construct of **multi-rooted feature structures** (MRSs) that naturally extend TFSs and use them to represent phrasal signs as well as grammar rules. We also employ abstractions of MRSs and give the mathematical foundations needed for manipulating them. We then present a simple bottom-up chart parser as a model for computation: grammars written in the TFS-based formalism are executed by the parser. Finally, we show that the parser is correct.

## 1 Introduction

Typed feature structures (TFSs) serve for the specification of linguistic information in current linguistic formalisms such as HPSG ([10]) or Categorial Grammar ([8]). They can represent lexical items, phrases and rules. Usually, no mechanism for manipulating TFSs (e.g., parsing algorithm) is inherent to the formalism. Current approaches to processing HPSG grammars either translate them to Prolog (e.g., [2, 5, 6]) or use a general constraint system ([16]).

In this paper we provide for parsing with grammars expressed in a general TFS-based formalism, a restriction of ALE ([2]). Our motivation is the design of an abstract (WAM-like) machine for the formalism ([14]); we consider parsing as a computational process and use it as an operational semantics to guide the design of the control structures for the abstract machine. In this paper the machine is not discussed further.

Section 2 outlines the theory of TFSs of [1, 3]. We emphasize **abstract typed feature structures** (AFSs) that encode the essential information of TFSs and extend unification to AFSs. Section 3 introduces an explicit construct of **multi-rooted feature structures** (MRSs) that naturally extend TFSs, used to represent phrasal signs as well as grammar rules. Abstraction is extended to MRSs and the mathematical foundations needed for manipulating them is given. In section 4 a simple bottom-up chart parser for the TFS-based formalism is presented and shown correct. The appendix contains examples of MRSs and grammars as well as a simulation of parsing. Due to space limitations we replace many proofs by informal descriptions and examples; the formal details are given in [15]. The main contributions of this paper are:

- Formalization and explication of the notion of multi-rooted feature structures that are used implicitly in the computational linguistics literature;
- Concise definitions of a TFS-based linguistic formalism, based on abstract MRSs;
- Specification and correctness proofs for parsing in this framework.

# 2 Theory of Feature Structures

## 2.1 Types, Features and Feature Structures

We assume familiarity with the theory of TFS as in [3, chapters 1-6], and only summarize some of its preliminary notions. When dealing with partial functions the symbol '$f(x)\downarrow$' means that $f$ is defined for the value $x$ and the symbol '$\uparrow$' means undefinedness. Whenever the result of an application of a partial function is used as an operand, it is meant that the function is defined for its arguments.

For the following discussion, fix non-empty, finite, disjoint sets TYPES and FEATS of types and feature names, respectively. Let PATHS $=$ FEATS$^*$ denote the collection of **paths**, where FEATS is totally ordered. Fix also an infinite set NODES of nodes and a typing function $\theta$ : NODES $\rightarrow$ TYPES. The set NODES is 'rich' in the sense that for every $t \in$ TYPES, the set $\{q \in$ NODES $\mid \theta(q) = t\}$ is infinite. We use the bounded complete partial order $\sqsubseteq$ over TYPES $\times$ TYPES to denote the **type hierarchy**, and the partial function *Approp* : FEATS $\times$ TYPES $\rightarrow$ TYPES to denote the **appropriate specification**.

A **feature structure** is a directed, connected, labeled graph consisting of a finite, nonempty set of nodes $Q \subseteq$ NODES, a root $\bar{q} \in Q$, and a partial function $\delta : Q \times$ FEATS $\rightarrow Q$ specifying the arcs such that every node $q \in Q$ is accessible from $\bar{q}$. We overload '$\sqsubseteq$' to denote also subsumption of feature structures. Two feature structures $A_1$ and $A_2$ are **alphabetic variants** ($A_1 \sim A_2$) iff $A_1 \sqsubseteq A_2$ and $A_2 \sqsubseteq A_1$.

Alphabetic variants have exactly the same structure, and corresponding nodes have the same types. Only the identities of the nodes distinguish them. The essential properties of a feature structure, excluding the identities of its nodes, can be captured by three components: the set of paths, the type assigned to every path, and the sets of paths that lead to the same node. In contrast to other approaches (e.g., [3]), we first define abstract feature structures and then show their relation to concrete ones. The representation of graphs as sets of paths is inspired by works on the semantics of concurrent programming languages, and the notion of fusion-closure is due to [4].

**Definition 2.1 (Abstract feature structures)** *A **pre- abstract feature structure** (pre-AFS) is a triple* $\langle \Pi, \Theta, \approx \rangle$, *where*

- $\Pi \subseteq$ PATHS *is a non-empty set of paths*
- $\Theta : \Pi \rightarrow$ TYPES *is a total function, assigning a type to every path*
- $\approx \subseteq \Pi \times \Pi$ *is a relation specifying reentrancy (with* $[\approx]$ *the set of its equivalence classes)*

*An **abstract feature structure** (AFS) is a pre-AFS for which the following requirements hold:*

- $\Pi$ *is prefix-closed: if* $\pi\alpha \in \Pi$ *then* $\pi \in \Pi$ *(where* $\pi, \alpha \in$ PATHS*)*
- $A$ *is fusion-closed: if* $\pi\alpha \in \Pi$ *and* $\pi'\alpha' \in \Pi$ *and* $\pi \approx \pi'$ *then* $\pi\alpha' \in \Pi, \Theta(\pi\alpha') = \Theta(\pi'\alpha')$ *(as well as* $\pi'\alpha \in \Pi, \Theta(\pi'\alpha) = \Theta(\pi\alpha)$*), and* $\pi\alpha' \approx \pi'\alpha'$ *(as well as* $\pi'\alpha \approx \pi\alpha$*)*
- $\approx$ *is an equivalence relation with a finite index*
- $\Theta$ *respects the equivalence: if* $\pi_1 \approx \pi_2$ *then* $\Theta(\pi_1) = \Theta(\pi_2)$

An AFS $\langle \Pi, \Theta, \approx \rangle$ is *well-typed* if $\Theta(\pi) \neq \top$ for every $\pi \in \Pi$ and if $\pi f \in \Pi$ then $Approp(f, \Theta(\pi))\downarrow$ and $Approp(f, \Theta(\pi)) \sqsubseteq \Theta(\pi f)$. It is *totally well typed* if, in addition, for every $\pi \in \Pi$, if $Approp(f, \Theta(\pi))\downarrow$ then $\pi f \in \Pi$.

Abstract features structures can be related to concrete ones in a natural way: If $A = (Q, \bar{q}, \delta)$ is a TFS then $Abs(A) = \langle \Pi_A, \Theta_A, \approx_A \rangle$ is defined by:

- $\Pi_A = \{\pi \mid \delta(\bar{q}, \pi)\downarrow\}$
- $\Theta_A(\pi) = \theta(\delta(\bar{q}, \pi))$
- $\pi_1 \approx_A \pi_2$ iff $\delta(\bar{q}, \pi_1) = \delta(\bar{q}, \pi_2)$

It is easy to see that $Abs(A)$ is an abstract feature structure.

For the reverse direction, consider an AFS $A = \langle \Pi, \Theta, \approx \rangle$. First construct a 'pseudo-TFS', $Conc(A) = (Q, \bar{q}, \delta)$, that differs from a TFS only in that its nodes are not drawn from the set NODES. Let $Q = \{q_{[\pi]} \mid [\pi] \in [\approx]\}$. Let $\theta(q_{[\pi]}) = \Theta(\pi)$ for every node – since $A$ is an AFS, $\Theta$ respects the equivalence and therefore $\theta$ is representative-independent. Let $\bar{q} = q_{[\epsilon]}$ and $\delta(q_{[\pi]}, f) = q_{[\pi f]}$ for every node $q_{[\pi]}$ and feature $f$. Since $A$ is fusion-closed, $\delta$ is representative-independent. By injecting $Q$ into NODES making use of the richness on NODES, a concrete TFS $Conc(A)$ is obtained, representing the equivalence class of alphabetic variants that can be obtained that way. We abuse the notation $Conc(A)$ in the sequel to refer to this set of alphabetic variants.

**Theorem 2.2** *If $A' \in Conc(A)$ then $Abs(A') = A$.*

AFSs can be partially ordered: $\langle \Pi_A, \Theta_A, \approx_A \rangle \preceq \langle \Pi_B, \Theta_B, \approx_B \rangle$ iff $\Pi_A \subseteq \Pi_B, \approx_A \subseteq \approx_B$ and for every $\pi \in \Pi_A, \Theta_A(\pi) \sqsubseteq \Theta_B(\pi)$. This order corresponds to the subsumption ordering on TFSs, as the following theorems show.

**Theorem 2.3** *$A \sqsubseteq B$ iff $Abs(A) \preceq Abs(B)$.*

**Theorem 2.4** *For every $A \in Conc(A'), B \in Conc(B'), A \sqsubseteq B$ iff $A' \preceq B'$.*

**Corollary 2.5** *$A \sim B$ iff $Abs(A) = Abs(B)$.*

**Corollary 2.6** *$Conc(A') \sim Conc(B')$ iff $A = B$.*

## 2.2 Unification

As there exists a one to one correspondence between AFSs and (alphabetic variants of) concrete ones, we define unification over AFSs. This leads to a simpler definition that captures the essence of the operation better than the traditional definition. We use the term 'unification' to refer to both the operation and its result.

**Definition 2.7 (Closure operations)** *Let $Cl$ be a fusion-closure operation on pre-AFSs: $Cl(A) = A'$, where $A'$ is the least extension of $A$ to a fusion-closed structure. Let $Eq(\langle \Pi, \Theta, \approx \rangle) = \langle \Pi, \Theta, \approx' \rangle)$ where $\approx'$ is the least extension of $\approx$ to an equivalence relation. Let $Ty(\langle \Pi, \Theta, \approx \rangle) = \langle \Pi, \Theta', \approx \rangle$ where $\Theta'(\pi) = \bigsqcup_{\pi' \approx \pi} \Theta(\pi)$.*

**Definition 2.8 (Unification)** *The unification $A \sqcup B$ of two AFSs $A = \langle \Pi_A, \Theta_A, \approx_A \rangle$ and $B = \langle \Pi_B, \Theta_B, \approx_B \rangle$ is an AFS $C' = Ty(Eq(Cl(C)))$, where:*

- $C = \langle \Pi_C, \Theta_C, \approx_C \rangle$
- $\Pi_C = \Pi_A \cup \Pi_B$
- $\Theta_C(\pi) = \begin{cases} \Theta_A(\pi) \sqcup \Theta_B(\pi) & \text{if } \pi \in \Pi_A \text{ and } \pi \in \Pi_B \\ \Theta_A(\pi) & \text{if } \pi \in \Pi_A \text{ only} \\ \Theta_B(\pi) & \text{if } \pi \in \Pi_B \text{ only} \end{cases}$

- $\approx_C = \approx_A \cup \approx_B$

*The unification* ***fails*** *if there exists a path* $\pi \in \Pi_{C'}$ *such that* $\Theta_{C'}(\pi) = \top$.

**Lemma 2.9** *Cl preserves prefixes: If A is a prefix-closed pre-AFS and* $A' = Cl(A)$ *then* $A'$ *is prefix-closed.*

**Lemma 2.10** *Eq preserves prefixes and fusions: If A is a prefix- and fusion-closed pre-AFS and* $A' = Eq(A)$ *then* $A'$ *is prefix- and fusion-closed.*

**Corollary 2.11** *If A and B are AFSs, then so is* $A \sqcup B$.

$C'$ is the smallest AFS that contains $\Pi_C$ and $\approx_C$. Since $\Pi_A$ and $\Pi_B$ are prefix-closed, so is $\Pi_C$. However, $\Pi_C$ and $\approx_C$ might not be fusion-closed. This is why $Cl$ is applied to them. As a result of its application, new paths and equivalence classes might be added. By lemma 2.9, if a path is added all its prefixes are added, too, so the prefix-closure is preserved. Then, $Eq$ extends $\approx$ to an equivalence relation, without harming the prefix- and fusion-closure properties (by lemma 2.10). Finally, $Ty$ sees to it that $\Theta$ respects the equivalences.

**Lemma 2.12** *Unification is commutative:* $A \sqcup B = B \sqcup A$.

**Lemma 2.13** *Unification is associative:* $(A \sqcup B) \sqcup C = A \sqcup (B \sqcup C)$.

The result of a unification can differ from any of its arguments in three ways: paths that were not present can be added; the types of nodes can become more specific; and reentrancies can be added, that is, the number of equivalence classes of paths can decrease. Consequently, the result of a unification is always more specific than any of its arguments.

**Theorem 2.14** *If* $C' = A \sqcup B$ *then* $A \preceq C'$.

TFSs (and therefore AFSs) can be seen as a generalization of first-order terms (FOTs) (see [1]). Accordingly, AFS unification resembles FOT unification; however, the notion of *substitution* that is central to the definition of FOT unification is missing here, and as far as we know, no analog to substitutions in the domain of feature structures was ever presented.

# 3 Multi-rooted Structures

To be able to represent complex linguistic information, such as phrase structure, the notion of feature structures has to be extended. HPSG does so by introducing special features, such as DTRS (daughters), to encode trees in TFSs. This solution requires a declaration of the special features, along with their intended meaning; such a declaration is missing in [10]. An alternative technique is employed by Shieber ([11]): natural numbers are used as special features, to encode the order of daughters in a tree. In a typed system this method necessitates the addition of special types as well; theoretically, the number of features and types necessary to state rules is unbounded.

As a more coherent, mathematically elegant solution, we define multi-rooted structures, naturally extending TFSs. These structures provide a means to represent phrasal signs and grammar rules. They are used implicitly in the computational linguistics literature, but to the best of our knowledge no explicit, formal theory of these structures and their properties was formulated before.

**Definition 3.1 (Multi-rooted structures)** *A* ***multi-rooted feature structure*** *(MRS) is a pair* $\langle \bar{Q}, G \rangle$ *where* $G = \langle Q, \delta \rangle$ *is a finite, directed, labeled graph consisting of a set* $Q \subseteq$ NODES *of nodes and a partial function* $\delta : Q \times$ FEATS $\rightharpoonup Q$ *specifying the arcs, and where* $\bar{Q}$ *is an ordered, non-empty (repetition-free) list of distinguished nodes in Q called* ***roots****. G is not necessarily connected, but the union of all the nodes reachable from all the roots in* $\bar{Q}$ *is required to yield exactly Q. The* ***length*** *of a MRS is the number of its roots,* $|\bar{Q}|$.

Meta-variables $\sigma, \rho$ range over MRSs, and $\delta, Q$ and $\bar{Q}$ over their constituents. If $\langle \bar{Q}, G \rangle$ is a MRS and $\bar{q}_i$ is a root in $\bar{Q}$ then $\bar{q}_i$ naturally induces a feature structure $Pr(\bar{Q}, i) = (Q_i, \bar{q}_i, \delta_i)$, where $Q_i$ is the set of nodes reachable from $\bar{q}_i$ and $\delta_i = \delta|_{Q_i}$.

One can view a MRS $\langle \bar{Q}, G \rangle$ as an ordered sequence $\langle A_1, \ldots, A_n \rangle$ of (not necessarily disjoint) feature structures, where $A_i = Pr(\bar{Q}, i)$ for $1 \leq i \leq n$. Note that such an ordered list of feature structures is not a sequence in the mathematical sense: removing an element from the list effects the other elements (due to reentrancy among elements). Nevertheless, we can think of a MRS as a sequence where a subsequence is obtained by taking a subsequence of the roots and considering only the feature structures they induce. We use the two views interchangeably. Figure 1 depicts a MRS and its view as a sequence of feature structures.

A MRS is well-typed if all its constituent feature structures are well-typed, and is totally well-typed if all its constituents are. Subsumption is extended to MRSs as follows:

**Definition 3.2 (Subsumption of multi-rooted structures)** *A MRS $\sigma = \langle \bar{Q}, G \rangle$ subsumes a MRS $\sigma' = \langle \bar{Q}', G' \rangle$ (denoted by $\sigma \sqsubseteq \sigma'$) if $|\bar{Q}| = |\bar{Q}'|$ and there exists a total function $h : Q \rightarrow Q'$ such that:*

- *for every root $\bar{q}_i \in \bar{Q}, h(\bar{q}_i) = \bar{q}'_i$*
- *for every $q \in Q$, $\theta(q) \sqsubseteq \theta'(h(q))$*
- *for every $q \in Q$ and $f \in$ FEATS, if $\delta(q, f)\downarrow$ then $h(\delta(q, f)) = \delta'(h(q), f)$*

We define abstract multi-rooted structures in an analog way to abstract feature structures.

**Definition 3.3 (Abstract multi-rooted structures)** *A pre- abstract multi rooted structure (pre-AMRS) is a quadruple $A = \langle Ind, \Pi, \Theta, \approx \rangle$, where:*

- *$Ind$, the indices of $A$, is the sequence $\langle 1, \ldots, n \rangle$ for some $n$*
- *$\Pi \subseteq Ind \times$ PATHS is a non-empty set of indexed paths, such that for each $i \in Ind$ there exists some $\pi \in$ PATHS that $(i, \pi) \in \Pi$.*
- *$\Theta : \Pi \rightarrow$ TYPES is a total type-assignment function*
- *$\approx \subseteq \Pi \times \Pi$ is a relation*

*An **abstract multi-rooted structure** (AMRS) is a pre-AMRS $A$ for which the following requirements, naturally extending those of AFSs, hold:*

- *$\Pi$ is prefix-closed: if $(i, \pi\alpha) \in \Pi$ then $(i, \pi) \in \Pi$*
- *$A$ is fusion-closed: if $(i, \pi\alpha) \in \Pi$ and $(i', \pi'\alpha') \in \Pi$ and $(i, \pi) \approx (i', \pi')$ then $(i, \pi\alpha') \in \Pi$ (as well as $(i', \pi'\alpha) \in \Pi$), and $(i, \pi\alpha') \approx (i', \pi'\alpha')$ (as well as $(i', \pi'\alpha) \approx (i, \pi\alpha)$)*
- *$\approx$ is an equivalence relation*
- *$\Theta$ respects the equivalence: if $(i_1, \pi_1) \approx (i_2, \pi_2)$ then $\Theta(i_1, \pi_1) = \Theta(i_2, \pi_2)$*

An AMRS $\langle Ind, \Pi, \Theta, \approx \rangle$ is well-typed if for every $(i, \pi) \in \Pi$, $\Theta(i, \pi) \neq \top$ and if $(i, \pi f) \in \Pi$ then $Approp(f, \Theta(i, \pi))\downarrow$ and $Approp(f, \Theta(i, \pi)) \sqsubseteq \Theta(i, \pi f)$. It is totally well typed if, in addition, for every $(i, \pi) \in \Pi$, if $Approp(f, \Theta(i, \pi))\downarrow$ then $(i, \pi f) \in \Pi$. The **length** of an AMRS $A$ is $len(A) = |Ind_A|$.

The closure operations $Cl$ and $Eq$ are naturally extended to AMRSs: If $A$ is a pre-AMRS then $Cl(A)$ is the least extension of $A$ that is prefix- and fusion-closed, and $Eq(A)$ is the least extension of $A$ to a pre-AMRS in which $\approx$ is an equivalence relation. In addition, $Ty(\langle Ind, \Pi, \Theta, \approx \rangle) = \langle Ind, \Pi, \Theta', \approx \rangle$ where $\Theta'(i, \pi) = \bigsqcup_{(i', \pi') \approx (i, \pi)} \Theta(i', \pi')$. The partial order $\preceq$ is extended to AMRSs: $\langle Ind_A, \Pi_A, \Theta_A, \approx_A \rangle \preceq \langle Ind_B, \Pi_B, \Theta_B, \approx_B \rangle$ iff $Ind_A = Ind_B, \Pi_A \subseteq \Pi_B, \approx_A \subseteq \approx_B$ and for every $(i, \pi) \in \Pi_A, \Theta_A(i, \pi) \sqsubseteq \Theta_B(i, \pi)$.

AMRSs, too, can be related to concrete ones in a natural way: If $\sigma = \langle \bar{Q}, G \rangle$ is a MRS then $Abs(\sigma) = \langle Ind_\sigma, \Pi_\sigma, \Theta_\sigma, \approx_\sigma \rangle$ is defined by:

- $Ind_\sigma = \langle 1, \ldots, |\bar{Q}| \rangle$
- $\Pi_\sigma = \{(i, \pi) \mid \delta(\bar{q}_i, \pi)\downarrow\}$
- $\Theta_\sigma(i, \pi) = \theta(\delta(\bar{q}_i, \pi))$
- $(i, \pi_1) \approx_\sigma (j, \pi_2)$ iff $\delta(\bar{q}_i, \pi_1) = \delta(\bar{q}_j, \pi_2)$

It is easy to see that $Abs(\sigma)$ is an AMRS. In particular, notice that for every $i \in Ind_\sigma$ there exists a path $\pi$ such that $(i, \pi) \in \Pi_\sigma$ since for every $i, \delta(\bar{q}_i, \epsilon)\downarrow$. The reverse operation, $Conc$, can be defined in a similar manner.

AMRSs are used to represent ordered collections of AFSs. However, due to the possibility of value sharing among the constituents of AMRSs, they are not sequences in the mathematical sense, and the notion of sub-structure has to be defined in order to relate them to AFSs.

**Definition 3.4 (Sub-structures)** *Let $A = \langle Ind_A, \Pi_A, \Theta_A, \approx_A \rangle$; let $Ind_B$ be a finite (contiguous) subsequence of $Ind_A$; let $n+1$ be the index of the first element of $Ind_B$. The sub-structure of $A$ induced by $Ind_B$ is an AMRS $B = \langle Ind_B, \Pi_B, \Theta_B, \approx_B \rangle$ such that:*

- $(i - n, \pi) \in \Pi_B$ *iff* $i \in Ind_B$ *and* $(i, \pi) \in A$
- $\Theta_B(i - n, \pi) = \Theta_A(i, \pi)$ *if* $i \in Ind_B$
- $(i_1 - n, \pi_1) \approx_B (i_2 - n, \pi_2)$ *iff* $i_1 \in Ind_B, i_2 \in Ind_B$ *and* $(i_1, \pi_1) \approx_A (i_2, \pi_2)$

A sub-structure of $A$ is obtained by selecting a subsequence of the indices of $A$ and considering the structure they induce. Trivially, this structure is an AMRS. We use $A^{j..k}$ to refer to the sub-structure of $A$ induced by $\langle j, \ldots, k \rangle$. If $Ind_B = \{i\}$, $A^{i..i}$ can be identified with an AFS, denoted $A^i$.

The notion of concatenation has to be defined for AMRSs, too:

**Definition 3.5 (Concatenation)** *The concatenation of $A = \langle Ind_A, \Pi_A, \Theta_A, \approx_A \rangle$ and $B = \langle Ind_B, \Pi_B, \Theta_B, \approx_B \rangle$ of lengths $n_A, n_B$, respectively (denoted by $A \cdot B$), is an AMRS $C = \langle Ind_C, \Pi_C, \Theta_C, \approx_C \rangle$ such that*

- $Ind_C = \langle 1, \ldots, n_A + n_B \rangle$
- $\Pi_C = \Pi_A \cup \{(i + n_A, \pi) \mid (i, \pi) \in \Pi_B\}$
- $\Theta_C(i, \pi) = \begin{cases} \Theta_A(i, \pi) & \text{if } i \leq n_A \\ \Theta_B(i - n_A, \pi) & \text{if } i > n_A \end{cases}$
- $\approx_C = \approx_A \cup \{((i_1 + n_A, \pi_1), (i_2 + n_A, \pi_2)) \mid (i_1, \pi_1) \approx_B (i_2, \pi_2)\}$

We now extend the definition of unification to AMRSs: we want to allow the unification of two $AFSs$, one of which is a part of an AMRS. Therefore, one operand is a pair consisting of an AMRS and an index, specifying some element of it, and the second operand is an AFS. Recall that due to reentrancies, other elements of the AMRS can be affected by this operation. Therefore, the result of the unification is a new AMRS.

**Definition 3.6 (Unification in context)** *Let $A = \langle Ind_A, \Pi_A, \Theta_A, \approx_A \rangle$ be an AMRS, $B = \langle \Pi_B, \Theta_B, \approx_B \rangle$ an AFS. $(A, j) \sqcup B$ is defined if $j \in Ind_A$, in which case it is the AMRS $C' = Ty(Eq(Cl(\langle Ind_C, \Pi_C, \Theta_C, \approx_C \rangle)))$, where*

- $Ind_C = Ind_A$
- $\Pi_C = \Pi_A \cup \{(j, \pi) \mid \pi \in \Pi_B\}$
- $\Theta_C(i, \pi) = \begin{cases} \Theta_A(i, \pi) & \text{if } i \neq j \\ \Theta_A(i, \pi) \sqcup \Theta_B(\pi) & \text{if } i = j \text{ and } (i, \pi) \in \Pi_A \text{ and } \pi \in \Pi_B \\ \Theta_A(i, \pi) & \text{if } i = j \text{ and } (i, \pi) \in \Pi_A \text{ and } \pi \notin \Pi_B \\ \Theta_B(\pi) & \text{if } i = j \text{ and } (i, \pi) \notin \Pi_A \text{ and } \pi \in \Pi_B \end{cases}$

- $\approx_C = \approx_A \cup \{((j,\pi_1),(j,\pi_2)) \mid \pi_1 \approx_B \pi_2\}$

*The unification fails if there exists some pair* $(i,\pi) \in \Pi_{C'}$ *such that* $\Theta_{C'}(i,\pi) = \top$.

Many of the properties of AFSs, proven in the previous section, hold for AMRSs, too. In particular, if $A$ is an AMRSs then so is $(A,j) \sqcup B$ if it is defined, $len((A,j) \sqcup B) = len(A)$ and $(A,j) \sqcup B \sqsupseteq A$.

# 4 Parsing

Parsing is the process of determining whether a given string belongs to the language defined by a given grammar, and assigning a structure to the permissible strings. We formalize and explicate some of the notions of [3, chapter 13]. We give direct definitions for rules, grammars and languages, based on our new notion of AMRSs. This presentation is more adequate to current TFS-based systems than [7, 12], that use first-order terms. Moreover, it does not necessitate special, ad-hoc features and types for encoding trees in TFSs as [11] does. We don't assume any explicit context-free back-bone for the grammars, as does [13].

We describe a pure bottom-up chart-based algorithm. The formalism we presented is aimed at being a platform for specifying grammars in HPSG, which is characterized by employing a few very general rules; selecting the rules that are applicable in every step of the process requires unification anyhow. Therefore we choose a particular parsing algorithm that does not make use of top down predictions but rather assumes that every rule might be applied in every step. This assumption is realized by initializing the chart with predictive edges for every rule, in every position.

## 4.1 Rules and Grammars

We define rules and grammars over a fixed set WORDS of words. However, we assume that the lexicon associates with every word $w$ a feature structure $C(w)$, its **category**,[1] so we can ignore the terminal words and consider only their categories. The input for the parser, therefore, is a sequence[2] of TFSs rather than a string of words.

**Definition 4.1 (Pre-terminals)** *Let* $w = w_1 \ldots w_n \in \text{WORDS}^*$ *and* $A_i = C(w_i)$ *for* $1 \leq i \leq n$. $PT_w(j,k)$ *is defined if* $1 \leq j \leq k \leq n$, *in which case it is the AMRS* $Abs(\langle A_j, A_{j+1}, \ldots, A_k \rangle)$. *Note that* $PT_w(j,k) \cdot PT_w(k+1,m) = PT_w(j,m)$. *We omit the subscript* $w$ *when it is clear from the context.*

**Definition 4.2 (Rules)** *A* ***rule*** *is a MRS of length* $n > 1$ *with a distinguished last element. If* $\langle A_1, \ldots, A_{n-1}, A_n \rangle$ *is a rule then* $A_n$ *is its* ***head***[3] *and* $\langle A_1, \ldots, A_{n-1} \rangle$ *is its* ***body***.[4] *We write such a rule as* $\langle A_1, \ldots, A_{n-1} \Rightarrow A_n \rangle$. *In addition, every category of a lexical item is a rule (with an empty body). We assume that such categories don't head any other rule.*

**Definition 4.3 (Grammars)** *A* ***grammar*** $G = (\mathcal{R}, A_s)$ *is a finite set of rules* $\mathcal{R}$ *and a* ***start symbol*** $A_s$ *that is a TFS.*

An example grammar, whose purpose is purely illustrative, is depicted in figure 2. For the following discussion we fix a particular grammar $G = (\mathcal{R}, A_s)$.

**Definition 4.4 (Derivation)** *An AMRS* $A = \langle Ind_A, \Pi_A, \Theta_A, \approx_A \rangle$ ***derives*** *an AMRS* $B$ *(denoted* $A \rightarrow B$*) if there exists a rule* $\rho \in \mathcal{R}$ *with* $len(\rho) = n$ *and* $R = Abs(\rho)$, *such that*

- *some element of* $A$ *unifies with the head of* $R$: *there exist AMRSs* $A', R'$ *and* $j \in Ind_A$ *such that* $A' = (A,j) \sqcup R^n$ *and* $R' = (R,n) \sqcup A^j$

[1] Ambiguous words are associated with more than one category. We ignore such cases in the sequel.
[2] We assume that there is no reentrancy among lexical items.
[3] This use of *head* must not be confused with the linguistic one, the core features of a phrase.
[4] Notice that the traditional direction is reversed and that the head and the body need not be disjoint.

- *B can be obtained by replacing the j-th element of A′ with the body of R′.*[5]

*'$\xrightarrow{*}$' is the reflexive transitive closure of '$\rightarrow$'.*

Intuitively, $A$ derives $B$ through some AFS $A^j$ in $A$, if some rule $\rho \in \mathcal{R}$ licenses the derivation. $A^j$ is unified with the head of the rule, and if the unification succeeds, the (possibly modified) body of the rule replaces $A^j$ in $A$.

**Definition 4.5 (Language)** *The **language** of a grammar $G$ is $L(G) = \{w = w_1 \cdots w_n \in \text{WORDS}^* \mid A \xrightarrow{*} B$ for some $A \sqsupseteq Abs(A_s)$ and $B \sqsupseteq PT_w(1,n)\}$.*

Figure 3 shows a sequence of derivations, starting from some feature structure that is more specific than the initial symbol and ending in a sequence of structures that can stand for the string "John loves fish", based upon the example grammar.

## 4.2 Parsing as Operational Semantics

We view parsing as a computational process endowing TFS formalisms with an operational semantics, which can be used to derive control mechanisms for an abstract machine we design ([14]). A computation is triggered by some input string of words $w = w_1 \cdots w_n$ of length $n > 0$. For the following discussion we fix a particular input string $w$ of length $n$. A *state* of the computation is a set of *items*, and states are related by a transition relation. The presentation below corresponds to a pure bottom-up algorithm.

**Definition 4.6 (Dotted rules)** *A **dotted rule** (or **edge**) is a pair $\langle A, k\rangle$ where $A = Abs(\sigma)$ is an AMRS such that $\rho \sqsubseteq \sigma$ for some $\rho \in \mathcal{R}$ and where $0 \leq k < len(A)$. An edge $\langle A, k\rangle$ is **complete** if $k = len(A) - 1$; an edge is **active** otherwise.*

A dotted rule consists of an AMRS $A$ that is more specific than (the abstraction of) some grammar rule, and a number $k$ that denotes the position of the **dot** within $A$. The dot can precede any element of $A$ (in the case of lexical rules, it can also succeed the rule).

**Definition 4.7 (Items)** *An item is a triple $[i, \langle A, k\rangle, j]$ where $0 \leq i \leq j \leq n$ and $\langle A, k\rangle$ is a dotted rule. An item is complete if the edge in it is complete. Let* ITEMS *be the collection of all items.*

During parsing, the intuitive meaning of an item is that the part of $A$ prior to the dot (which is indicated by $k$) derives a substring of the input, and if it can be shown that the part of $A$ succeeding the dot derives a consecutive substring of the input, then the head of $A$ derives the concatenation of the two substrings. This invariant is formally defined and proven in the section 4.3. $i$ and $j$ indicate the *span* of the item.

A computation is determined by a sequence of states, each of which is a collection of items, where the first state corresponds to the initialization and each subsequent state contains its predecessor and is related to it by the transition relation.

**Definition 4.8 (States)** *A **state** $S \subseteq$ ITEMS is a finite set of items.*

**Definition 4.9 (Initialization)** *Let $\hat{S} = I_{lex} \cup I_{predict}$ be the **initial state**, where:*

$$I_{lex} = \{[i-1, \langle A_i, 0\rangle, i] \mid 1 \leq i \leq n \text{ and } A_i = PT_w(i,i)\}$$

$$I_{predict} = \{[i, \langle Abs(\rho), 0\rangle, i] \mid 0 \leq i \leq n \text{ and } \rho \in \mathcal{R}\}$$

$I_{lex}$ contains the (complete) items that correspond to categories of the input words, whereas $I_{predict}$ contains an (active) item for each grammar rule and a position in the input string.

[5] The exact details can be found in [15].

**Definition 4.10 (Dot movement)** *The partial function DM* : ITEMS × ITEMS → ITEMS *is defined as follows:* $DM([i,\langle A,k_A\rangle,l_A],[l_B,\langle B,k_B\rangle,j]) = [i,\langle C,k_C\rangle,j]$, *where:*

- $l_A = l_B, n = len(A), m = len(B)$
- $k_A < n-1$ *(the edge* $\langle A,k_A\rangle$ *is active),* $k_B = m-1$ *(the edge* $\langle B,k_B\rangle$ *is complete),* $k_C = k_A + 1$
- $C = (A,k+1) \sqcup B^m$ *(C is obtained from A by unifying the element of A succeeding the dot with the head of B)*

*DM* is not defined if $l_A \neq l_B$, if the edge in its first argument is complete, if the edge in its second argument is active, or if the unification fails.

**Lemma 4.11** *If* $x = [i_x,\langle A_x,k_x\rangle,j_x] \in$ ITEMS, $y = [i_y,\langle A_y,k_y\rangle,j_y] \in$ ITEMS *and* $z = DM(x,y)$ *is defined, then* $z = [i_z,\langle A_z,k_z\rangle,j_z]$ *where* $i_z = i_x, j_z \geq i_z, A_z \succeq A_x$ *and* $k_z > k_x$.

**Corollary 4.12** *If* $x,y \in$ ITEMS *then* $DM(x,y) \in$ ITEMS *if it is defined.*

To compute the next state, new items are added if they result by applying *DM* to existing items, unless the result is more specific than existing items. This is a realization of the *subsumption check* suggested in [11, 12].

**Definition 4.13 (Ordering items)** *If* $x = [i_x,\langle A_x,k_x\rangle,j_x]$ *and* $y = [i_y,\langle A_y,k_y\rangle,j_y]$ *are items, x subsumes y (written* $x \preceq y$*) iff* $i_x = i_y, j_x = j_y, k_x = k_y$ *and* $A_x \preceq A_y$.

**Definition 4.14** *Let* $\Delta(S) = \{z \mid z = DM(x,y)$ *for some* $x,y \in S$ *and there does not exist* $z' \in S$ *such that* $z' \preceq z\}$. *The transition relation '*$\vdash$*' holds between two states S and S' (denoted by* $S \vdash S'$*) if* $S' = S \cup \Delta(S)$.

**Definition 4.15 (Computation)** *A* ***computation*** *is an infinite sequence of states* $S_i, i \geq 0$, *such that* $S_0 = \hat{S}$ *and for every* $i \geq 0$, $S_i \vdash S_{i+1}$. *A computation is* ***terminating*** *if there exists some* $m \geq 0$ *for which* $S_m = S_{m+1}$ *(i.e., a fix-point is reached). A computation is* ***successful*** *if one of its states contains an item of the form* $[0,\langle A,k-1\rangle,n]$ *where n is the input length,* $k = len(A)$ *and* $Abs(A_s) \sqsubseteq A^k$; *otherwise, the computation fails.*

## 4.3 Proof of Correctness

In this section we show that parsing, as defined above, is (partially) correct. First, the algorithm is *sound*: computations succeed only for input strings that are sentences of the language. Second, it is *complete*: if a string is a sentence, it is accepted by the algorithm. Then we show that the computation terminates for *off-line parsable* grammars.

### 4.3.1 Soundness

**Lemma 4.16** *If* $[i,\langle A,k\rangle,j] \in S_m$ *for some* $m \geq 0$ *then* $i = j$ *only if* $\langle A,k\rangle$ *is not complete and* $k = 0$.

**Proof:** By induction on $m$.

**Theorem 4.17 (Completion)** *If* $\langle A,k\rangle$ *is a complete edge,* $len(A) > 1$ *and* $A^{1..k} \xrightarrow{*} B$ *then* $A^{k+1} \xrightarrow{*} B$.

**Proof:** Since $\langle A,k\rangle$ is an edge such that $len(A) > 1$, there exists an abstract rule $R$ such that $R \sqsubseteq A$. Hence $(A,k+1) \sqcup R^{k+1} = A^{k+1}$, $(R,k+1) \sqcup A^{k+1} = A$ and $A^{k+1} \rightarrow A^{1..k}$. Since $A^{1..k} \xrightarrow{*} B$ we obtain $A^{k+1} \xrightarrow{*} B$.

**Theorem 4.18 (Parsing invariant (a))** *If* $z = [i_z,\langle A_z,k_z\rangle,j_z] \in S_m$ *for some* $m \geq 0$, $l = len(A_z)$ *and* $i_z < j_z$, *then* $A_z^{1..k_z} \xrightarrow{*} PT(i_z+1,j_z)$ *if* $l > 1$, $A_z^1 \xrightarrow{*} PT(i_z+1,j_z)$ *if* $l = 1$.

**Proof:** By induction on $m$. Base: for all items $z \in I_{predict}$, $i_z = j_z$ and the proposition obtains (vacuously). For all items $z \in I_{lex}$, $l = 1$ and $A_z = PT(i_z + 1, j_z)$.
If $m > 0$: Let $x = DM(x, y)$ where $x = [i_x, \langle A_x, k_x \rangle, j_x]$, $y = [i_y, \langle A_y, k_y \rangle, j_y]$ and $x, y \in$ ITEMS. $\langle A_y, k_y \rangle$ is complete, hence by 4.16 $i_y < j_y$ and by the induction hypothesis and the completion theorem, $A_y^{k_y} \xrightarrow{*} PT(i_y + 1, j_y)$. Also, $len(A_z) > 1$ since all rules are of length $> 1$. By the induction hypothesis, $A_x^{1..k_x} \xrightarrow{*} PT(i_x + 1, j_x)$. Therefore $A_z^{1..k_x} \xrightarrow{*} PT(i_x + 1, j_x)$. Since $A_y^{k_y} \xrightarrow{*} PT(i_y + 1, j_y)$, we get $A_z^{k_x+1} \xrightarrow{*} PT(i_y + 1, j_y)$. Therefore $A_z^{1..k_z} \xrightarrow{*} PT(i_z + 1, j_z)$.

**Corollary 4.19** *If a computation triggered by $w = w_1, \ldots, w_n$ is successful then $w \in L(G)$.*

**Proof:** For a computation to be successful there must be a state $S_m$ that contains some item $[0, \langle A, k-1 \rangle, n]$ where $k = len(A)$ and $Abs(A_s) \sqsubseteq A^k$. From the above theorem it follows that $A^{1..k-1} \xrightarrow{*} PT(1, n)$. Since $A$ is complete, by the completion theorem $A^k \xrightarrow{*} PT(1, n)$, and thus $w_1, \ldots, w_n \in L(G)$.

### 4.3.2 Completeness

The following lemma shows that one derivation step, licensed by a rule $R$ of length $r + 1$, corresponds to $r$ applications of the $DM$ function, starting with an item that predicts $R$ and advancing the dot $r$ times, until a complete item is generated.

**Lemma 4.20** *If $A \rightarrow B$ and there exists $m \geq 0$ such that for every $1 \leq b \leq len(B)$ there exists $x_b \in S_m$ such that $x_b = [i_b, \langle B_b, k_b \rangle, j_b]$ is complete and $B_b^{k_b+1} = B^b$, and $i_1 = 0, j_{b-1} = i_b$ for $0 < b < len(B)$ and $j_{len(B)} = n$, then there exists $m' \geq 0$ such that for every $1 \leq a \leq len(A)$ there exists $y_a \in S_{m'}$ such that $y_a = [i_a, \langle A_a, k_a \rangle, j_a]$ is complete and $A_a^{k_A+1} = A^a$ and $i_1 = 0, j_{a-1} = i_a$ for $0 < a < len(B)$ and $j_{len(B)} = n$.*

**Proof:** (sketch) $A \rightarrow B$ by some rule $R$ of length $r + 1$ that expands the $p$-th element of $A$ to the elements $q_1, \ldots, q_r$ in $B$. For all elements of $A$ except $p$, the proposition holds by assumption. Since $R$ is a rule, there exists an item $x_R \in I_{predict}$ that $x_R = [i_{q_1}, \langle R, 0 \rangle, i_{q_1}]$ where $i_{q_1}$ is the first index of $x_{q_1}$. Let $y_1 = DM(x_R, x_{q_1})$ and $y_l = DM(y_{l-1}, x_{q_l})$ for $1 < l \leq r$. All the $y$ items exist: by the requirements on the indices of the $x_b$-s, the indices of the $y$ items fit. The unifications performed by $DM$ don't fail: if they would, $A$ wouldn't derive $B$. Then $y_r \in S_{m+r}$ is complete (as there were exactly $r$ applications of $DM$) and $y_r^{r+1} = A^p$.

**Theorem 4.21 (Parsing invariant (b))** *If $A \xrightarrow{*} PT(i+1, j)$ for $i < j$ then there exist $m \geq 0$ and $x \in S_m$ such that $x = [i, \langle B, k-1 \rangle, j]$ where $k = len(B)$ and $B^k = A$.*

**Proof:** By induction on $d$, the number of derivation steps. Base: if $d = 0$ then $A \rightarrow PT(i+1, j)$ iff $PT(i+1, j) \in I_{lex}$, in which case an item as required exists. If $d > 1$, an immediate application of the above lemma to the induction hypothesis gives the required result.

**Corollary 4.22** *If $w = w_1, \ldots, w_n \in L(G)$ then the computation triggered by $w$ is successful.*

**Proof:** Since $w_1, \ldots, w_n \in L(G)$, there exists an AFS $A \sqsupseteq Abs(A_s)$ such that $A \xrightarrow{*} PT(1, n)$. By the parsing invariant, there exist $m \geq 0$ and $x \in S_m, x = [0, \langle B, k-1 \rangle, n]$ where $k = len(B)$ and $A = B^k$. $Abs(A_s) \sqsubseteq A = B^k$ and therefore the computation is successful.

### 4.3.3 Termination

It is well-known (see, e.g.,[9, 11]) that unification-based grammar formalisms are Turing-equivalent, and therefore decidability cannot be guaranteed in the general case. This is also true for the formalism we describe here. However, for grammars that satisfy a certain restriction, termination of the computation can be proven. The following definition is an adaptation of the one given in [11].

**Definition 4.23 (Off-line parsability)** *A grammar $G$ is **off-line parsable** if there exists a function $f$ from AMRSs to AMRSs such that:*

- *for every AMRS $A$, $f(A) \sqsubseteq A$*
- *the range of $f$ is finite*
- *for every string $w$ and AMRSs $A, B$, if there exist $k_A, k_B$, that $A^{1..k_A} \xrightarrow{*} PT_w(i,j)$ and $B^{1..k_B} \xrightarrow{*} PT_w(i,j)$ and $A \not\sqsubseteq B$ and $B \not\sqsubseteq A$ then $f(A) \neq f(B)$.*

**Theorem 4.24** *If $G$ is off-line parsable then every computation terminates.*

**Proof:** (sketch) Select some computation triggered by some input $w$ of length $n$. We want to show that only a finite number of items can be generated during the computation. Observe that the indices that determine the span of the item are limited: $0 \leq i \leq j \leq n$. The location of the dot within each AMRS $A$ is also limited: $0 \leq k < len(A)$. It remains to show that only a finite number of edges are generated. Suppose that $[i, \langle A, k \rangle, j] \in S$ is an item that was generated during the computation. Now suppose another item is generated where only the AMRS is different: $[i, \langle B, k \rangle, j]$. If $B \sqsupseteq A$ it will not be included in $\Delta(S)$ because of the subsumption test. There is only a finite number of AMRSs $A'$ such that $B \sqsubseteq A$ (since subsumption is a well-founded relation). Now suppose $A \not\sqsubseteq B$ and $B \not\sqsubseteq A$. By the parsing invariant (a) there exist $A', B'$ such that $A^{1..k} \xrightarrow{*} PT_w(i,j)$ and $B^{1..k} \xrightarrow{*} PT_w(i,j)$. Since $G$ is off-line parsable, $f(A) \neq f(B)$. Since the range of $f$ is finite, there are only finitely many edges with equal span that are pairwise incomparable.

Since only a finite number of items can be generated, and the states of the computation are such that $S_m \subseteq S_{m+1}$ for $m \geq 0$, a fix-point is reached within a finite number of state transitions.

# References


[1] Bob Carpenter. Typed feature structures: A generalization of first-order terms. In Vijai Saraswat and Ueda Kazunori, editors, *Logic Programming – Proceedings of the 1991 International Symposium*, pages 187–201, Cambridge, MA, 1991. MIT Press.

[2] Bob Carpenter. ALE – the attribute logic engine: User's guide. Technical report, Laboratory for Computational Linguistics, Philosophy Department, Carnegie Mellon University, Pittsburgh, PA 15213, December 1992.

[3] Bob Carpenter. *The Logic of Typed Feature Structures.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

[4] E. Allen Emerson. Alternative semantics for temporal logics. *Theoretical Computer Science*, 26:121–130, 1983.

[5] Gregor Erbach. ProFIT: Prolog with features, inheritance and templates. CLAUS Report 42, Computerlinguistik, Universität des Saarlandes, D-66041, Saarbrücken, Germany, July 1994.

[6] Dale Gerdemann. Troll: Type resolution system – fundamental principles and user's guide. Unpublished manuscript, September 1993.

[7] Andrew Haas. A parsing algorithm for unification grammar. *Computational Linguistics*, 15(4):219–232, December 1989.

[8] Nicholas Haddock, Ewan Klein, and Glyn Morill, editors. *Categorial Grammar, Unification and Parsing*, volume 1 of *Working Papers in Cognitive Science.* University of Edingburgh, Center for Cognitive Science, 1987.

[9] Mark Johnson. *Attribute-Value Logic and the Theory of Grammar*, volume 16 of *CSLI Lecture Notes*. CSLI, Stanford, California, 1988.

[10] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, 1994.

[11] Stuart M. Shieber. *Constraint-Based Grammar Formalisms*. MIT Press, Cambridge, Mass., 1992.

[12] Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. Technical Report TR-11-94, Center for Research in Computing Technology, Division of Applied Sciences, Harvard University, April 1994.

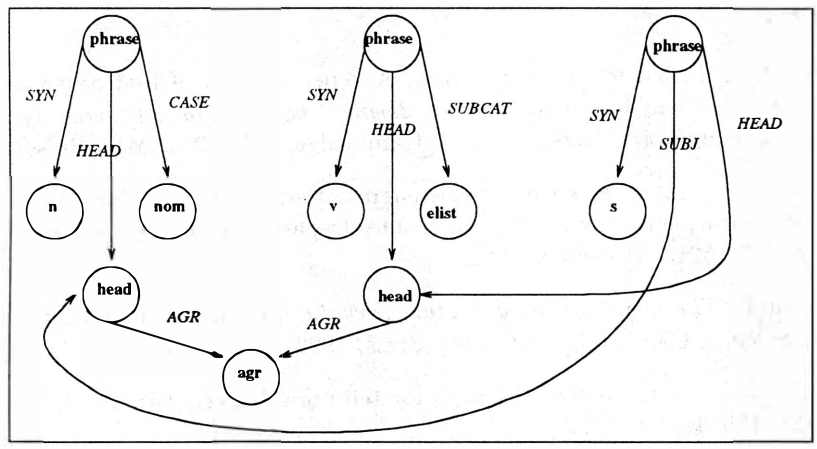
[13] Klaas Sikkel. *Parsing Schemata*. Klaas Sikkel, Enschede, 1993.

[14] Shuly Wintner and Nissim Francez. An abstract machine for typed feature structures. In *Proceedings of the 5th Workshop on Natural Language Understanding and Logic Programming*, Lisbon, May 1995.

[15] Shuly Wintner and Nissim Francez. Parsing with typed feature structures. Technical report, Laboratory for Computational Linguistics, Computer Science Department, Technion, 32000 Haifa, Israel, Forthcoming.

[16] Rémi Zajac. Inheritance and constraint-based grammar formalisms. *Computational Linguistics*, 18(2):159–182, 1992.

# A Examples




$$\begin{bmatrix} \textbf{phrase} \\ SYN: & [\textbf{n}] \\ HEAD: & \boxed{1}\begin{bmatrix}\textbf{head} \\ AGR: & \boxed{3}[\textbf{agr}]\end{bmatrix} \\ CASE: & [\textbf{nom}] \end{bmatrix} \quad \begin{bmatrix} \textbf{phrase} \\ SYN: & [\textbf{v}] \\ HEAD: & \boxed{2}\begin{bmatrix}\textbf{head} \\ AGR: & \boxed{3}\end{bmatrix} \\ SUBCAT: & [\textbf{elist}] \end{bmatrix} \quad \begin{bmatrix} \textbf{phrase} \\ SYN: & [\textbf{s}] \\ SUBJ: & \boxed{1} \\ HEAD: & \boxed{2} \end{bmatrix}$$

Figure 1: A graph- and AVM- representation of a MRS

The grammar listed in figure 2 consists of four rules and three lexical entries. The rules are extensions of the common context-free rules S → NP VP, NP → PN and VP → V NP. Notice that the head of each rule is on the right hand side of the '⇒' sign. Note also that values are shared among the body and the head of each rule, thus enabling percolation of information during derivation.

Initial symbol:

$$\begin{bmatrix} \textbf{phrase} \\ SYN: & [\textbf{s}] \end{bmatrix}$$

Rules:

$$\begin{bmatrix} \textbf{phrase} \\ SYN: & [\textbf{n}] \\ HEAD: & \boxed{1}\begin{bmatrix}\textbf{head} \\ AGR: & \boxed{3}\end{bmatrix} \\ CASE: & [\textbf{nom}] \end{bmatrix} \begin{bmatrix} \textbf{phrase} \\ SYN: & [\textbf{v}] \\ HEAD: & \boxed{2}\begin{bmatrix}\textbf{head} \\ AGR: & \boxed{3}\end{bmatrix} \\ SBCT: & [\textbf{elist}] \end{bmatrix} \Rightarrow \begin{bmatrix} \textbf{phrase} \\ SYN: & [\textbf{s}] \\ SUBJ: & \boxed{1} \\ HEAD: & \boxed{2} \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} \textbf{word} \\ SYN: & [\textbf{pn}] \\ HEAD: & \boxed{1} \\ CASE: & \boxed{2} \end{bmatrix} \Rightarrow \begin{bmatrix} \textbf{phrase} \\ SYN: & [\textbf{n}] \\ HEAD: & \boxed{1} \\ CASE: & \boxed{2} \end{bmatrix} \quad (2)$$

$$\begin{bmatrix} \textbf{word} \\ SYN: & [\textbf{v}] \\ HEAD: & \boxed{1} \\ SBCT: & \begin{bmatrix}\textbf{nelist} \\ 1ST: & \boxed{3} \\ RST: & \boxed{2}\end{bmatrix} \end{bmatrix} \boxed{3} \begin{bmatrix} \textbf{phrase} \\ SYN: & [\textbf{n}] \\ HEAD: & [\textbf{head}] \\ CASE: & [\textbf{acc}] \end{bmatrix} \Rightarrow \begin{bmatrix} \textbf{phrase} \\ SYN: & [\textbf{v}] \\ HEAD: & \boxed{1} \\ SBCT: & \boxed{2} \end{bmatrix} \quad (3)$$

Lexicon:

$$\text{"}John\text{"} \mapsto \begin{bmatrix} \textbf{word} \\ SYN: & [\textbf{pn}] \\ HEAD: & \begin{bmatrix}\textbf{head} \\ AGR: & \begin{bmatrix}\textbf{agr} \\ PERS: & [\textbf{3rd}] \\ NUM: & [\textbf{sg}]\end{bmatrix}\end{bmatrix} \\ CASE: & [\textbf{case}] \end{bmatrix} \quad (4)$$

$$\text{"}loves\text{"} \mapsto \begin{bmatrix} \textbf{word} \\ SYN: & [\textbf{v}] \\ HEAD: & \begin{bmatrix}\textbf{head} \\ AGR: & [\textbf{agr} \\ & NUM: [\textbf{sg}]]\end{bmatrix} \\ SBCT: & \begin{bmatrix}\textbf{nelist} \\ 1ST: & [\textbf{phrase} \\ & SYN: [\textbf{n}]] \\ RST: & [\textbf{elist}]\end{bmatrix} \end{bmatrix} \quad (5)$$

$$\text{"}fish\text{"} \mapsto \begin{bmatrix} \textbf{word} \\ SYN: & [\textbf{pn}] \\ HEAD: & \begin{bmatrix}\textbf{head} \\ AGR: & [\textbf{agr}]\end{bmatrix} \\ CASE: & [\textbf{case}] \end{bmatrix} \quad (6)$$

Figure 2: An example grammar

A leftmost derivation of the string "John loves fish" is given in figure 3, where each derivation

bears the number of the rule that licenses it. The string is a sentence of the grammar since the derivation starts with a feature structure that is more specific than the initial symbol and ends with feature structures that are subsumed by the lexical entries of the input string.

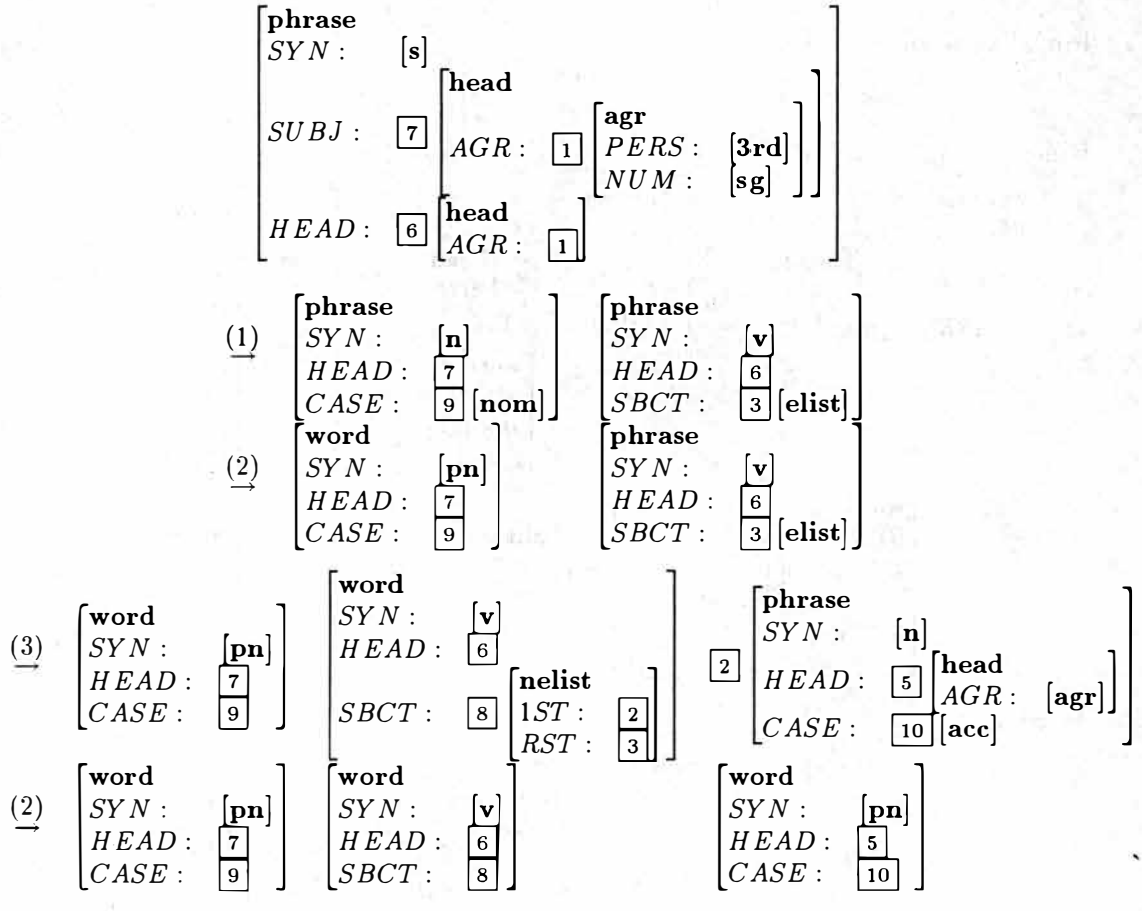
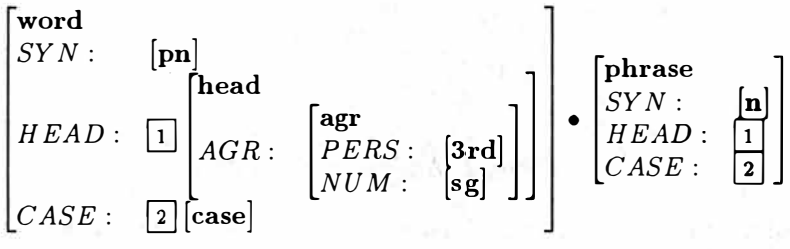



Figure 3: A leftmost derivation

Finally, we simulate the process of parsing with the example grammar and the input "John loves fish". As a matter of convention, if $[i, \langle A, k\rangle, j]$ is an item, we say that the edge $\langle A, k\rangle$ is in the $(i, j)$ entry. We also explicitly indicate the position of the dot (denoted by '•') within MRSs instead of using an integer.

The first state, $S_0$, consists of $I_{lex} \cup I_{predict}$; $I_{lex}$ contains three items: the pre-terminals corresponding to "John", "loves" and "fish" with the dot set to 0 in entries $(0,1), (1,2)$ and $(2,3)$, respectively. $I_{predict}$ contains, for each entry $(i, i)$, where $0 \leq i \leq 3$, an edge of the form $\langle R, 0\rangle$, where $R$ is one of the grammar rules. Thus there are 12 items in $I_{predict}$.

$S_1$ contains three more items. Application of $DM$ to the item corresponding to rule 2 in entry $(0, 0)$ and to the item corresponding to "John" in $(0, 1)$ results in the addition of the edge




(7)

to entry $(0,1)$. In a similar way, the edge

$$\left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{v}] \\ HEAD: & \boxed{1}\left[\begin{array}{ll}\textbf{head} & \\ AGR: & \left[\begin{array}{ll}\textbf{agr} & \\ NUM: & [\textbf{sg}]\end{array}\right]\end{array}\right] \\ SBCT: & \left[\begin{array}{ll}\textbf{nelist} & \\ 1ST: & \boxed{3} \\ RST: & \boxed{2}[\textbf{elist}]\end{array}\right]\end{array}\right] \bullet \boxed{3}\left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{n}] \\ HEAD: & [\textbf{head}] \\ CASE: & [\textbf{acc}]\end{array}\right]\left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{v}] \\ HEAD: & \boxed{1} \\ SBCT: & \boxed{2}\end{array}\right] \tag{8}$$

is added to entry $(1,2)$, by virtue of rule 3 and "loves" and the edge

$$\left[\begin{array}{ll}\textbf{word} & \\ SYN: & [\textbf{pn}] \\ HEAD: & \boxed{1}\left[\begin{array}{ll}\textbf{head} & \\ AGR: & [\textbf{agr}]\end{array}\right] \\ CASE: & \boxed{2}[\textbf{case}]\end{array}\right] \bullet \left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{n}] \\ HEAD: & \boxed{1} \\ CASE: & \boxed{2}\end{array}\right] \tag{9}$$

is added to entry $(2,3)$ by virtue of the rule 2 and "fish".

Several items are added to $S_2$, two of which are of more interest. On the basis of item 7 and the item corresponding to rule 1 in $(0,0)$, the edge

$$\left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{n}] \\ HEAD: & \boxed{1}\left[\begin{array}{ll}\textbf{head} & \\ AGR: & \boxed{3}\left[\begin{array}{ll}\textbf{agr} & \\ PERS: & [\textbf{3rd}] \\ NUM: & [\textbf{sg}]\end{array}\right]\end{array}\right] \\ CASE: & [\textbf{case}]\end{array}\right] \bullet \left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{v}] \\ HEAD: & \boxed{2}\left[\begin{array}{ll}\textbf{head} & \\ AGR: & \boxed{3}\end{array}\right] \\ SBCT: & [\textbf{elist}]\end{array}\right]\left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{s}] \\ SUBJ: & \boxed{1} \\ HEAD: & \boxed{2}\end{array}\right] \tag{10}$$

is added to $(0,1)$. On the basis of item 8 in $(1,2)$ and item 9 in $(2,3)$, the following edge is added to $(1,3)$:

$$\left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{v}] \\ HEAD: & \boxed{1}\left[\begin{array}{ll}\textbf{head} & \\ AGR: & \left[\begin{array}{ll}\textbf{agr} & \\ NUM: & [\textbf{sg}]\end{array}\right]\end{array}\right] \\ SBCT: & \left[\begin{array}{ll}\textbf{nelist} & \\ 1ST: & \boxed{3} \\ RST: & \boxed{2}[\textbf{elist}]\end{array}\right]\end{array}\right] \boxed{3}\left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{n}] \\ HEAD: & \left[\begin{array}{ll}\textbf{head} & \\ AGR: & [\textbf{agr}]\end{array}\right] \\ CASE: & [\textbf{acc}]\end{array}\right] \bullet \left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{v}] \\ HEAD: & \boxed{1} \\ SBCT: & \boxed{2}\end{array}\right] \tag{11}$$

This complete edge can now be used with edge 10 to form, in $S_3$, the following edge in $(0,3)$:

$$\left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{n}] \\ HEAD: & \boxed{1}\left[\begin{array}{ll}\textbf{head} & \\ AGR: & \boxed{3}\left[\begin{array}{ll}\textbf{agr} & \\ PERS: & [\textbf{3rd}] \\ NUM: & [\textbf{sg}]\end{array}\right]\end{array}\right] \\ CASE: & [\textbf{case}]\end{array}\right] \left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{v}] \\ HEAD: & \boxed{2}\left[\begin{array}{ll}\textbf{head} & \\ AGR: & \boxed{3}\end{array}\right] \\ SBCT: & [\textbf{elist}]\end{array}\right] \bullet \left[\begin{array}{ll}\textbf{phrase} & \\ SYN: & [\textbf{s}] \\ SUBJ: & \boxed{1} \\ HEAD: & \boxed{2}\end{array}\right] \tag{12}$$

and since the head of this complete edge, which spans the entire input string, is more specific than the initial symbol, the string "John loves fish" is accepted by the parser.