Preface

This report was written as a Master's Thesis in Computer Science with the Universiteit Leiden. My supervisors were Dr. Crit Cremers from the Department of General Linguistics, and Dr. Hendrik-Jan Hoogeboom from the Leiden Institute of Advanced Computer Science. These names at once sketch the interdisciplinary nature of this project.

Although Master's Thesis projects are formally confined to one semester, most students take well above that time to complete their projects; I myself are no exception. In a sense, preparations for this project began way back in my first year of studying in Leiden, when a fellow member of my students' association introduced me to the lectures provided by the Linguistics Department. These lectures served to give my interest in languages a formal footing. No less importantly, I made my acquaintance with Dr. Cremers, an acquaintance which lasted during the years and eventually led to an opening for a final project.

Crit Cremers, together with his technical assistant Maarten Hijzelendoorn, is responsible for the development of the Delilah system, a parsing system for Dutch to which this report is devoted. This system achieves quite impressive results, giving sensible syntactic and semantic analyses for all kinds of long and complicated Dutch sentences. However, it was not quite known whether Delilah was a *polynomial time* algorithm — the only way to ensure reasonable performance in practice, especially when its knowledge of Dutch words approaches a realistic level and more aspects of semantic analysis are brought in. By the same token, an important question to be settled was whether the kind of grammar used by Delilah could be classified as adhering to some well-known and well studied grammar formalism. If so, it is likely that one can apply what is known about the efficiency of parsing a grammar made up in such a formalism.

Being brought up as a Computer Scientist and having caught up on Linguistics, it was a pleasure for me to try and shed some light on these questions. It was an even greater pleasure to find that I have obtained to some degree conclusive answers to both of them. The greatest pleasure was that now Delilah comes equipped with a polynomial time parser, while her grammar has proved to fit in with a currently active tradition of research in Formal Language Theory called the Mildly Context-Sensitive Grammars (cf. [JVW91]). This was about as much as I could wish (while rumour has it that my supervisors agree with me on this point).

Thanks are due to Crit and Maarten for your enthousiasm (still crazy about Delilah after all these years!), the many hours of discussion and the very pleasant cooperation. Thanks also to Hendrik-Jan Hoogeboom for declaring this to be real Computer Science and for biking to the Linguistics Department with me on more than one occasion. Thanks finally to the God in heaven above, the living Word and the great Father, who provided me with all knowledge, skills, and time of living to complete this exercise.

Leiden, July 20, 1999
Christiaan van de Woestijne
e-mail: cvdwoest@math.LeidenUniv.nl

Contents

Chapter 1

Introduction

Delilah is an automated system for "parsing categorial Dutch", which has been and is being developed at the Universiteit Leiden. Its authors are Crit Cremers and Maarten Hijzelendoorn (see [Cre93, Hijwww]).

Delilah currently takes the form of a PROLOG-based modular system that takes written Dutch sentences (or allegedly so), verifies their Dutchness, and in doing so produces a syntactic and semantic analysis of their content. The main inspiration for the syntactic components has been drawn from Categorial Grammars (see [Bar53, Lam58, BMvB88, Ste00, Cre93], to name just a few references). The semantic component may be described as an Attribute Grammar (see [DJL88] for more information).

The analyzing process comprises the following stages:

categorisation: During this stage, each word of the input sentence is assigned a category drawing upon the knowledge stored in Delilah's lexicon. This produces a set containing all lists of categories which may yield a syntactically valid analysis. Typically, this set is large, as most words are contained in many categories. This stage is designated as the 'count protocol', owing to the Van Benthem count being used as the main device for doing a first prune in the gigantic space of possible categorisations.

syntactic parsing: For each of the 'well-formed type lists' produced in the previous stage, Delilah tries to construct a parse tree. Such a tree contains all information obtained in the process of trying to reduce the initial list of categories via merging of categories to the single (atomic) categories of either 'sentence' or 'question'. It is possible that a type list does not give rise to a well-formed parse tree, which means rejection of the sentence as ungrammatical, or that several essentially different analyses are possible.

semantic analysis: Taking one parse tree, in this stage it is attempted to calculate several semantic attributes of each node in the tree, such as its Theta role, its Person, Tense, and Number, and a representation of its meaning in a formalism of Modal Predicate Logic. Failure may represent basic ingrammaticalities such as incongruence in number between verb and subject in a sentence, incompleteness of the lexicon in semantic respect, or a need for further sophistication of the semantic module.

In this work, we will provide a detailed description of the formalisms and algorithms used in the first two stages. In this process, we will restrict ourselves to non-coordinated sentences. In principle, Delilah is able to parse sentences which contain one coordinator (such as 'en' (*and*) or 'of' (*or*)); however, this gives rise to essentially new techniques which do not expand on the basic mechanisms exposed in the following chapters. Likewise, we will assume the semantic module as given.

We will show furthermore that the categorisation and parsing algorithms currently in use with Delilah are not optimal, by describing another parsing algorithm based on the CKY-algorithm for context-free grammars, that operates in polynomial time. We will report on the practical behaviour of this algorithm.

We will conclude the work by presenting our conclusions and providing several pointers to future research concerning the Delilah system.

Chapter 2

Investigating Delilah's Grammar

This chapter is devoted to the formal investigation of the grammar formalism used by Delilah. First, we will present the general format of the objects and rules used in this grammar. Secondly, we explore the relation between this grammar and two currently studied general grammar formalisms, to wit, CCGs and LIGs. These formalisms, which are weakly equivalent [JVW91], are interesting for two reasons: although not all languages recognized by CCGs and LIGs are context free, it has been shown that not all context sensitive languages have grammars of this type. Furthermore, these formalisms have been shown to be parsable in polynomial time [VW93].

After these formal investigations, we will address the issue of parsing in the subsequent chapters.

2.1 Standard notations and definitions

In a standard categorial framework, a grammar consists of

1. a set of terminal symbols T
2. a finite set of nonterminals V
3. a possibly infinite set $\mathcal{C}_V$ of *categories* over V, defined as follows:
   (a) Nonterminals are in $\mathcal{C}_V$.
   (b) If c_1 and c_2 are in $\mathcal{C}_V$, then so are $(c_1 \backslash c_2)$ and (c_1/c_2).
4. a lexical assignment f assigning to each $t \in T$ a finite subset of $\mathcal{C}_V$
5. a set R of reduction rules, which may be finite or given by a finite set of rule templates
6. a finite subset (in many cases, a singleton subset) of V which are distinguished categories.

Categories may be viewed as denoting types of functions, with an added directionality component: (c_1/c_2) denotes the type or category of functions taking an object of category c_2 to an object of category c_1. Thus c_2 is the (right) *argument* of the function, and the category is said to have a right searching argument c_2. Likewise, $(c_1 \backslash c_2)$ has a left searching argument, and the two types of functions are distinct.

It is possible to abstract away from directionality, thus obtaining a formal system that is occupied solely with function application and composition; however, when applying categorial grammars to natural languages it is clear that directionality cannot be dispensed with.

There are a number of useful concepts linked to categorial grammars, which we will define in turn.

Let $\mathbf{x}$ and $\mathbf{y}$ be categories (i.e., members of $\mathcal{C}_V$). If $\mathbf{x}$ is atomic (i.e., a member of V), then $\mathbf{x}$ is its own *head* (or *target*). Furthermore, the *head* of $\mathbf{x/y}$ and of $\mathbf{x \backslash y}$ is inductively defined to be the head of $\mathbf{x}$.

The *degree* of a list of categories $\mathbf{x_1 x_2 \dots x_m}$ is defined inductively as follows. degree($\mathbf{x}$) = 1 if $\mathbf{x}$ is in V; degree($\mathbf{x}$ $\mathbf{y}$) = degree($\mathbf{x/y}$) = degree($\mathbf{x\backslash y}$) = degree($\mathbf{x}$) + degree($\mathbf{y}$).

The *order* of a category is its recursion depth: order($\mathbf{x}$) = 0 if $\mathbf{x}$ is in T; order($\mathbf{x/y}$) = order($\mathbf{x\backslash y}$) = max(order($\mathbf{x}$), order($\mathbf{y}$)) + 1.

Finally, the *arity* of a category denotes the number of arguments the category can take when viewed as a function, and is defined as follows: arity($\mathbf{x}$) = 0 if $\mathbf{x}$ is in V; arity($\mathbf{x\backslash y}$) = arity($\mathbf{x/y}$) = arity($\mathbf{x}$) + 1.

One particular categorial grammar formalism is the Combinatory Categorial Grammars (CCGs), which originate from linguistic work by Steedman [Ste86, Ste85]. The definition of CCGs below is as given by [VW93, p. 610]. A CCG has:

- A finite set of terminals T;
- One distinguished basic category $\mathbf{s}$;
- A *finite* set of reduction rules R of the following form:
  1. A forward rule has the following form where $m \geq 0$.

  $$(x/y) \quad (y|_1 z_1|_2 \cdots |_m z_m) \rightarrow (x|_1 z_1|_2 \cdots |_m z_m)$$

  2. A backward rule has the following form where $m \geq 0$.

  $$(y|_1 z_1|_2 \cdots |_m z_m) \quad (x\backslash y) \rightarrow (x|_1 z_1|_2 \cdots |_m z_m)$$

The sources differ as to whether it is allowed to assign categories to the empty string (for example, in [JVW91] it is allowed); I am under the impression that one needs to resort to this in order to establish the equivalence of CCGs with LIGs and some other formalisms proved by Joshi *et al.* (cf. [JVW91]). To be specific, from their papers it is apparent that every CCG can be easily transformed to an LIG; the other way around, however, goes via TAGs, and it is here that assigning categories to the empty string is probably essential. In Delilah, however, no use is made of empty ("nonlexical") categories.

A grammar formalism which turns out to be closely related to CCGs is the Linear Indexed Grammars (see [JVW91, p. 56–57]). They were defined by Gazdar as a restriction of the Indexed Grammar formalism, introduced by Aho as a generalisation of CF grammars. Indexed grammars may be described as CF grammars in which to each nonterminal we associate a stack of so-called stack indices. Rule applications can be made subject to conditions on the top of the stack. Gazdar's restriction consists in letting only one daughter nonterminal inherit its parent's stack, instead of all. This gives the following definition for Linear Indexed Grammars (LIGs), as taken from [VW93, p. 592]: a LIG has:

- A finite set of nonterminals V_N;
- A finite set of terminals V_T;
- A finite set of indices (stack symbols) V_I;
- One distinguished nonterminal S;
- A finite set of productions P of the following form: let $A(\cdot\cdot\,\alpha)$ denote the nonterminal A associated with a stack consisting of an arbitrary (possibly empty) bottom part and a given top part α. $A(\,)$ will denote A associated with the empty stack. Then the general form of a production is:

  $$A(\cdot\cdot\,\alpha) \rightarrow w_1 A_1(\alpha_1) w_2 \dots A_{i-1}(\alpha_{i-1}) w_i A_i(\cdot\cdot\,\alpha_i) w_{i+1} A_{i+1}(\alpha_{i+1}) \dots A_n(\alpha_n) w_{n+1}$$

  for some $n \geq 0$, and with $w_1, \dots, w_{n+1}$ members of V_T^*.

Vijay-Shanker and Weir point out [VW93, p. 594] that it is easy to find an analogue of the Chomsky normal form for CF grammars to LIGs. This normal form is given, in a slightly adapted form, by

1. $A(\alpha) \to \varepsilon$ where $\varepsilon \in V_T \cup \{\epsilon\}$ (I drop the restriction that α is nonempty).
2. $A(\cdot\cdot\gamma_1 \ldots \gamma_m) \to A_p(\cdot\cdot\gamma_p)A_s(\alpha_s)$ where $m \geq 0$.
3. $A(\cdot\cdot\gamma_1 \ldots \gamma_m) \to A_s(\alpha_s)A_p(\cdot\cdot\gamma_p)$ where $m \geq 0$.
4. $A(\cdot\cdot\gamma_1 \ldots \gamma_m) \to A_p(\cdot\cdot\gamma_p)$ where $m \geq 0$.

From this normal form, the close resemblance of CCGs and LIGs stands out immediately. Obviously, p stands for *primary*, and s for *secondary*. This normal form is used by Vijay-Shanker and Weir to prove the (weak) equivalence of these two formalisms [VW94].

2.2 The Delilah grammar format

2.2.1 Alterations in the Delilah notations

As motivated in [Cre93], the Delilah grammar implements several restrictions and alterations on the categorial grammar formalism. First, the categories assigned to a word are required to be *linear*, that is to say, to have order 1. Thus they may be represented as

$$x|_1 z_1|_2 z_2 \ldots |_m z_m$$

where $|_i \in \{\backslash, /\}$ and $x, z_i \in V$. For a nontrivial linear grammar, it is necessary to utilize an extended nonterminal set. By comparison, Lambek [Lam58] initially used only two symbols: n and s.

Secondly, the form of the categories is adapted for both computational and linguistic purposes. The categories actually assigned by Delilah take the form $\mathbf{x\backslash L\backslash L_{sp}/R/R_{sp}}$, where $\mathbf{x}$ is an atom, and $\mathbf{L}$, $\mathbf{L_{sp}}$, $\mathbf{R}$ and $\mathbf{R_{sp}}$ are lists of arguments. As may be seen, the set of arguments has been split into left and right searching arguments, both of which are split further into *special* and *ordinary* arguments. This last difference will induce a different form of cancellation rule.

The relative order of the arguments is reversed with respect to the order displayed in the traditional category notation: the leftmost element of both lists represents the searcher which is to be satisfied first — we might say, the most peripheral argument. Thus a representation of the argument lists as Prolog lists becomes straightforward.

Thirdly, all heads and arguments of Delilah categories are flagged. These flags trigger certain restrictions on the application of the reduction rules given below. Possible conditions for rule applicability are on the head flags of both $\mathbf{x}$ and $\mathbf{y}$, and on some or all of the argument lists, which may be required to be empty. Furthermore, every rule instance thus defined also has to specify the head flags of the resulting category.

2.2.2 Delilah's reduction rules

As to the reduction rules used, Delilah uses one rule template, the so-called Generalized Directional Categorial Rule, which may be formulated as in [Cre93, p. 35], but incorporating the special arguments:

(a) $\mathbf{x\backslash L\backslash L_{sp}/[y\ R]/R_{sp}} \quad \mathbf{y\backslash L'\backslash L'_{sp}/R'/R'_{sp}} \to \mathbf{x\backslash[L'L]\backslash[L_{sp}L'_{sp}]/[R'R]/[R_{sp}R'_{sp}]}$

(b) $\mathbf{y\backslash L'\backslash L'_{sp}/R'/R'_{sp}} \quad \mathbf{x\backslash[y\ L]\backslash L_{sp}/R/R_{sp}} \to \mathbf{x\backslash[L'L]\backslash[L_{sp}L'_{sp}]/[R'R]/[R_{sp}R'_{sp}]}$

(c) $\mathbf{x\backslash L\backslash L_{sp}/[\,]/[y\ R_{sp}]} \quad \mathbf{y\backslash L'\backslash L'_{sp}/R'/R'_{sp}} \to \mathbf{x\backslash[LL']\backslash[L_{sp}L'_{sp}]/R'/[R_{sp}R'_{sp}]}$

(d) $\mathbf{y\backslash L'\backslash L'_{sp}/R'/R'_{sp}} \quad \mathbf{x\backslash[\,]\backslash[y\ L_{sp}]/R/R_{sp}} \to \mathbf{x\backslash L'\backslash[L_{sp}L'_{sp}]/[R'R]/[R_{sp}R'_{sp}]}$

Here $\mathbf{x}$ and $\mathbf{y}$ range over V, and the capital letters denote possibly empty lists of categories.

This rule format has been designed to satisfy three conditions. Firstly, for any rule, the number of occurrences of each basic category should be the same on both sides (counting arguments as negative, arguments of arguments as positive, and so on, recursively), and should thus be occurrence-sensitive (cf. [Cre93, p. 12–13], [Ben86]).

Secondly, each rule has to respect directionality. This is taken to mean that if a category c asks for some argument category c' on its left, the lexical element which introduces c''s head should occur to the left of the lexical constituent bearing the category c, and the same, m.m., holds for the right searching arguments.

It is important to note that this does not imply anything about the position of the lexical material corresponding to the *arguments* of c', as these may very well occur to the right of c. This phenomenon is called *disharmony*, and is used in the Delilah grammar in an essential way to accommodate crossing dependencies in Dutch.

Thirdly, all rules are required to have an antecedent of at most length 2 (to enforce adjacency — cf. [Cre93, p. 26–27]).

A pleasant property of these rules is that they are degree-decreasing. Roughly speaking, this means that the number of basic categories occurring on the right hand side is strictly less than that on the left hand side — and here all occurrences are counted as positive. This immediately implies that the language membership for this type of grammars is decidable, as the height of a derivation tree is bounded above by the length of the input string.

2.3 Actual implementation

In this section, we elaborate on the way the described grammar format is actually used in the Delilah system for parsing written Dutch sentences. We might first remark, somewhat obviously perhaps, that Delilah's set of terminals consists of words of Dutch.

2.3.1 Categories

In Delilah, all heads carry two flags (left and right), indicating whether the corresponding argument lists have undergone any changes with respect to their originally assigned values. Their values are taken from the set **{0, 1, wh}**, indicating no change, change, and incorporation of a Wh-movement, respectively.

As the head flags indicate what may be called the *virginity* of the corresponding argument list [Cre93, p. 45ss.], the head flags of the resulting category may be computed from those of the primary and secondary heads, and from the operations done on the argument lists. The argument list of the primary category containing **y** will have been changed by the merging with the corresponding argument list of the secondary category, and will be flagged **1** accordingly, unless **y** is marked **^wh**, in which case the flag becomes **wh**. The merge of the other two lists will be marked **1** unless one of them is empty and the other is still virginal, or both are empty. However, there are exceptions to this rule, which follow from the concrete rule instances below.

The argument flags for right searching ordinary arguments are taken from the set **{0, 1, 4, 50, 56, 59, 6, 8, 9}**. For left searching ordinary arguments, the possibilities are **{0, 4, 6, 7, 8, 88, 9}**. Of these flags, the left searching **4** and **9** seem not to be used anymore. The flags refer to differences in the *degree of transparency* that is allowed for the secondary constituent in a reduction. Being transparent means that part or all of the internal structure of the constituent is visible from 'outside' the constituent. As may be seen, allowing harmonious or even disharmonious composition has the effect of revealing the structure of the secondary constituent. For the exact meaning of the **0, 4, 6**, and **9** flags we refer to [Cre93, Chapter 1, Section 4] and to Chapter 4.

The remaining flags are used for specific linguistic constructions and subsume the meaning of one of the 'basic' flags, with additional constraints.

The flag **1** can only take closed constituents in which a Wh-movement has taken place. This is indicated by a **wh** head flag.

The flags **5x** pose the same requirements as the **x** flag with the additional one that the empty right argument list of the secondary category result from the satisfaction of one or more arguments. This situation arises in Infinitive-pro-participle (IPP) constructs like

ik heb hem zien werken

I have him see work
I have seen him working

Here the **5x** flag is assigned to the **vp** argument of the auxiliary for the perfect tense ('heb').

The flag **7** is used for accommodating the so-called 'green word-order'. In the opposition

ik denk dat henk heeft gewerkt
ik denk dat henk gewerkt heeft
I think that Henk has been working

the first sentence has red order, while the second is in green order.

The flag **8** is used mainly for adjunct types, and gives the maximal freedom for selecting secondary categories.

Flag **88** has been added for the case of resultant attributive adjuncts, like

...dat hij de auto groen verfde
...that he painted the car green

The use of the special argument lists is heavily restricted. On the left side, no special list is allowed to be longer than one element. This element is always flagged **wh**, indicating a Wh-movement. Concerning the right side, the authors are currently trying to find out whether it is possible to dispense with the special argument lists altogether. Consequently, they are left out in the table below.

However, this severe restriction should not be taken to mean that categories flagged **wh** are rare. On the contrary, as almost any constituent can be subject to Wh-displacement in Dutch, this is a substantial new source of lexical ambiguity. Namely, for each category assigned to a word in the lexicon, Delilah also assigns the same category with one of its right-searching arguments moved over to the left and flagged **wh**, or with one of its left-searching arguments moved over to the special stack and also flagged **wh**.

2.3.2 Grammar rules

The different restrictions on rule applications that are posed by the argument flags become apparent from the complete grammar of Delilah given in the following table. Unless otherwise indicated, the head flags of the resulting category follow from the virginity rules given above.

$\mathbf{/\hat{}0}$ $\quad \langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r} \rangle \backslash \mathbf{L} \backslash \mathbf{L_{sp}} / [\mathbf{y\hat{}0} \mid \mathbf{R}] \quad \langle \mathbf{y}, \mathbf{f'_l}, \mathbf{f'_r} \rangle \backslash [\,] \backslash [\,] / [\,] \rightarrow \langle \mathbf{x}, \mathbf{f''}, \mathbf{1} \rangle \backslash \mathbf{L} \backslash \mathbf{L_{sp}} / \mathbf{R}$

It is required that $\mathbf{f'_l} \neq \mathbf{wh}$.

$\mathbf{\backslash\hat{}0}$ $\quad \langle \mathbf{y}, \mathbf{f'_l}, \mathbf{f'_r} \rangle \backslash [\,] \backslash [\,] / [\,] \quad \langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r} \rangle \backslash [\mathbf{y\hat{}0} \mid \mathbf{L}] \backslash \mathbf{L_{sp}} / \mathbf{R} \rightarrow \langle \mathbf{x}, \mathbf{1}, \mathbf{f_r} \rangle \backslash \mathbf{L} \backslash \mathbf{L_{sp}} / \mathbf{R}$

$\mathbf{/\hat{}1}$ $\quad \langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r} \rangle \backslash \mathbf{L} \backslash \mathbf{L_{sp}} / [\mathbf{y\hat{}1} \mid \mathbf{R}] \quad \langle \mathbf{y}, \mathbf{f'_l}, \mathbf{f'_r} \rangle \backslash [\,] \backslash [\,] / [\,] \rightarrow \langle \mathbf{x}, \mathbf{f_l}, \mathbf{f''_r} \rangle \backslash \mathbf{L} \backslash \mathbf{L_{sp}} / \mathbf{R}$

$\mathbf{f'_l}$ may be either **0** or **wh** (the possibility of **0** was added to make preposed prepadj with pied piping possible).

$\mathbf{/\hat{}4}$ $\quad \langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r} \rangle \backslash \mathbf{L} \backslash \mathbf{L_{sp}} / [\mathbf{y\hat{}4} \mid \mathbf{R}] \quad \langle \mathbf{y}, \mathbf{f'_l}, \mathbf{f'_r} \rangle \backslash \mathbf{L'} \backslash \mathbf{L'sp} / [\,] \rightarrow \langle \mathbf{x}, \mathbf{f_l}, \mathbf{f''_r} \rangle \backslash [\mathbf{L'L}] \backslash [\mathbf{L_{sp}L'_{sp}}] / \mathbf{R}$

It is required that $\mathbf{f'_l} \neq \mathbf{wh}$. A condition that $\mathbf{f_l} \neq \mathbf{1}$ is left out to accommodate auxiliary inversion.

$\mathbf{\backslash\hat{}\ 4}$ $\quad \langle \mathbf{y}, \mathbf{f'_l}, \mathbf{f'_r} \rangle \backslash \mathbf{L'} \backslash \mathbf{L'_{sp}} / \mathbf{R'} \quad \langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r} \rangle \backslash [\mathbf{y\hat{}4} \mid \mathbf{L}] \backslash \mathbf{L_{sp}} / \mathbf{R} \rightarrow \langle \mathbf{x}, \mathbf{f''_l}, \mathbf{f''_r} \rangle \backslash [\mathbf{L'L}] \backslash [\mathbf{L_{sp}L'_{sp}}] / [\mathbf{R'R}]$

$\mathbf{/\hat{}50}$ $\quad \langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r} \rangle \backslash \mathbf{L} \backslash \mathbf{L_{sp}} / [\mathbf{y\hat{}50} \mid \mathbf{R}] \quad \langle \mathbf{y}, \mathbf{f'_l}, \mathbf{1} \rangle \backslash [\,] \backslash \mathbf{L'_{sp}} / [\,] \rightarrow \langle \mathbf{x}, \mathbf{f''_l}, \mathbf{1} \rangle \backslash \mathbf{L} \backslash [\mathbf{L_{sp}L'_{sp}}] / \mathbf{R}$

$\mathbf{/\hat{}56}$ $\quad \langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r} \rangle \backslash \mathbf{L} \backslash \mathbf{L_{sp}} / [\mathbf{y\hat{}56} \mid \mathbf{R}] \quad \langle \mathbf{y}, \mathbf{f'_l}, \mathbf{1} \rangle \backslash [\,] \backslash \mathbf{L'_{sp}} / [\,] \rightarrow \langle \mathbf{x}, \mathbf{f''_l}, \mathbf{1} \rangle \backslash \mathbf{L} \backslash [\mathbf{L_{sp}L'_{sp}}] / \mathbf{R}$

$\mathbf{/\hat{}59}$ $\quad \langle \mathbf{x}, \mathbf{0}, \mathbf{f_r} \rangle \backslash \mathbf{L} \backslash \mathbf{L_{sp}} / [\mathbf{y\hat{}59} \mid \mathbf{R}] \quad \langle \mathbf{y}, \mathbf{0}, \mathbf{1} \rangle \backslash \mathbf{L'} \backslash \mathbf{L'_{sp}} / [\,] \rightarrow \langle \mathbf{x}, \mathbf{0}, \mathbf{1} \rangle \backslash [\mathbf{L'L}] \backslash [\mathbf{L_{sp}L'_{sp}}] / \mathbf{R}$

$/\hat{}6$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\mathbf{y}\hat{}\mathbf{6} \mid \mathbf{R}]$ $\langle \mathbf{y}, \mathbf{f'_l}, \mathbf{f'_r}\rangle\backslash[\,]\backslash\mathbf{L'_{sp}}/\mathbf{R'}$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{1}\rangle\backslash\mathbf{L}\backslash[\mathbf{L_{sp}L'_{sp}}]/[\mathbf{R'R}]$

It is required that $\mathbf{f'_l} \neq \mathbf{wh}$. One might pose the condition that $\mathbf{R'} = [\,]$.

$\backslash\hat{}6$ $\langle \mathbf{y}, \mathbf{f'_l}, \mathbf{f'_r}\rangle\backslash[\,]\backslash\mathbf{L'_{sp}}/[\,]$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\mathbf{y}\hat{}\mathbf{6} \mid \mathbf{L}]\backslash\mathbf{L_{sp}}/\mathbf{R}$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{1}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash[\mathbf{L_{sp}L'_{sp}}]/\mathbf{R}$

$\backslash\hat{}7$ $\langle \mathbf{y}, \mathbf{f'_l}, \mathbf{0}\rangle\backslash[\,]\backslash[\,]/\mathbf{R'}$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\mathbf{y}\hat{}\mathbf{7} \mid \mathbf{L}]\backslash\mathbf{L_{sp}}/\mathbf{R}$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{1}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\mathbf{R'R}]$

$/\hat{}8$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\mathbf{y}\hat{}\mathbf{8} \mid \mathbf{R}]$ $\langle \mathbf{y}, \mathbf{f'_l}, \mathbf{f'_r}\rangle\backslash\mathbf{L'}\backslash\mathbf{L'_{sp}}/\mathbf{R'}$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{f''_l}, \mathbf{f'_r}\rangle\backslash[\mathbf{L'L}]\backslash[\mathbf{L_{sp}L'_{sp}}]/[\mathbf{R'R}]$

I would recommend requiring that $f'_r \neq \mathbf{wh}$, as adjunction to a sentence is a form of topicalisation, and thus should not be allowed to sentences which already contain an element that has undergone Wh-movement.

$\backslash\hat{}8$ $\langle \mathbf{y}, \mathbf{f'_l}, \mathbf{f'_r}\rangle\backslash\mathbf{L'}\backslash\mathbf{L'_{sp}}/\mathbf{R'}$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\mathbf{y}\hat{}\mathbf{8} \mid \mathbf{L}]\backslash\mathbf{L_{sp}}/\mathbf{R}$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{f''_l}, \mathbf{f''_r}\rangle\backslash[\mathbf{L'L}]\backslash[\mathbf{L_{sp}L'_{sp}}]/[\mathbf{R'R}]$

$\backslash\hat{}88$ $\langle \mathbf{y}, \mathbf{0}, \mathbf{0}\rangle\backslash[\,]\backslash[\,]/[\,]$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\mathbf{y}\hat{}\mathbf{88} \mid \mathbf{L}]\backslash\mathbf{L_{sp}}/\mathbf{R}$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/\mathbf{R}$

$/\hat{}9$ $\langle \mathbf{x}, \mathbf{0}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\mathbf{y}\hat{}\mathbf{9} \mid \mathbf{R}]$ $\langle \mathbf{y}, \mathbf{0}, \mathbf{f'_r}\rangle\backslash\mathbf{L'}\backslash\mathbf{L'_{sp}}/[\,]$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{0}, \mathbf{1}\rangle\backslash[\mathbf{L'L}]\backslash[\mathbf{L_{sp}L'_{sp}}]/\mathbf{R}$

$\backslash\hat{}9$ $\langle \mathbf{y}, \mathbf{f'_l}, \mathbf{0}\rangle\backslash\mathbf{L'}\backslash\mathbf{L'_{sp}}/\mathbf{R'}$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\mathbf{y}\hat{}\mathbf{9} \mid \mathbf{L}]\backslash\mathbf{L_{sp}}/\mathbf{R}$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{f''_l}, \mathbf{f''_r}\rangle\backslash[\mathbf{L'L}]\backslash[\mathbf{L_{sp}L'_{sp}}]/[\mathbf{R'R}]$

$\backslash\hat{}\mathrm{wh}$ $\langle \mathbf{y}, \mathbf{f'_l}, \mathbf{f'_r}\rangle\backslash[\,]\backslash[\,]/[\,]$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\,]\backslash[\mathbf{y}\hat{}\mathbf{wh}]/[\,]$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{wh}, \mathbf{f''_r}\rangle\backslash[\,]\backslash[\,]/[\,]$

$\mathbf{x}$ is one of $\{\mathbf{s}, \mathbf{s_{vn}}, \mathbf{prepadj}, \mathbf{pp}\}$; in addition, if $\mathbf{x} \in \{\mathbf{prepadj}, \mathbf{pp}\}$, then $\mathbf{f'_l} = \mathbf{f'_r} = \mathbf{0}$. It is not sure whether also $\mathbf{f_r}$ should be $\mathbf{0}$ in this case.

A special set of rules comes from so-called automorphism types, i.e. types that can combine with categories headed by almost any basic category to form a new category of that head.

We have the following direct type rewritings, the use of which depends on whether the automorphism, which is always the primary type, appears on the left or on the right:

- $\langle \mathbf{sentadj}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\,]$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\mathbf{x}\hat{}\mathbf{8}]$

  Here $\mathbf{x} \in \{\mathbf{s_{vn}}, \mathbf{vp}, \mathbf{vp_t}, \mathbf{vp_d}, \mathbf{vp_p}\}$
- $\langle \mathbf{sentadj}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\,]$ $\rightarrow$ $\langle \mathbf{s}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\mathbf{s}\hat{}\mathbf{8}]$
- $\langle \mathbf{sentadj}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\,]\backslash[\,]/\mathbf{R}$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\mathbf{x}\hat{}\mathbf{8}]\backslash[\,]/\mathbf{R}$

  Here $\mathbf{x}$ is either $\mathbf{s}$ or $\mathbf{q}$.
- $\langle \mathbf{prepadj}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\,]$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\mathbf{x}\hat{}\mathbf{4}]$

  Here $\mathbf{x} = \mathbf{s_{vn}}, \mathbf{vp}, \mathbf{vp_t}, \mathbf{vp_d}$, or $\mathbf{vp_p}$.
- $\langle \mathbf{prepadj}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\,]$ $\rightarrow$ $\langle \mathbf{s}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\mathbf{s}\hat{}\mathbf{0}]$
- $\langle \mathbf{prepadj}, \mathbf{wh}, \mathbf{f_r}\rangle\backslash[\,]\backslash[\,]/[\,]$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{wh}, \mathbf{f_r}\rangle\backslash[\,]\backslash[\,]/[\mathbf{x}\hat{}\mathbf{0}]$

  Here $\mathbf{x}$ is either $\mathbf{s}$ or $\mathbf{s_{vn}}$.
- $\langle \mathbf{prepadj}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\,]$ $\rightarrow$ $\langle \mathbf{x}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\mathbf{x}\hat{}\mathbf{8} \mid \mathbf{L}]\backslash\mathbf{L_{sp}}/[\,]$

  Here $\mathbf{f_l} \neq \mathbf{wh}$, and $\mathbf{x} \in \{\mathbf{s}, \mathbf{q}, \mathbf{s_{vn}}, \mathbf{vp}, \mathbf{vp_d}, \mathbf{vp_p}, \mathbf{vp_t}, \mathbf{ap}\}$; the rule is invalid if $\mathbf{L} = [\,]$ and $\mathbf{L_{sp}} = [\mathbf{x}\hat{}\mathbf{wh}]$.
- $\langle \mathbf{prepadj}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\,]$ $\rightarrow$ $\langle \mathbf{np}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\mathbf{np}\hat{}\mathbf{0} \mid \mathbf{L}]/[\,]$

  The rule is invalid if $\mathbf{L} = [\,]$ and $\mathbf{L_{sp}} = [\mathbf{np}\hat{}\mathbf{wh}]$, and it is required that at least one head flag of the secundary category is $\mathbf{1}$.
- $\langle \mathbf{r}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\,]\backslash[\,]/\mathbf{R}$ $\rightarrow$ $\langle \mathbf{y}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash[\mathbf{y}\hat{}\mathbf{8}]\backslash[\,]/\mathbf{R}$

  Here $\mathbf{y}$ is either $\mathbf{s}$ or $\mathbf{q}$; the secundary category is required to have $\mathbf{f'_r} = \mathbf{0}$.
- $\langle \mathbf{r}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/\mathbf{R}$ $\rightarrow$ $\langle \mathbf{y}, \mathbf{f_l}, \mathbf{f_r}\rangle\backslash\mathbf{L}\backslash\mathbf{L_{sp}}/[\mathbf{y}\hat{}\mathbf{f} \mid \mathbf{R}]$

  Here either $\mathbf{y} = \mathbf{s}$ and $\mathbf{f} = \mathbf{0}$ or $\mathbf{y} = \mathbf{s_{vn}}$ and $\mathbf{f} = \mathbf{6}$.

2.4 Relation to the CCG and LIG formalisms

Delilah's grammar as it stands does not conform to the CCG formalism. In the following, we will indicate the obstacles, and show if and how they may be circumvented. It turns out that most features of the Delilah grammar are expressible in an LIG.

2.4.1 Formal considerations

The category format First, the format of the categories themselves has been changed, in that the left and right searching arguments are sorted out and grouped into two separate argument lists or 'stack's. Of course, this gives us greater freedom in the order of application of the reduction rules: we have the choice between reducing to the left and to the right, whereas this choice is restricted by the ordering of the arguments in a standard categorial framework. This is easily seen to add to the strong recognizing capacity of the grammar.

In principle, if we were to achieve the same strong recognizing capacity using only standard form categories, for each category $\boldsymbol{c}$ assigned to a word we would have to add all categories formed by mixing up the left and right searching arguments in another way to this word's categorisation.

It is important to note that in the standard Lambek calculus, all these 'mixed-up' categories are equivalent, due to the Lambekian theorem $(a/b)\backslash c \leftrightarrow (a\backslash c)/b$.

Any derivation tree over the Delilah grammar may be transformed into one containing only categories in the standard form by proceeding as follows (not considering the special arguments):

1. The label of the starting point (the root) is in standard form, as it is atomic.
2. Any non-atomic category $\mathcal{C}$ in standard form may be written as $\mathbf{a}\ \mathcal{L}_0\mathcal{R}_1\mathcal{L}_1 \ldots \mathcal{R}_m\mathcal{L}_m$ with $\mathbf{a}$ atomic, and $\mathcal{L}_i$ and $\mathcal{R}_i$ lists of left and right slashed arguments, respectively. Suppose now that, at some point in the derivation tree, we have the nodes

$$\mathcal{C}$$

$$\mathbf{a\backslash L/[yR]} \qquad \mathbf{y\backslash L'/R'}$$

   where the parent node is in standard form, and the children in the form used by Delilah. Then depending on the precise Delilah rule used, either $\mathrm{rev}(\mathbf{L'L})$ or $\mathrm{rev}(\mathbf{LL'})$ is the concatenation of all $\mathcal{L}_i$; and analogously, the right searching arguments together form either $\mathrm{rev}(\mathbf{R'R})$ or $\mathrm{rev}(\mathbf{RR'})$. This shows how to share the parent category's arguments among its child categories.

Suppose that we let Delilah produce all grammatical analyses of one particular sentence. If we execute the above construction on all of these trees, we obtain a new partial lexicon by considering the yields of the resulting trees. It might be that this partial lexicon contains categorisations that were not in the original lexicon, but may be obtained from it by using the above mentioned theorem of the Lambek calculus. In principle, even all mix-ups of a given category might turn out to be effectively needed in order to form all grammatical analyses.

However, due to the flag conditions, the rule application possibilities are limited in such a way that most lexical entries need only one standard form of each of their assigned categories. This means that it is possible to reformulate the lexicon using only one list of arguments for each category; exception being made for the special arguments, who keep their own list.

There are examples of categorisations which seem to require more than one mix-up. One of which is given by the sentence

hij zei dat Henk dacht dat ik wilde werken
he said that Henk thought that I wanted to-work

In our partial lexicon, obtained as described above, 'wilde' is categorized both as $\langle \mathbf{s_{vn}, 0, 0} \rangle \backslash \mathbf{np\hat{\ }0/s_{sub}\hat{\ }6}$ and $\langle \mathbf{s_{vn}, 0, 0} \rangle \mathbf{/s_{sub}\hat{\ }6\backslash np\hat{\ }0}$. When we examine the corresponding derivation trees, this division appears to depend on whether 'dacht dat ik wilde werken' or 'henk dacht', respectively, is the parent constituent of 'dacht'.

However, this may be considered as a form of spurious ambiguity. I would prefer selecting one of the two analyses as a normal form (i.c., the right branching version having 'dacht dat ik wilde werken' for a constituent), and include only that one in the lexicon in standard form.

Use of flags and unary rules Secondly, the general format of a rule makes use of flagged atomic categories. These flags restrict the possible instantiations of rules. Furthermore, the flags on the heads of the primary and result categories need not be the same, yielding different heads. This is impossible within a categorial framework.

The flags on categories used as arguments are lexically assigned and do not vary with cancellations. However, we have the same problem we had with the head flags: we want the argument **y^flag** to match with the secondary head **y**, or even with **y** with some head flags attached to it. A literal match is impossible, so we would have to abstract away from these restrictions in order to remain within the CCG framework.

One could save the parsing issue this way: if the abstracted grammar is a CCG, which is parsable in polynomial time, then our language is polynomially parsable as well — at least in the bottom-up framework described in chapter 4. Here, enforcement of the restrictions imposed by the head flags is straightforward and yields a proper subset of the unrestricted language.

I would like to point out here that as far as the flags are concerned, the Delilah grammar is more like a LIG, as this formalism incorporates the possibility for changing the 'category' heads during the cancellation; as well, this formalism allows the use of unary productions like the direct rewriting rules for adjunct categories, which are not part of the CCG formalism either.

Another analogy that may be drawn is with the modal $\Box$-operator suggested by Morrill to deal with locality constraints on binding and reflexivisation. Hepple [Hep92] shows how this operator, which may be attached both to heads and arguments of categories, may be incorporated into a categorial grammar framework, and parsed.

The special arguments Thirdly, we have the special arguments. With these, the cancellation rules use a different merging rule: when merging the list **L** from the primary and the list **L'** from the secondary category, one would normally form the list **L'L**. With special lists, however, this order is reversed. An important point with this is that at least the count properties and the directionality are preserved under the use of special arguments: the only difference is in the ordering, and thus the only difference in the corresponding language is also in the word order.

With Delilah, the use of special arguments is restricted to the implementation of Wh-movement, which may span arbitrarily long distances, and does not interfere with the head-argument relations occurring between the 'original' and the 'final' location of the moved element (speaking in the transformational language).

However, the use of special arguments (i.c. the accommodation of Wh-gaps) still is an essential addition to the LIG-formalism. I do not know what family of languages is derivable when this feature would be used in full strength, instead of the restricted fashion used at present. However, the present formalism may be said to generate a LIL with the additional application of Wh-movement in the sense of transformational grammars (cf. [BH89, p. 189 sqq.]).

2.4.2 Boundedness conditions

A last obstacle is the secondary categories occurring in instantiations of Delilah rules are not necessarily bounded, as they are in a CCG. This property of CCGs follows from the fact that only a finite number of rule instantiation models is chosen beforehand, so that the arity of the secondary category is bounded from above.

In Delilah, there definitely are bounds on the arity of secondary categories. As the lexical assignment is linear, any category occurring in a rule instantiation may not be longer than the sentence length. This follows from the fact that for any basic category except the designated one, there must be an eventual match between the total number of positive and negative occurrences of this basic category in the sentence's categorisation. This principle, known as the Van Benthem count, holds for Delilah because in any rewriting step, the total of positive and negative occurrences of each basic category does not change (see above, p. 7, and cf. [Cre93, p. 28]). As only one positive occurrence of some basic category may be introduced for each word in the sentence, we see that no secondary category (and no primary category, either) can have more arguments (i.e., basic categories occurring negatively) than the sentence length.

On the other hand, we can show that there are constructions which give rise to arities which are not bounded by the sentence length. Consider the following example:

ik zou Jan Piet Wim willen zien zien zien
I would Jan Piet Wim like see see see
I would like to see Jan see Piet see Wim

This type of sentence rapidly becomes unwieldable as the number of 'zien's increases, but it is definitely productive. Now the final verbal cluster of these sentences is taken to be one constituent of type

$$\mathbf{vp\backslash[np\hat{}0, np\hat{}0, np\hat{}0]/[\,]}$$

which results of the merging of $\mathbf{vp\backslash[\,]/[vp\hat{}9]}$ for 'willen' and the mentioned category for 'zien zien zien'. Thus the length of the secondary category increases linearly with the sentence length. Furthermore, there is no derivation of these sentences in Delilah which circumvents this particular cancellation, as may be easily shown experimentally by enforcing a bound of, say, 4 on the length of secondary categories involved in any cancellation. It should be noted that I believe this to be the only example, together with the analogous situation for flag **59**.

This is a serious obstacle, as it undermines the central consideration which renders parsing of CCGs polynomial in principle (see p. 27 for more details). However, it is possible to solve the problem with a modification of the grammar rule for flags **9** and **59** (right searching). Let us replace the rules by

$$\mathbf{/\hat{}59} \quad \mathbf{\langle x, 0, f_r\rangle\backslash L\backslash L_{sp}/[y\hat{}59 \mid R]} \quad \mathbf{\langle y, 0, 0\rangle\backslash L'\backslash L'_{sp}/R'} \;\rightarrow\; \mathbf{\langle x, 0, 1\rangle\backslash[L'L]\backslash[L_{sp}L'_{sp}]/[R'R]}$$

$$\mathbf{/\hat{}9} \quad \mathbf{\langle x, 0, f_r\rangle\backslash L\backslash L_{sp}/[y\hat{}9 \mid R]} \quad \mathbf{\langle y, 0, 0\rangle\backslash L'\backslash L'_{sp}/R'} \;\rightarrow\; \mathbf{\langle x, 0, 1\rangle\backslash[L'L]\backslash[L_{sp}L'_{sp}]/[R'R]}$$

Now the roles of primary and secondary categories are reversed due to a different order of reduction. Still the constituent 'zien zien zien' has the same category as mentioned above, but it is always the primary constituent in a reduction, which poses no problems with the CCG formalism: the primary constituent is allowed to be of unbounded length.

2.4.3 Conclusion

I would like to point out that it turns out that we can bring the Delilah grammar into CCG format with modal operators, head flags, and Wh-movement, or into LIG format with Wh-movement:

1. We may return to the native format for categories without losing significant grammatical analyses; we only lose some analyses which may be considered redundant. This replaces the left and right argument stacks with one mixed stack. It remains possible to enforce the emptiness conditions on one-sided stacks of secondary categories, as those are bounded in length. Thus we may in principle give all possibilities, and thus enforce any constraints we wish on them. There are no one-sided emptiness conditions on primary categories, which would have been difficult to enforce, as this might have taken an unlimited amount of stack-travelling.
2. We may describe the set of flagged head and argument categories as a nonterminal and stack symbol alphabet, respectively, thus conforming to the LIG formalism.
3. By replacing two grammar rules, we can convince ourselves that all secondary categories occurring in derivations of the grammar are bounded in length. This is clear by inspection of the lexicon. Thus we may apply any fast parsing techniques developed for the discussed grammar formalisms. It is clear that handling the (rather sporadic) Wh-arguments poses no serious problems, due to the severe restrictions on their occurrence (at most one per category).

In Chapter 5 we will discuss Vijay-Shanker and Weir's polynomial parsing algorithm for CCGs and LIGs (developed in [VW93]), and we will discuss how we can find a polynomial algorithm for parsing the Delilah grammar.

Chapter 3

Investigating Delilah's Count Protocol

3.1 Complexity issues for NLP algorithms

Let $x_1x_2 \dots x_n$ be a sentence over Delilah's lexicon. The type assignment in Delilah's lexicon, f, assigns to each x_i a finite number of categories. Let us denote the number of types assigned to x_i, $\#f(x_i)$, by c_i. Thus the whole sentence may be assigned a number of $\prod_{i=1}^{n} c_i$ type strings, some of which may be grammatical.

In a real world grammar for any natural language, it is probable that most c_i's will be greater than one. This clearly yields a number of possible type strings that is exponential in n. Thus any algorithm wanting to pick out all type strings that yield grammatical analyses in polynomial time cannot compute all these type strings, or execute any fixed task, however small, for each one of them. Put in another way, the algorithm will want to discard most of these type strings without computing them all, and retaining only polynomially many of them which will be tested for grammaticality.

We must make one caution, however. Sentences of natural languages frequently display forms of so-called genuine ambiguity, like homophony and unclear binding of prepositional phrases. These are examples of semantically-conditioned ambiguity which induces several distinct grammatical analyses of one and the same type string. There even exist families of strings whose number of grammatical analyses grows exponentially with their lengths, as the following example shows. Consider s_i = de man (met de fiets)^i loopt *the man (with the bicycle)^i walks.* Different binding possibilities for the prepositional phrases yield at least 2^{i-1} grammatical analyses for these sentences.

This means that even if we succeed in filtering out the polynomially many possibly valid type strings[1], we might still need an exponential amount of work to obtain all possible parse trees. Therefore, we will assume that the user is willing to grant the parser all time needed, provided that the time spent on each *parse tree* is polynomial in the length of the user input.

In the remainder of this chapter, we describe the implementation of this first selection stage used by Delilah. Its structure is roughly the following: Delilah generates sets of partial type strings from left to right, starting with the set of length-one strings assigned to x_1. These are defined as follows: for i from 1 to n, the set of i-partial type strings is a set $S_i \subset f(x_1) \times f(x_2) \times \dots \times f(x_i)$. We take $S_1 = f(x_1)$, and at step i, we consider each element $s \bullet t$ of the Cartesian product $S_{i-1} \times f(x_i)$, and make it a member of S_i if it satisfies certain simple rules. Of course, we ensure that these rules are necessary conditions for grammaticality, so that only ungrammatical strings are rejected.

It will turn out, however, that the count algorithm most probably needs exponential time to execute in the average case. This does not imply that it cannot be practical for short sentences and/or a restricted lexicon; however, with a full-fledged lexicon, reflecting all subtle differences in word usage possibilities, and real world sentences the execution time rapidly becomes unacceptable. We will give concrete examples of this behaviour.

[1] I do not think that there exist sentences in a natural language which possess exponentially many valid categorisations.

3.2 The count protocol

3.2.1 The data structure

As a fundamental data structure, the count protocol uses the 'well-formed bag'. This structure is completed as the words of the input sentence are processed, and contains all possible categorisations of the prefix of the sentence which has been considered up to now, where the processing direction is left to right.

These partial type lists take the form [*list*]:*register*. The *list* part is a list of Delilah categories, the form of which has been discussed in the previous chapter. The so-called register keeps track of the following information:

1. for each basic category of the Delilah grammar:
   (a) nh, the number of uncancelled heads (positive occurrences) up to now
   (b) rnz, the number of uncancelled right searching arguments introduced up to now
   (c) rsz, the number of heads cancelled by a right searching argument
   (d) lsh, the number of heads cancelled by a left searching argument
   (e) (For the right wing of a coordinated sentence, we add lnz, the number of uncancelled left searching arguments introduced up to now)
   (f) A flag indicating whether any of these counter values have changed up to now. I am under the impression that this flag has no significant use in the program.
2. *sumrnz*, the total number of uncancelled right searching arguments of all basic categories
3. (For the right wing of coordinated sentences: *sunlnz*, the analogous value for left searching arguments)
4. *tdstack*, a stack containing the currently valid category blockades; this so-called top-down filter is described in more detail below.
5. $whstatus$, representing the current state in the Wh-network, a finite state network controlling the occurrence of Wh-displacement.

3.2.2 The count proper

From the fact that the category assignment used in the Delilah lexicon is linear, and the reduction rules are count-invariant [Cre93, p. 12, 28], we can deduce the following conditions which must be fulfilled for grammatical strings.

Suppose our string x has length n, and we are currently examining position m. Then we know that

1. All left searching arguments introduced up to now must have been cancelled by some head
2. The total number of unsatisfied right searching arguments may not exceed the number of words remaining in the sentence

Therefore, if the word at position m is categorized as c, then

1. If c introduces a left searching argument b, then the number of unsatisfied heads of this type must be greater than zero, and the two occurrences cancel each other
2. If c introduces a right searching argument b, then the total number of right searchers *sumrnz* may not exceed $n - m$, and will be increased by 1
3. Let c's head be of type b; then if the number of unsatisfied right searchers for type b is greater than zero, the head is cancelled against one searcher

When processing coordinated sentences, which are interpreted as two sentences having both lost part of their content by ellipsis, the conditions on left searching arguments cannot be quite so strict. In fact, we can only apply the analogon of the test for right searching arguments, i.e., there cannot be too much of them.

We can derive one more necessary condition: when the whole string has been processed, all arguments and heads must have been cancelled except one head of a distinguished basic category. In the Delilah case, it is even required that one and only one head of a distinguished category appears in the whole sentence. This property follows from the grammar (no lexically assigned category asks for an argument of a distinguished category); there is one exception, namely the Wh-element **q/s^1**.

3.2.3 Keeping track of Wh-displacement

As indicated in the preceding chapter (p. 9), all lexically assigned categories have a counterpart with one of their arguments replaced by a **wh**-flagged argument. This means that the corresponding constituent has been moved from its original place and is to be found somewhere to the left of it. However, in Dutch only one Wh-movement is allowed per sentential domain, be it a normal sentence or a subordinate clause[2]. The landing position of the displaced constituent is the leftmost position of some enclosing domain. It is a subtle linguistic question to tell what and how many domain boundaries the displacement may cross; for a detailed description of these phenomena, we refer to [BH89, Chapter 7].

However, it is possible to keep track of the domain structure of the input sentence, and hence to control the number of Wh-moved constituents that are allowed to appear at the current input position and rightward.

This so-called Wh-filter is implemented in Delilah in the form of a finite state automaton, whose state is updated with each processing of a head or an argument, as described above. It enforces some more necessary conditions for sentencehood on the string categories assigned to the input words.

3.2.4 Blocking of specific categories

Consider the following situation: while examining position m, we find that the input word on that position has category c (among others, possibly), where c has the form $\mathbf{x\backslash L/R}$. Let y be a basic category occurring in $\mathbf{R}$. Obviously, somewhere to the right of the current position there must be a word having a category headed by y. The question is: what categories can occur in between? Or, formulated differently, it is possible to predict that some atoms cannot occur as head of a category between c and the y-headed category?

Within a Context-Free Grammar framework, this question can be easily answered: we can compute the set of nonterminals which are reachable from a given string of nonterminals. In our framework, it is possible that disharmony occurs, i.e., that atoms introduced as right searching arguments by words left of our word, or as left searching arguments by words to the right of the y-headed category, are satisfied by a head occurring in between those two.

These considerations show that a complete answer to the given question is difficult. However, there are some obvious cases where applying our linguistic intuition leads to substantial savings on time and space requirements. We give some examples. Firstly, when we encounter a right searching $\mathbf{s_{vn}}$ argument, we expect a subordinate clause to the right. This means that no **s** or **q**-headed categories are allowable until we have ascertained that the subordinate clause is ended. On the other hand, no $\mathbf{s_{vn}}$ headed category is allowable if we are in a main sentence. Secondly, when processing either a main sentence or a subordinate clause, we know that there must be exactly one finite verb. Thus, if we have been able to find the finite verb, all other verbs can only be infinitives or participles.

Delilah contains an implementation of these considerations, called the Top-Down filter, which dramatically reduces the number of possible alternatives especially in case of verbal clusters.

[2]Probably, an exception should be made for parasitary gaps, which are not supported in Delilah.

3.3 Time and space complexity

After having described the various tests that Delilah applies to tell the ungrammatical type string prefixes from the grammatical ones, we will now analyze the complexity of this algorithm. Our unit of measure for the size of the input will always be the number of words in the input sentence.

It is clear that the tests mentioned above are all straightforward — they take constant time to check for each input position, and linear time for a whole type string. This means that the complexity of the count protocol is determined by the number of type string prefixes that are currently under investigation — in other words, by its space complexity.

3.3.1 Worst case

We now ask ourselves what is needed to prove polynomial complexity for the count algorithm in worst case. As described above, at any input position Delilah constructs a set of all categorisations of the currently processed part of the input sentence that may still lead to a grammatical analysis, and then applies some tests to all of those type strings. If we can find an infinite family of input sentences with the property that at some input position there are exponentially many possible type strings, we have proved that the count algorithm takes exponential space and time in this case — notwithstanding the fact that the algorithm may eventually succeed in cutting this large array of possibilities down to just a polynomially large set.

We will present an example of a family of sentences accepted by the Delilah grammar and parser, which indeed take exponential time and space. In this case, the count algorithm also gives an exponential number of possible categorisations, thus causing the parser stage to take exponential time as well.

Consider the sentences 'ik (zie dat ik)^i werk' (*I (see that I)^i am-working*). This is an infinite family of sentences which are unambiguously accepted as grammatical in Dutch. In the tables below, we present the running time of the count algorithm for the first members of this family, as well as the number of categorisations surviving the count stage, for versions 1 and 2 of the Delilah system. We would like to point out that within the Delilah grammar and lexicon there is only one categorisation that leads to a valid parse tree. It is given (in a shorthand format) by $\mathbf{np\ s\backslash np\hat{}wh}$ for $\boldsymbol{i = 0}$, by $\mathbf{np\ s\backslash np\hat{}wh/s_{sub}\hat{}6(s_{sub}/s_{vn}\hat{}6\ np\ s_{vn}\backslash np\hat{}0/s_{sub}\hat{}6\ np)^{i-1}\ s_{vn}\backslash np\hat{}0}$ for $\boldsymbol{i \geq 1}$. One reason for the multitude of possible categorisations is the ambiguity between the categories $\mathbf{s\backslash np\hat{}wh}$, $\mathbf{s_{vn}\backslash np\hat{}0}$, and $\mathbf{s_{vn}\backslash np\hat{}wh}$ all assigned to 'zie'. It is near to impossible for the diverse count techniques to distinguish between these three.

i	count time (V1)	survived (V1)	count time (V2)	survived (V2)
0	10ms	1	10ms	1
1	30ms	12	40ms	12
2	250ms	56	430ms	64
3	2090ms	304	4450ms	428
4	18310ms	1792	46150ms	3244

The above example being quite generic, we may conclude that the count algorithm exhibits exponential behaviour in worst case.

3.3.2 Average case — experiments

Due to the very complex nature of the input space, it is difficult to achieve a formally proven complexity measure for the average case of Delilah's input. However, we will present both experimental data and a theoretical argument which seem to show that even in the average case, the count algorithm needs exponential time and space to execute.

The experimental data we will present at once show the difference between versions V1 and V2. In version V1, the lexicon allows for much more modest (sub)categorisation possibilities than in version V2. To give one example, the introduction of so-called **rnp**s (**np**s of special form, all containing an *r*, like 'waar' (*where*), 'daar' and 'er' (both *there*)) has led to a doubling of the number of (sub)categorisations available for prepositions. Likewise, the number of categorisations for finite verbs in particular has increased.

The sentences we present are taken from a fixed set of example sentences tailored to test particular features of the Delilah grammar. However, we have removed all coordination phenomena from our sentences, as we have excluded the processing of coordination in Delilah from our research. For each sentence, we give the output of Delilah for this input. This output gives the number of possible categorisations found up to the current input position. This enables one to find out which words are the most responsible for the growth of the list of well-formed type lists.

From these data, it is obvious that the general tendency is toward exponential growth, except when approaching the end of the input. Of course, the number of right searching arguments that can be introduced by the last few categories in a type list is limited; and this limits the number of alternatives for those last words. Furthermore, it is clear that the verb forms in particular attribute a large number of alternatives. For a large part, this is due to the opposition between heads of main sentences (categories **s** and **q**) and of subordinate clauses (category $\mathbf{s_{vn}}$), and to the treatment of Wh-displacement (see p. 2.3.1).

Sentence 6

misschien is Henk door de man in de auto gedwongen te zeggen dat hij mij niet wilde laten werken
maybe has-been Henk by the man in the car forced to say that he me not wanted-to let work
maybe Henk has been forced by the man in the car to say that he would not let me work

Count version 1:

misschien 1 is 4 henk 4 door 6 de 6 man 6 in 8 de 8 auto 8 gedwongen 13 te 13 zeggen 29 dat 58 hij 58 mij 58 niet 58 wilde 202 laten 32 werken 32

Of 8820000 possible type assignments, (the size of the Cartesian product of all lexical assignments), 29 possibilities remain after having gone through the sentence and applying the count tests. This takes Delilah 540ms.

Count version 2:

misschien 1 is 10 henk 10 door 20 de 20 man 20 in 80 de 80 auto 80 gedwongen 168 te 168 zeggen 424 dat 1272 hij 1272 mij 1272 niet 1272 wilde 4954 laten 285 werken 216

Of 622080000 possible type assignments, 98 survive. This takes Delilah 13250ms.

Sentence 7

omdat ik Henk niet met de pop wilde laten spelen, werd ik door de vrouw met het mooie boek met de mooie gedichten gedwongen te proberen Henk Agnes een auto te laten geven
because I Henk not with the doll wanted-to let play was I by the woman with the beautiful book with the beautiful poems forced to try Henk Agnes a car to let give
because I would not let Henk play with the doll, I was forced by the woman with the beautiful book with the beautiful poems to try to make Henk give Agnes a car

Count version 1:

omdat 1 ik 1 henk 1 niet 1 met 1 de 1 pop 1 wilde 2 laten 1 spelen 1 werd 3 ik 3 door 4 de 4 vrouw 4 met 5 het 5 mooie 5 boek 5 met 6 de 6 mooie 6 gedichten 6 gedwongen 7 te 7 proberen 8 henk 8 agnes 8 een 8 auto 8 te 8 laten 8 geven 9

Of 793800000 possible type assignments, 9 survive. This takes Delilah 110ms.

Count version 2:

omdat 1 ik 1 henk 1 niet 1 met 4 de 4 pop 4 wilde 12 laten 16 spelen 10 werd 21 ik 21 door 42 de 42 vrouw 42 met 168 het 336 mooie 504 boek 504 met 2016 de 2016 mooie 4032 gedichten 8064 gedwongen 26376 te 26376 proberen

Here we interrupted the Prolog processing after 125 seconds, to avoid memory shortage problems.

Sentence 69

welke mannen die ik had proberen te laten werken werden door de vrouw aan die ik had gezegd dat henk mij geen boek durfde te geven gedwongen agnes met het boek met de mooie gedichten te laten spelen
which men which I had tried to let work were by the woman to whom I had said that Henk me no book dared to give forced Agnes with the book with the beautiful poems to let play
which men which I had tried to get working were forced, by the woman to whom I had said that Henk did not dare to give me a book, to let Agnes play with the book with the beautiful poems

Count version 1:

welke 2 mannen 2 die 2 ik 2 had 8 proberen 15 te 15 laten 31 werken 41 werden 49 door 54 de 54 vrouw 54 aan 79 die 79 ik 79 had 372 gezegd 242 dat 484 henk 484 mij 484 geen 484 boek 484 durfde 1410 te 1410 geven 2102 gedwongen 4073 agnes 4073 met 5325 het 5325 boek 5325 met 6729 de 6458 mooie 5040 gedichten 5040 te 2437 laten 835 spelen 835

Of 4065949440000 possible type assignments, 532 survive. This takes Delilah 23620ms.

Count version 2:

welke 1 mannen 2 die 2 ik 2 had 12 proberen 20 te 20 laten 34 werken 39 werden 96 door 153 de 153 vrouw 153 aan 612 die 612 ik 612 had 4284 gezegd 6222 dat 18554 henk 18554 mij 18554 geen 18554 boek 18554 durfde 37092 te 37092 geven

Here again we broke off the computation after 96 seconds.

3.3.3 Average case — theoretical considerations

In my opinion, it is safe to say that any algorithm based on the count protocol as described in the first section above will display exponential growth behaviour in the average case. Of course, it is possible to refine the tests applied to the partial categorisations of the input by deriving more necessary conditions for sentencehood.

For example, an improvement of the algorithm could be to try and identify subdomains in the input which are relatively isolated from the rest of the sentence, and processing those independently — i.e., ignoring the categorisations found for the preceding part of the input. If we could find conditions under which the categorisations of the preceding part and of the identified subdomain are largely independent on each other, we could execute the count protocol for the two parts separately, and only then form the Cartesian product of the resulting partial categorisations, instead of reexecuting the count algorithm on the subdomain for each categorisation of the preceding part.

However, the basic idea of forming the Cartesian product of a number of sets of partial categorisations is itself an essential source of exponential behaviour. Whether we take those partial categorisations to span a sequence of words or one word only, their number will be linear in the length of the input sentence — at least within infinite families of sentences, which is the interesting situation when discussing complexity. Thus if the ith sentence part receives c_i categorisations, the sentence as a whole will receive some fraction of $\prod_{i=1}^{m} c_i$, where the number of parts m is linear in the sentence length n. This product is an exponential function in n, unless we can manage that at most $c \log n$ of the c_i is larger than 1 (with c some constant, independent of n). Of course, this latter bound is very difficult to achieve — it implies that within an infinite family of sentences, if n is large enough each sentence part should receive one unambigious categorisation on the basis of count tests alone. Namely, the number of sentence parts grows linearly with n, whereas the number of ambiguous parts should not exceed $c \log n$ for some constant c. If n is large, this means that there must not be some structural source of ambiguity in the sentence, in order to achieve a polynomial number of possible categorisations. In view of the excessive ambiguity of natural languages on the word level, this is bound to fail.

In my opinion, the solution is to abandon Cartesian products all together. Specifically, this implies that a parsing algorithm should avoid trying to generate all possible categorisations of the whole input sentence, before invoking the parser proper to make the final separation between grammatical and ungrammatical categorisations. Instead, it should try to decide grammaticality once and for all on parts of the sentence, and combine these partial analyses with others to form an analysis of the whole sentence. This opinion is fed by the existence of *chart parsers*, which obtain excellent results (both memory usage and execution time are cubic in the input size) on Context Free Grammars, and whose design exactly follows the lines just sketched. However, it remains to be investigated whether this class of algorithms is applicable to the type of grammar used by Delilah, which is not context free. I refer to Chapter 5 for an elaboration of this suggestion.

Chapter 4

Investigating Delilah's Parser

As indicated, the syntactical component of Delilah consists of two stages. The first stage is the count protocol which we discussed in the previous chapter. The outcome of this stage is a set of strings of lexically assigned categories. In the second stage, each of these categorisations of the input is offerred to the parser, which decides whether a grammatical analysis exists for this type string. To a large extent, this parser is a deterministic algorithm. In this chapter, we will describe the diverse features that combine in bringing this property about.

4.1 Shift-reduce parsers

The backbone of the parser is the well-known shift-reduce algorithm. The general strategy of this algorithm is as follows: we maintain a stack of partial results obtained up to now. The input is traversed from left to right; at each point, we first push (**shift**) the current input word's category onto the stack, and then we may execute one or more **reduce** steps: check whether the topmost n stack symbols together form the left hand side of a reduction rule of the grammar; apply the rule, and replace the n symbols with the right hand side of the rule. The algorithm is said to have succeeded when the input is exhausted and the stack consists of only the initial symbol of the grammar. If, on the other hand, the contents of the stack may not be reduced to only the initial symbol and there is no more input left, the parser has blocked (cf. [Per85]).

There is much indeterminacy involved in the **reduce** steps. At any time, several reduction rules may be applicable and we must make a choice between them. Furthermore, it is possible not to do the reduction at all, but rather to continue with the next input word. On the other hand, we may execute several reduction steps in succession if the outcome of one reduction makes some new reduction rule applicable.

Consequently, executing this algorithm without further provisions will most probably require massive backtracking and will be inefficient. Thus we need a means to cut down on the possible moves in each situation. Pereira uses the term *oracle* for this, and defines it as a device that examines the moves that are possible in a configuration and forbids those moves that it knows will only lead to blocking move sequences [Per85, p. 310]. This may still leave the choice between a number of moves none of which can be told to lead to a dead end.

Thus there are two ways of making the parsing stage more efficient: reducing the number of applicable rules at any moment, and increasing the predictive quality of the oracle, using look-ahead perhaps. Both features are present in the Delilah system, and will be discussed in the following sections.

4.2 Decreasing rule applicability

The first problem to cope with is the problem of spurious ambiguity, i.e., the generation of multiple derivation trees for the grammatical analysis of one string of lexically assigned categories. As indicated in a previous chapter (see p. 14), there are sentences in our language which do require multiple analyses (even exponentially many in some cases), which are semantically distinguishable. However, one wants to reduce as much as possible the generation of semantically equivalent derivation trees. Of course the concept of semantical equivalence is somewhat fleeting, as it varies with the exact semantic representation of sentences.

However, there are several syntactic considerations that lead to a preference for one type of derivation over another. We will give those in the following.

4.2.1 Closedness conditions

Several categories are subjected to closedness conditions before they are accepted as secondary constituents in a reduction. For example, arguments of **np** type always carry a **0** flag, meaning that a category with head **np** may only be cancelled if all its arguments have been satisfied, and thus it may be said to represent a closed constituent of the sentence. These closedness conditions translate to emptiness conditions on argument lists in the grammar.

As closedness of a category eliminates the possibility of disharmonious composition, it directly implies the visible closedness of the corresponding part of the input sentence. This may pertain only to one direction (left or right) or to both. This makes it possible to derive closedness conditions from linguistic observations. Thus we predict that there always exists an analysis using these categories only in closed form, even if there exist others in which they are involved in functional composition.

We will list below the most important closedness properties:

1. Categories which satisfy arguments flagged **0** are required to be closed to both sides. Examples are nominal phrases, and subordinate questions, which are never allowed to be built up out of non-contiguous parts.
2. Arguments flagged **6** as well require categories that are closed, except for Wh-extraction: one part of the lexical constituent is allowed to be in dislocation to the (sometimes far) left of the constituent. Examples are all other constituents not related to adjuncts or the verbal cluster.

Furthermore, most secondary constituents are required to be closed rightward, and in the cases where this is not required, my feeling is that in most cases this requirement could be enforced without disallowing grammatical analyses.

4.2.2 Forced disharmony

In a few places, the opposite of closedness happens. This is mainly the case with the famous crossed dependencies in Dutch verbal clusters. For more information, we refer to [Cre93, Chapter 1, Section 4]. Arguments flagged **59** and **9** only cancel categories that had none of their arguments (on one side) satisfied.

Additionally, it is required that the corresponding argument list of the primary category is still intact as well.

These provisions of course force the parser to do this reduction, whenever it has the choice between this one and another reduction which would perhaps satisfy some of the arguments of the secondary category.

4.2.3 Adjuncts

Adverbial and attributive adjuncts may adjoin at any level in the derivation tree, in as far as allowed by the grammar rules. This means that the **reduce** steps corresponding to the attachment of an adjunct should be open for backtracking; it also means that adjuncts constitute a major source of syntactical ambiguity, part or all of which should be considered *genuine* ambiguity (cf. p. 14).

In practice, the attachment freedom often leads to a number of grammatical analyses which is exponential in the sentence length, as may be seen by parsing any sentence which contains several sentential adjuncts. As many of these appear to be semantically equivalent (according to the meaning assignment made by the Delilah semantic component), I would recommend the enforcement of restrictions on the attachment possibilities. It would be particularly interesting to be able to tell by syntactic means whether two attachment possibilities will yield equivalent semantics.

As adjuncts are modeled as automorphisms, which take an argument of some type and return one of the same type, it is not possible to enforce any closedness conditions on the secondary category to match with an automorphism type. On the other hand, closer examination seems to learn that the adjunct itself is mostly a closed constituent, even if it is a subordinate clause, itself containing extraposition and all kinds of disharmony. If we could enforce this condition, we could probably exclude some spurious derivations.

An example is given by the sentence

henk sliep omdat hij gewerkt had
henk slept because he worked had
henk slept because he had been working

Here 'omdat' is categorized as $\langle\mathbf{sentadj},\mathbf{0},\mathbf{0}\rangle\backslash[\,]/[\mathbf{s_{vn}}\hat{}\mathbf{0}]$, as it still needs a subordinate clause in order to become a full-fledged sentential adjunct. However, in the current grammar it is allowed to match 'henk sliep' to form the constituent 'henk sliep omdat', which bears type $\langle\mathbf{s},\mathbf{wh},\mathbf{0}\rangle\backslash[\,]/[\mathbf{s_{vn}}\hat{}\mathbf{0}]$. In my opinion, this kind of derivations should be prevented in general; however, the consequences of such a step should be further investigated.

4.3 Improving the oracle

4.3.1 The basic recipe

In general, Delilah executes **reduce** steps whenever they are possible. By consequence, it exhibits a preference for left-branching derivation trees. Of course, this general rule is perfectly valid as an oracle. However, it is very likely to lead to a great deal of backtracking.

An important point with the Delilah grammar is that there is only one way in which a string of categories — always of length two, by the way — may be reduced to one category. Thus it is unnecessary to decide between different **reduce** steps which might be possible at the same time.

4.3.2 Anticipating on forced disharmony

Sometimes it is possible to decide against executing a **reduce** step on the basis of the remaining arguments of a category.

Suppose a category c has a right searching argument flagged **9**, say. As discussed above, this implies that it expects a secondary category which has a 'virginal' left argument list. However, it is possible that this secondary category has a left searching argument of the same category as a left searcher of c. In this case, it is prudent to postpone the satisfaction of c's left argument in favour of the satisfaction of the **9**-flagged argument on c's right hand.

The same situation holds with **1**, **4**, and **59**-flagged arguments. For all these flags, it is preferable to satisfy them with secondary categories having as much left searching arguments as possible, and hence to postpone satisfaction of the primary category's own left searching arguments.

To implement this, all leftward cancellations are subjected to a test called **check19459**, which tests whether the primary category has any unsatisfied right searching arguments flagged with one of the flags just discussed. If so, the reduction is disallowed.

4.3.3 Some specific cases

There are some specific cases when the **reduce** step should be postponed. I will not work out all those in detail here, as some have rather an *ad hoc* character; I only note that some of these involve a look-ahead of two or even three words.

One device, however, takes a more general form. It consists in checking whether the current input has the form $A/\mathbf{x}\ \mathbf{x}\ B\backslash\mathbf{x}$. An example is the categorisation of relative pronouns: $\langle\mathbf{np},\mathbf{0},\mathbf{0}\rangle\backslash[\mathbf{np\hat{\ }0}]/[\mathbf{s_{vn}\hat{\ }6}]$. When parsing a sentence like 'zie jij de man die henk ziet' (do you see the man (that) henk sees), the constituent 'zie jij' bears type $\langle\mathbf{q},\mathbf{0},\mathbf{1}\rangle\backslash[\,]/[\mathbf{np\hat{\ }0}]$, and 'de man' has type $\langle\mathbf{np},\mathbf{0},\mathbf{1}\rangle\backslash[\,]/[\,]$. However, these should not be allowed to cancel each other, as the nominal phrase is still not really complete. Consequently, **reduce** steps which have this form are not executed by Delilah.

4.4 A deterministic parser

Using all devices mentioned above, the shift-reduce parser used by Delilah is able to resolve all its conflicts in a deterministic manner. Speaking in Prolog terms, this means that cuts can be placed on all rules implementing parser moves, except for the ones that bind adjuncts (automorphisms), and that backtracking is unnecessary (but cf. page 34).

On the other hand, this might mean that not all analyses of a sentence which are licensed by the grammar, are effectively constructed by Delilah. We will explore this for the diverse efficiency measures discussed above.

It is clear by inspection that the **check19459** rule, as well as the rules mentioned as 'specific cases' do not disallow any grammatical analyses. They merely reject reduction paths which can be told in advance to be useless.

However, the general rule to prefer reduction steps cannot be seen that way. When considering the example given on p. 11, it is clear that one of the reduction possibilities will be simply ignored. As this example may be viewed as simply spurious ambiguity, though, I recommend putting a restriction on the grammar in order to establish a preference for either way of derivation in this case. I refer to Chapter 6 for more details.

This having been done, I believe that the shift-reduce parser will have overcome almost all non-lexical ambiguities implied by the Delilah lexicon and grammar (not forgetting that, as I have said above, a critical investigation of adjunct attachment possibilities is due). As stated in the previous chapter, however, the lexical ambiguities play such a disastrous role that this fast behaviour is not enough to make Delilah as a whole operate in polynomial time.

Chapter 5

An alternative to the count/shift-reduce parser

5.1 Parsing of Context-Free Languages

As indicated in the preceding chapters, the parsing algorithm currently employed by Delilah displays an exponential time complexity in the average case. Therefore, it is worthwhile to see whether an improvement of the shift-reduce parser or some other parsing technique provides us with better complexity figures. Here a main source of inspiration comes from parsing techniques for context-free languages (CF languages).

5.1.1 Why context-free?

For one part, the reason for looking at existing CF grammar parsers is that the Delilah language itself contains large CF sublanguages. For example, if a derivation includes only cancellations of arguments flagged **0**, **6**, **wh**, and **88**, the grammar reduces to a standard Ajdukiewicz-Bar-Hillel (AB) grammar, which have been proved to be CF (cf. [BGS60], or [FT96] for a summary). AB grammars are characterized as categorial grammars without functional composition, or, put differently, having function application only. On a strictly syntactical level, the Delilah grammar may even be weakly equivalent to a CF language — for example, the famous unbounded crossing dependencies in Dutch are syntactically indistinguishable from the language $\{\mathbf{np}^n\mathbf{vp}^n\}$, which is CF. However, in order to obtain a compositional semantics for our language, the actual Delilah derivation trees are incompatible with derivation trees of CF grammars.

In the second place, the only known parsing algorithm for CCGs and LIGs which exhibits polynomial time complexity, that of Vijay-Shanker and Weir (referred to in the sequel as the VW-algorithm, and developed in [VW93]), is based on a well-known parsing technique for CF grammars, namely, the chart parsing technique.

5.1.2 Chart parsers

A chart parser is an algorithm that handles sets of partial specifications of derivation trees (called *items* in more general context by Sikkel and Nijholt [SN97]). It employs two data structures: an *agenda*, containing items that will be actively used to search for new items that can be recognized, and a *chart*, storing the items that need no further attention. At each step, one item, say ξ, is taken from the agenda and put on the chart. Then the chart is searched for items that combine with ξ to give a new item ζ, which is subsequently added to the agenda. Chart parsing algorithms may differ as to the type of items they use (their data structure), and the processing order (which follows from the algorithm's control structure).

A detailed description of a chart parser is found in [Win83, Section 3.6]. This is an version of the Earley algorithm (cf. [Ear70]), which may be used for the recognition and parsing of arbitrary CF grammars. Another technique, known as the Cocke-Kasami-Younger (CKY) algorithm (cf. [You67]), requires the grammar to be in Chomsky normal form (i.e., to contain only productions of the form $A \to a$ with A nonterminal and a terminal, or $A \to A_1 A_2$ with A, A_1 and A_2 all nonterminals). Note that for CCGs this restriction is automatically satisfied.

Here we will describe the CKY algorithm (cf. [SN97, p. 66–67]). It has a different flavour from the Earley algorithm, in that there is no obvious 'agenda' structure; instead, the algorithm operates in a strict bottom-up fashion.

Suppose we have an input sentence $a_1 a_2 \ldots a_n$ for some $n \geq 1$, where the a_i are terminals of some CF grammar G. The recognition algorithm will complete an $n \times n$ array P such that a nonterminal A is stored in $P[i, d]$ if and only if $A \stackrel{*}{\Longrightarrow} a_i \ldots a_{i+d-1}$. Here $\stackrel{*}{\Longrightarrow}$ signifies the closure of the direct derivation relation; if this relation holds, we define the *span* of the entry A to be $i - (i + d)$. The initialisation phase consists in completing all basic entries — i.e., for all productions of the form $A \to a_i$ we add A to $P[i, 1]$. Then, whenever we find a production $A \to A_1 A_2$ and entries $A_1 \in P[i, d_1]$ and $A_2 \in P[i + d_1, d_2]$, we add A to $P[i, d_1 + d_2]$, provided it was not already there. This task is repeated until all combinations of two categories have been tried. Now we can accept or reject the string depending on whether $P[1, n]$ contains the grammar's distinguished nonterminal S. If so, we know that $S \stackrel{*}{\Longrightarrow} a_1 \ldots a_n$, which is the criterion for a string to be in $\mathcal{L}(G)$.

Of course, we need to define a proper control structure for the algorithm to ensure that all combinations of entries are tried, and that no combinations are tried more than once. This can be achieved, for example, by first considering all combinations leading to a derivation spanning two input words, then three, and so on, up to n. A more sophisticated structure, leading to a strictly left-to-right algorithm, is the following: at position i, we first try all combinations which would give derivations with span $(i - 1) - i$, then $(i - 2) - i$, up to $1 - i$. Then we proceed to the next input position. It is also possible to break the initialisation phase up into n steps, finding each word's categorisations as the algorithm proceeds along the input string.

5.2 Parsing CCGs and friends in polynomial time and space

The following description of the VW algorithm is adapted from [Kom97]. It should be noted that this algorithm is adapted by Vijay-Shanker and Weir (cf. [VW93]) to parsing LIGs and Tree Adjoining Grammars (TAGs). It is clear from VW's diverse publications that the relation between LIGs and CCGs is especially close. The essential additions in the parsing algorithm for LIGs is about the inclusion of unary productions and productions rewriting a nonterminal to the empty string ϵ.

Suppose we were to use a CKY-style array for testing membership of a CCG. Thus the array cells would be filled with categories of the format described in Section 2.1. Two conclusions emerge:

1. The length of a category in a cell can grow proportionally to the input size.
2. Thus the number of categories in a cell may grow exponentially to the input size.

A formal example where this happens is given in [Kom97]. Consider a lexicon where the terminal **a** is categorized as both **s/np/s** and **s/pp/s**. Then for the input **aa...a**, of length n, the top cell of the CKY-array includes 2^n combinations of categories like $\mathbf{S}\left\{ \begin{array}{c} \mathbf{/NP} \\ \mathbf{/PP} \end{array} \right\} \cdots \left\{ \begin{array}{c} \mathbf{/NP} \\ \mathbf{/PP} \end{array} \right\} \mathbf{/S}$ derived by functional composition. Thus, we have exponential worst-case performance with respect to the input size.

The idea behind the VW algorithm is to store categories as if they were some kind of linked list. Informally, a long category $y|_1 z_1|_2 \cdots |_m z_m$ is stored as y\{this portion is in some other cell [p.q] with *index* $|_{m-B} z_{m-B}$\} $|_{m-B+1} z_{m-B+1} \cdots |_m z_m$. A crucial point here is that the data represented at the current cell (y, and a bounded number of arguments) are *bounded*. At first sight, this seems a bold claim, since arguments in a CCG may represent arbitrary categories; in particular, the combinatory rules do not place restrictions on the order of the primary and secondary categories. However, it is straightforward to see that all arguments occurring in a derivation of some CCG G must be taken from some finite set $args(G)$, which is completely determined by the grammar's lexical assignment. Thus individual arguments are bounded as well.

The pair [p.q] can be represented as a n^2 matrix. Thus by setting up n^2 subcells in each CKY-table cell, we can represent a category in a bounded manner. The effectiveness of such a representation comes from the fact that CCG rule application does not depend on the entire primary category: as a matter of fact, only the most peripheral argument is needed. Note that, on the other hand, the secondary category must be available in its entirety; however, in a CCG both the length of this category and the size of individual argument categories are bounded, so that there are no problems in this respect. This all means that the proposed representation does not slow down the process of executing cancellations. When the result category exceeds a certain limit, we leave the excess portion in the original cell and set up a link to it.

One complication is that when an argument (e.g., z_m) in the above example) is cancelled, we may have to restore a portion of the category from the linked cell (by means of the 'index' part). We need to scan the linked cells and find the categories with the same index from n^2 subcells. Even though there may be multiple such categories, all of them can be restored in one of n^2 subcells associated with the result category. This case dominates the computation complexity, but it can be done in $O(n^3)$. Since it is executed once for each loop in the three outer loops of the CKY algorithm (with indices i, d_1, and d_2 as described above), the overall complexity is $O(n^6)$.

The dependence of this algorithm on the boundedness of the secondary category is crucial. If we assume that these can be no longer than MSL (for Maximum Secondary Length), it is shown by Vijay-Shanker and Weir that the so-called middle part of an entry in an array cell cannot be longer than a constant B differing by a constant from MSL (this is the same B as in the above example; cf. [VW93, p. 601]). This part is a word over the basic alphabet of the grammar (in the CCG case), so there are at most $O(|V|^B)$ possibilities for this part, which is thus bounded, whereas there are at most $O(n^2)$ possibilities per array cell for the remaining components of the entries.

Thus the constant B (and thus MSL) may depend at most logarithmically on the input length, or this technique will only provide us with exponential upper bounds on the algorithm's space complexity.

5.3 Choice of an algorithm for Delilah

As described in the previous sections, a standard CKY algorithm will do its job for categorial grammars just as well as for CG grammars. This is because cancellations occurring within a categorial grammar obviously take the form of binary rewriting rules, so that formally the categorial grammar may be said to be in Chomsky normal form. However, when applying this technique we are forewarned that its performance may deteriorate dramatically from cubic to exponential.

However, in practice the algorithm does not quite behave as bad as predicted. I ran an implementation of the standard CKY-algorithm, extended to cover double types and Wh-displacement, on a set of example sentences, and obtained all but acceptable parse times. Furthermore, these parse times appear to grow as the third power of the sentence length when considering families of sentences. Details of these experiments will be given in the section on empirical results. Interestingly, Komagata has obtained the same results for English and Japanese [Kom97].

One can argue that this is bound to happen. Viz., the possible exponential behaviour is caused by the presence of arbitrarily long argument sequences, which are words over a finite alphabet. However, to my knowledge, the only situation which gives unboundedly long lists of arguments is the final verb cluster; and these arguments may only be nominal phrases (category **np**). Thus the number of possibilities for the argument list is bounded for each value of the list length, and thus the number of categories in one array cell still does not exceed a bound which is polynomial in the sentence length.

Of course, if one would want to have a firmly proven upper bound for the worst case complexity of the parsing algorithm, one could implement the linked list device described in [VW93]. It is my opinion, however, that using this technique will result in slower performance in practice, due to the large amount of bookkeeping that ensues. Furthermore, in the large majority of cases all occurring categories are of bounded length anyway.

One may conclude that due to the strictly linear assignment of categories and the restrictive nature of the reduction rules, it is reasonable to expect a cubic performance rate from a standard CKY parser when applied to parse the Delilah grammar.

5.4 Implementation details

In the file chart.pl, I have given an implementation of a standard CKY-type parsing algorithm, along with several additions to do, for example, with the semantic module. This implementation is written in the same language as Delilah, namely SICStus Prolog, a Prolog version which allows for program compilation. It makes essential use of the lexical (lex) and semantic (unify) modules of Delilah.

The first phase of the algorithm consists in finding all possible analyses of parts of the input sentence and putting these in the parse array (the chart). This phase, which is executed by the recognize procedure, takes roughly cubic time. The control structure is such that this first phase is strictly left-to-right (cf. Section 5.1.2 above). If the output mode of the system is set to 'verbose', the recognize routine generates some diagnostic output: after each word processed, it outputs the word, the current size of the chart, the amount of time spent on trying to match categories against each other (as opposed to traversing the chart to see whether there are any categories to be matched), and the total time spent on extending the chart with this input position. For example, with the sentence 'ik heb gezegd dat Henk moest werken', *I have said that Henk had-to work*:

ik 1 (0/10ms) heb 31 (10/10ms) gezegd 80 (190/200ms) dat 85 (60/70ms) henk 86 (0/10ms) moest 105 (50/80ms) werken 132 (100/120ms)

After having filled the chart, we want to extract all valid and complete parse trees from it. There are various techniques to tackle this problem; it should be borne in mind that there may result exponentially many valid trees, so that one must decide beforehand on whether to generate them all one by one, or to find some kind of finitary representation for them (a so-called *shared forest* representation, an elaboration of which may be found in [VW90]).

In principle, it should be possible to take this approach, even when including the generation of semantic information. However, to avoid complications, I have given a straightforward top-down tree building algorithm, called maketree, which can be brought to enumerate all possible trees by backtracking upon the solutions it finds.

Alongside the tree generation process, in the current version maketree generates the semantic information that decorates these trees (in terms of Attribute Grammars). This behaviour can be switched on and off.

Also provided are two chart filtering routines. The first is a routine which eliminates all unreachable nodes (in terms of Formal Language Theory; meaning nodes which cannot be part of a valid parse tree) from the chart. It should be clear that mostly it is not possible to decide on whether a node is reachable or not before the whole chart has been filled. This is because one has to know what nodes constitute valid top nodes for derivation trees, and what their child nodes are. The reach_check procedure just described removes all nodes from the chart which are not descendants of a valid top node.

The second is a routine aiming at the removal of spurious ambiguity from the set of trees implicit in the full chart. This is done by implementing an adapted version of an algorithm originating in [HM89] and developed in [VW90], which detects and does away with spurious ambiguity at a local level. The idea is as follows: given three consecutive constituents (i.e., sequences of words which can each be assigned a single category), we find out whether they combine in two different ways to form the same result category. Of course, the only possibilities are a left-branching and a right-branching tree, both of height three and having five nodes. If so, we decide on one alternative (e.g., we could always take the right-branching alternative), and make the other one impossible. In the context of our chart parser, where all categories have attached to them a list of the ways in which they can be formed out of two smaller constituents, we only have to remove the entry corresponding to the reduction applied at the top of the rejected tree. When we apply this process in a bottom-up fashion, we have eventually freed the full chart from any spurious ambiguity — or at least, that is what Vijay-Shanker and Weir claim [VW90]. The correctness of this routine (called ambicheck) is not guaranteed and needs a critical examination.

In all this, special attention is continuously given to double types. The essential point about these is that the eventual cancellation of the left and right parts of a double type against each other is treated as a unary production of the grammar, with all the ensuing problems. However, this seemed to be the most straightforward way in which one could avoid making radical changes to the Delilah grammar and lexicon.

The user interface to this machinery is provided by three procedures: the chartparse procedure takes a string of words, calls the recognize and maketree algorithm to find a parse tree, prints it and returns it to the caller. Furthermore, it calls the reach_check and ambicheck routines, in this order, just before calling maketree, if it has been so instructed (these two calls can be switched on and off independently). If the generation of semantic information has been switched on, it passes the information gathered by maketree in this situation to the findsem routine of the semantic module, which resolves all problems to do with anaphory and synthesises a logical representation of the sentence, and prints a representation of the semantics of the sentence, as complying with this particular parse tree.

The categorize predicate, which is only usable when 'semantics' is switched off, uses the recognize and maketree procedures to examine constituentship of isolated sequences of words: it verifies that a given category can grammatically be assigned to such a sequence, or finds all categories with this property, with their associated parse trees, by backtracking.

The do predicate, finally, uses backtracking on chartparse to find all parse trees with a valid category for root node (i.e., a fully satisfied category headed **s** or **q**), and print them. Furthermore, it gathers some statistics about the running time of the algorithm and the number of unique trees which was generated.

5.5 Empirical results

In order to test the practical performance of this implementation, I made use of the set of example sentences that comes with Delilah Version V1, and adapted this set to Version V2 and to the chart parser (by removing coordination from the sentences, mainly). The results below result from running the implementation on a non-busy SiliconGraphics Indy workstation. The designations 'with' and 'without' refer to either including or excluding the possibility that nominal phrases receive also an interpretation as sentential (temporal) adjuncts (see also Section 6.3.1). This amounts to a minor modification in the categorisation lists of the definite and indefinite articles 'de', 'het' and 'een'. As may be seen, this modification causes drastic changes in running time and the amount of syntactically possible trees. Another modification with the same effects is referred to by the phrase 'with strict adjunction', and consists of a syntactic restriction on adjunct attachment possibilities (see also Section 6.3.1).

Sentence 7 (35 words):

omdat ik Henk niet met de pop wilde laten spelen werd ik door de vrouw met het mooie boek met de mooie gedichten gedwongen te proberen Henk agnes een auto te laten geven
because I Henk not with the doll wanted-to let play was I by the woman with the beautiful book with the beautiful poems forced to try Henk agnes a car to let give
because I would not let Henk play with the doll, I was forced by the woman with the beautiful book with the beautiful poems to try to make Henk give Agnes a car

with: 252 unique trees; it took 22910ms for chart filling, 29470ms in total.
without: 63 unique trees; it took 17910ms for chart filling, 21530ms in total.

Sentence 69 (38 words):

welke mannen die ik had proberen te laten werken werden door de vrouw aan die ik had gezegd dat Henk mij geen boek durfde te geven gedwongen agnes met het boek met de mooie gedichten te laten spelen
which men which I had tried to let work were by the woman to whom I had said that Henk me no book dared to give forced Agnes with the book with the beautiful poems to let play
which men which I had tried to get working were forced, by the woman to whom I had said that Henk did not dare give me a book, to let Agnes play with the book with the beautiful poems

with: 624 unique trees; it took 22970ms/38380ms.
without: 432 unique trees; it took 21230ms/31210ms.

Sentence 87 (40 words):

de man met het boek dat ik Henk de vrouw met wie ik werkte had willen laten geven is door de vrouw gedwongen te zeggen dat hij mij niet in de mooie auto die ik had gezien wilde laten werken
the man with the book which I Henk the woman with whom I was-working had want-to let give has-been by the woman forced to say that he me not in the beautiful car which I had seen wanted-to let work
the man with the book which I had wanted to let Henk give to the woman with whom I worked has been forced by the woman to say that he would not let me work in the beautiful car which I had seen

with: 96 unique trees; it took 30450ms/34370ms.
without: 24 unique trees; it took 27500ms/28480ms.

Sentence 151 (40 words; this sentence requires an addition to the current lexicon, viz., the words 'dol' (*keen, crazy*) and 'op' (*on*)):

misschien wilde de vrouw in de auto de man met het boek met de mooie gedichten proberen te dwingen te zeggen dat sommige mannen met een auto hadden willen zeggen dat een vrouw met een boek dol op goud was
maybe wanted-to the woman in the car the man with the book with the beautiful poems try to force to say that some men with a car had want-to say that a woman with a book very-fond on gold was
maybe the woman in the car wanted to force the man with the book with the beautiful poems to say that some men with a car had wanted to say that some woman with a book was fond of gold

with: 600 unique trees; it took 341760ms/370130ms.
with 'strict adjunction': 360 unique trees; it took 130250ms/145700ms.
without: 600 unique trees; it took 49390ms/74770ms.

Two families of sentences Here 'with' and 'without' make hardly any difference; these results are 'without'. In both cases, I give a polynomial function of i approximating the actual chart filling time. As may be seen, in practice The exponent is smaller than 3. Notice that Family 1 is describable by a context-free grammar, whereas Family 2 displays unbounded crossing dependencies — at least when we want to construct a semantically adequate derivation.

Family 1 ($3i$+2 words):

ik (zie dat ik)^i werk
I (see that I)^i am-working

i	Chart (ms)	$110 \cdot i^{2.5}$	Total (ms)	Trees
0	10	0	10	1
1	250	110	250	1
2	1100	622	1110	2
3	2390	1715	2420	4
4	4150	3520	4230	8
5	6570	6149	6740	16
6	9730	9700	10140	32
7	13890	14260	14800	64
8	19030	19912	21070	128
9	26060	26730	30610	256
10	34920	34785	45190	512

Family 2 ($2i$+2 words):

ik wil (Henk)^i (zien)^i
I want-to (see Henk)^i

i	Chart (ms)	$130 \cdot i^{2.75}$	Total (ms)	Trees
0	10	0	10	0
1	270	130	280	2
2	1170	875	1180	3
3	2980	2667	3000	4
4	6130	5883	6170	5
5	10940	10867	11000	6
6	17860	17952	17960	7
7	27150	27413	27280	8
8	39200	39577	39390	9
9	54290	54715	54570	10
10	72800	73104	73170	11

Based on these experiments, one can say that parsing times are at least not astronomical (of course, a faster processor combined with an implementation in a fast programming language like C could gain us a factor of magnitude in these processing times), and seem to display at most cubic behaviour with respect to the sentence length.

One could also ask how the size of the chart grows as the recognizer moves through the input sentence, and ask the same question for the time spent per input position (see the previous section for an example of how these data may be obtained). It must be said that these numbers differ dramatically per input position, depending on the degree of lexical ambiguity of the input word, and on the number of constituent boundaries lying above this position. However, for words with few lexical categories, the processing times seem to follow a gently polynomial curve.

As an example, I take the diagnostic output produced during the analysis of Sentence 7, where both verbs and articles have their full ambiguity:

omdat 1 (0/0ms) ik 2 (0/0ms) henk 3 (0/10ms) niet 4 (0/10ms) met 12 (0/10ms) de 14 (10/20ms) pop 19 (0/20ms) wilde 116 (210/220ms) laten 236 (880/920ms) spelen 682 (3370/3430ms) werd 739 (3010/3060ms) ik 763 (1400/1460ms) door 768 (50/130ms) de 770 (0/100ms) vrouw 774 (20/130ms) met 785 (20/150ms) het 792 (30/180ms) mooie 794 (10/190ms) boek 803 (20/240ms) met 820 (50/300ms) de 822 (0/280ms) mooie 824 (0/310ms) gedichten 838 (40/400ms) gedwongen 947 (610/1040ms) te 948 (30/530ms) proberen 1033 (640/1110ms) henk 1037 (50/700ms) agnes 1038 (10/750ms) een 1040 (0/810ms) auto 1043 (0/860ms) te 1045 (0/930ms) laten 1097 (50/1080ms) geven 1509 (2700/3710ms)

The discussed behaviour of the verbs is apparent in this example: both the growth in the number of chart entries and the processing time are in no proportion to these data for the other words.

Chapter 6

Conclusions and further research

6.1 Conclusions

Several conclusions can be drawn from the preceding chapters. In a very short formulation, these are the following.

1. The grammar currently used by Delilah can be formulated as a LIG, by
   - Reverting the category to the standard form
   - Replacing the rule for flags **9** and **59** by the one suggested on page 13
   - Applying the standard conversion from CCGs to LIGs, given in [VW93]

   However, the use of special arguments (i.c. the accommodation of Wh-gaps) still is an essential addition to the LIG-formalism. I do not know what family of languages is derivable when this feature would be used in full strength, instead of the restricted fashion used at present. However, the present formalism may be said to generate a LIL with the additional application of Wh-movement in the sense of transformational grammars.
2. The parser currently used by Delilah, which consists of a count stage and a shift/reduce parser, probably uses exponential time and space in the average case.
3. A chart parser following the CKY-model with no additional modifications except for Wh-movement achieves worst case cubic time complexity in practice, under the provisions sketched on page 14 for exponential genuine ambiguity. Theoretically, a worst case time complexity of $O(n^6)$ can be obtained by applying the techniques described in [VW93] and [VW90].

6.2 Suggestions for improving the grammar

In this section, I will present the anomalies and problems I have encountered using the Delilah grammar. Being no linguist, I just formulate proposals to overcome these difficulties, which could contribute to a sharper formulation of the grammar. The modification of the rule for **9** and **59** flags has been discussed earlier, and will not be repeated here. For a proposal concerning adjuncts cancelling leftwards, see below.

6.2.1 Effects of adopting one single argument list

As said above, we must convert Delilah's lexicon to the standard category format if we want to describe the grammar as a LIG — viz., neither CCG nor LIG support the use of separate lists of left and right searching arguments (which would convert to left and right stacks for LIG). However, what are the effects of executing this conversion on the language accepted by the grammar, and does there at all exist such a conversion?

The answer to the latter question has been given in Section 2.4.1. With the exception of some very specific examples, most categorisations of lexical items are used in only one way, which corresponds to just one standard form. This also means that if we only give this standard form in the lexicon, the language accepted by the grammar will not change. With the very specific examples, we will have to decide on a normal form for derivations (see also below), like preferring right-branching trees over left-branching ones. My opinion is that this can be done.

It should be noted that the effect of the **9** and **59** argument flags may be achieved without further means just by making the arguments thus flagged peripheral (i.e., to be met first, before satisfying the other arguments). When a categorial grammar does not include type raising, like the Delilah grammar, and CCG in general, one can steer derivations to a large extent by deciding on the relative order of the left and right searching arguments of categories.

A technical complication arises whenever we want to verify that a secondary category in a cancellation has no left searching arguments (and may still have right searching ones). Whereas in the present situation this is achieved by just checking whether the left list is empty, for categories in standard form we will have to check each argument. However, this is a bounded task, since secondary categories have a bounded number of arguments.

6.2.2 q/s ambiguity and related questions

When parsing a Dutch main clause without topic, an ambiguity arises, as this sentence is assigned both **q** and **s** categories. This is because there is no way in the Delilah grammar to enforce Wh-movement. However, except in yes/no-questions, it is Wh-movement is necessary in order to obtain a grammatical sentence. Put another way, one could say that the [SPEC,CP]-position must be filled except with yes/no-questions. The grammar implements this restriction for other kinds of questions by means of the double types for Wh-elements and the handling of Wh-gaps. However, yes/no-questions can still be interpreted as sentences (category **s**) which have not yet undergone Wh-movement.

Thus, in order to accept a sentence, it must be verified that the entire sentence's category has the form $\langle \mathbf{s}, \mathbf{wh}, \mathbf{f} \rangle \backslash [\,] \backslash [\,] / [\,]$, the **wh**-flag indicating Wh-movement. A reason for not yet implementing this check is that the [SPEC,CP]-position may as well be filled by a sentential adjunct, as in 'misschien komt hij' (*maybe he will come*). In the current grammar, this is not considered as Wh-movement.

My suggestion for mending this situation would be to grant the **wh**-flag to a (fully satisfied) **s**-headed category upon consumption of a sentential adjunct to its left.

A related observation is that the [SPEC,CP]-position may be filled only once. A sentence like 'misschien hij komt' is not Dutch and should be ungrammatical, as well as a sentence like 'misschien met de trein komt hij'. At present, these sentences are allowed by Delilah. This is incorrect; therefore, I propose that the rule for right-searching **8**-flagged arguments be changed to forbid cancellation against a category with left head flag **wh**.

Finally, this latter problem arises in subordinate clauses as well, although grammaticality judgments vary here along strange lines. Compare e.g.

ik denk dat misschien hij komt	ik denk dat misschien de directeur weggaat
I think that maybe he will-come	*I think that maybe the CEO will-leave*

Here the sentence on the left is highly dubious, and can be admitted only if the denotation of 'hij' has not previously entered the discourse, and 'hij' receives heavy emphasis. The one on the right, on the other hand, is acceptable, whereas emphasis may lie on 'misschien', 'directeur' as well as 'weggaat'.

In general, the acceptability of subordinate clauses which are preceded by one or more sentential adjuncts seems to depend on whether new information is introduced in the sequel of the clause.

6.2.3 Topicalization of np-complements of prepositional adjuncts

When parsing the sentence 'de man heeft met de auto gewerkt' (*the man has been-working with the car*), Delilah gives, apart from the desired ones, a very curious analysis in which 'de man' is the **np**-complement of the prepositional adjunct 'met de man', which has been topicalized. Thus the sentence would run 'de man heeft met t de auto gewerkt' (*it is the man the car worked with*). This is not Dutch and should be ungrammatical, however. The corresponding question 'heeft met de man de auto gewerkt', apart from the question whether cars can work, sounds at least poetic and could only mean *has the car been working as well as the man?*, where 'met' means *together with* or *as well as*.

In Dutch, there are the following word order possibilities for this type of sentence (*it is this house the children have been-walking in*) and the corresponding questions (*have the children been-walking in this house?*):

hebben in dit huis de kinderen gelopen?
hebben de kinderen in dit huis gelopen?
hebben de kinderen gelopen in dit huis?
*dit huis hebben in t de kinderen gelopen
dit huis hebben de kinderen in t gelopen
*dit huis hebben de kinderen gelopen in t
in dit huis hebben t? de kinderen t? gelopen t?

Both ungrammatical options are currently allowed by Delilah. The first option can be excluded by the restriction on adjunction discussed below. The second option, however, is difficult to exclude. An option could be to require first only right branching derivations, and then to forbid attachment of nonclosed prepositional adjuncts to past participles.

6.2.4 Miscellaneous

Right searching 6 In the documentation of the grammar (in parsegr.pl/chgram.pl), it says that Secondary-ActiveList may be required to be empty. I would propose to do so. Furthermore, for the rule to be correct, the left searching argument lists must be appended using the special whappend predicate, in order to take care of the special arguments. An example where this is needed, is given by

```
1-8+q\0~[]/1~[]  [wie,dacht,jij,dat,henk,zei,dat,werkt]
   1-8+q\0~[]/0~[s^1]*s\wh~[]/1~[]  [wie,dacht,jij,dat,henk,zei,dat,werkt]
      1-1+q\0~[]/0~[s^1]*np\0~[]/0~[]  [wie]
      2-8+s\0~[np^wh]/1~[]  [dacht,jij,dat,henk,zei,dat,werkt]
         2-3+s\0~[]/1~[s_sub^6]  [dacht,jij]
            2-2+s\0~[]/0~[np^0,s_sub^6]  [dacht]
            3-3+np\0~[]/0~[]  [jij]
         4-8+s_sub\0~[np^wh]/1~[]  [dat,henk,zei,dat,werkt]
            4-4+s_sub\0~[]/0~[s_vn^6]  [dat]
            5-8+s_vn\1~[np^wh]/1~[]  [henk,zei,dat,werkt]
               5-5+np\0~[]/0~[]  [henk]
               6-8+s_vn\0~[np^0,np^wh]/1~[]  [zei,dat,werkt]
                  6-6+s_vn\0~[np^0]/0~[s_sub^6]  [zei]
                  7-8+s_sub\0~[np^wh]/1~[]  [dat,werkt]
                     7-7+s_sub\0~[]/0~[s_vn^6]  [dat]
                     8-8+s_vn\0~[np^wh]/0~[]  [werkt]
```

A curious observation is that this rule appears to give problems with the shift/reduce parser. Here the strategy followed to obtain a deterministic parser appears to be incorrect: with the sentence 'ik denk dat henk wil werken', Delilah finds only a very peculiar analysis, making ample use of a nonempty SecondaryActiveList with cancelling **6**-flagged arguments:

```
1-6+s\wh~[]/1~[] [ik,denk,dat,henk,wil,werken]
   1-1+np\0~[]/0~[] [ik]
   2-6+s\0~[np^wh]/1~[] [denk,dat,henk,wil,werken]
      2-5+s\0~[np^wh]/1~[vp^9] [denk,dat,henk,wil]
         2-3+s\0~[np^wh]/1~[s_vn^6] [denk,dat]
            2-2+s\0~[np^wh]/0~[s_sub^6] [denk]
            3-3+s_sub\0~[]/0~[s_vn^6] [dat]
         4-5+s_vn\1~[]/0~[vp^9] [henk,wil]
            4-4+np\0~[]/0~[] [henk]
            5-5+s_vn\0~[np^0]/0~[vp^9] [wil]
      6-6+vp\0~[]/0~[] [werken]
```

However, under the present rule there are 5 possible analyses, which differ in the balance between left and right branching. If we enforce the restriction discussed above, the only remaining analysis is the following purely right branching tree, which is also admitted by the present rule:

```
1-6+s\wh~[]/1~[]  [ik,denk,dat,henk,wil,werken]
   1-1+np\0~[]/0~[]  [ik]
   2-6+s\0~[np^wh]/1~[]  [denk,dat,henk,wil,werken]
      2-2+s\0~[np^wh]/0~[s_sub^6]  [denk]
      3-6+s_sub\0~[]/1~[]  [dat,henk,wil,werken]
         3-3+s_sub\0~[]/0~[s_vn^6]  [dat]
         4-6+s_vn\1~[]/1~[]  [henk,wil,werken]
            4-4+np\0~[]/0~[]  [henk]
            5-6+s_vn\0~[np^0]/1~[]  [wil,werken]
               5-5+s_vn\0~[np^0]/0~[vp^9]  [wil]
               6-6+vp\0~[]/0~[]  [werken]
```

Right searching 8 Apart from the considerations discussed in connection with the **q/s** ambiguity, the left argument lists should be appended using whappend instead of the ordinary append predicate.

Left searching 7 In the documentation it says that SecondaryPassiveList must be empty. However, this should be corrected to SecondaryActiveList, and subsequently enforced. This is necessary as crossing dependencies are forbidden when using green word order (which is the default word order in German). In order to ensure nested dependencies, SecondaryActiveList must be empty.

However, as long extraction from this position is possible, we must refine this condition to requiring that SecondaryActiveList may contain only special arguments. An example is

```
1-7+q\0~[]/1~[]  [wie,denk,je,dat,hij,gezien,heeft]
   1-7+q\0~[]/0~[s^1]*s\wh~[]/1~[]  [wie,denk,je,dat,hij,gezien,heeft]
      1-1+q\0~[]/0~[s^1]*np\0~[]/0~[]  [wie]
      2-7+s\0~[np^wh]/1~[]  [denk,je,dat,hij,gezien,heeft]
         2-3+s\0~[]/1~[s_sub^6]  [denk,je]
            2-2+s\0~[]/0~[np^0,s_sub^6]  [denk]
            3-3+np\0~[]/0~[]  [je]
         4-7+s_sub\0~[np^wh]/1~[]  [dat,hij,gezien,heeft]
            4-4+s_sub\0~[]/0~[s_vn^6]  [dat]
            5-7+s_vn\1~[np^wh]/0~[]  [hij,gezien,heeft]
               5-5+np\0~[]/0~[]  [hij]
               6-7+s_vn\1~[np^0,np^wh]/0~[]  [gezien,heeft]
                  6-6+vp_d\0~[np^wh]/0~[]  [gezien]
                  7-7+s_vn\0~[vp_d^7,np^0]/0~[]  [heeft]
```

6.3 Recommendations for further research

6.3.1 Adjunct attachment

Apart from attachment ambiguities, adjuncts also contribute to lexical ambiguity, as in Dutch many nominal phrases are valid as sentential (temporal) adjuncts. Two examples are:

hij heeft jaren gewerkt	de dag dat hij terugkwam, moest ik werken
he has been working for years	*the day that he returned, I had to work*

This shows that both simple and compound NPs can act as sentential adjuncts. Early disambiguation could rely on the noun involved denoting a time span; however, it should be asked whether and how semantic considerations of this kind can be let into the syntactic parsing process.

I would like to suggest extending the chart parser with a technique for postponing adjunct attachment. One way to do this would be the formation of 'gapped' partial analyses: if one has the constituents 5-6+$\langle \mathbf{np}, \mathbf{0}, \mathbf{1}\rangle\backslash[\,]/[\,]$, 7-7+$\langle \mathbf{sentadj}, \mathbf{0}, \mathbf{0}\rangle\backslash[\,]/[\,]$, and 8-8+$\langle \mathbf{s}, \mathbf{0}, \mathbf{0}\rangle\backslash[\mathbf{np\hat{\ }0}, \mathbf{np\hat{\ }wh}]/[\,]$, one could derive 5-[7-7]-8+$\langle \mathbf{s}, \mathbf{1}, \mathbf{0}\rangle\backslash[\mathbf{np\hat{\ }wh}]/[\,]$, or something to this effect, indicating that this derivation contains nonattached adjuncts. Afterwards, in the tree collecting phase, one could successively try all attachment possibilities for the adjuncts, discarding the ones that are semantically equivalent to another one.

If one would succeed in finding canonical attachment positions for adjuncts, one could apply the technique described in [VW90] for discarding spuriously ambiguous derivations on a local level. I have given an implementation of this technique in the file chart.pl. Application of this technique results in a substantial reduction of the number of generated trees (30 trees vs. 252 trees for sentence 7 from the test set. However, I feel that this filter's behaviour is not entirely consistent, and too strict as well, discarding analyses that have a distinct semantics from the remaining analyses.

A possible normal form for adjunct attachment positions could be defined the one with widest syntactical scope (among sets of semantically equivalent attachments). Furthermore, I think it is wise to demand that adjunct constituents be closed except for Wh-extraction. This could be enforced by modifying the reduction rules for flag **8**.

I do not think that the goals given above can be achieved by purely syntactical means. As an example of these, one could add restrictions on the unary productions defining adjunct rewriting possibilities, or on the **ˆ8** cancelling rules. A possibility I have tried is to require an empty Secondary Passive List (for this term, see the comments in parsegr.pl. This would rule out the fallacious analyses of 'henk heeft waarschijnlijk agnes gezien' (*henk has probably seen agnes*) which assign the subject role to 'agnes' and the object role to 'henk'. Note that these analyses are valid when the adjunct is removed. However, I am under the impression that this measure discards some valid analyses as well, as in the sentence

wilde de vrouw in de auto de man proberen te dwingen te werken
wanted-to the woman in the car the man try to force to work
did the woman (in the car) try to force the man (in the car) to work (in the car)

where one of the three attachment possibilities is lost. In any case, this deserves further investigation.

In order to convince onself that the current dealing of sentential adjuncts still needs a finishing touch, one could consider the sentence 'henk denkt dat ik de fiets wil proberen groen te verven' (*henk thinks that I want to paint the bicycle green*). The only position where sentential adjuncts (like 'misschien', *maybe*) are not allowed by the Delilah grammar is between 'de' and 'fiets'. Even a split infinitive ('te misschien verven') is admitted (cf. [Cre93, p. 51]).

6.3.2 Parser-semantics interface

At present, the interface between the chart parser and the semantic module is not optimal. This is because the top predicates of the semantic module are based on a shift/reduce parsing order. Either one should adapt the maketree procedure to produce the set of applied reductions in the correct order, or one should reformulate the semantic module to follow a chart parsing model. This latter option would even enable Delilah to do syntactic parsing and semantic attribute evaluation at the same time. (At present, this is also possible, but not implemented however, with the shift/reduce parser and semantic mechanism.)

I would like to remark that remoulding the semantic module after a chart parsing model will probably improve performance in the tree building phase. The reason for this is that many trees which are allowable syntactically are eventually rejected because they cannot be assigned a sensible semantics. Drawing an analogy with the standard top-down parsing process, to which the current tree building algorithm is comparable, one is likely to form the same inviable subtrees time and again. This might be avoided by proceeding from valid partial analyses only and building valid overall analyses from these. However, a prerequisite for this method to be reasonably fast is that first all unreachable nodes which result from the syntactical analysis are removed, so that no useless unifications will be done. In the file chart.pl, I have provided an implementation of this elimination process. I have found that in practice, over 90% of all nodes in the chart are unreachable from a valid top node and play no role in any valid derivation tree.

A second problem is that double types (used as categorisations of Wh-elements like 'waar' (*where*) and 'wie' (*who(m)*)) are not correctly transferred to the semantic module. This is due to the chart parser using a different processing of double types from the shift/reduce parser.

In order to be able to continue using the semantic module in practice, I have provided an *ad hoc* interface which translates individual reductions into the format used as input by the semantic module, bypassing the semantic module's top predicates, and calling only the unify_one_reduction procedure. These calls are built into the maketree procedure. Thus, if the generation of semantic information has been switched on, this routine only returns trees which are valid semantically as well as syntactically. In order to provide a correct treatment of double types, one had to indulge into a considerable amount of index juggling. It works, however.

6.3.3 Parsing of coordination

The chart parser does not yet incorporate the techniques used by Delilah to deal with coordination. I do not know whether these can be easily adapted to conform to a chart parsing model, although my feeling is optimistic. However, possibly their orientation on whole sentences needs to shift to some extent to an orientation on sentence parts.

6.3.4 Other parser possibilities

In Chapter 5, I have shown that a chart parser obtains very good results when applied to the Delilah grammar. However, the chart parsing model is not the only efficient parsing model for CF languages. In particular, it is possible that (an extension of) Tomita's Generalized LR parser will give equally good results, while retaining the flavour of a Pushdown Automaton present in the current shift/reduce parser. This algorithm was introduced in [Tom85]. Furthermore, Tomita has given an algorithm for parsing of CCGs [Tom88]. Vijay-Shanker and Weir note that this algorithm suffers from the same problems as the standard CKY-algorithm [VW93, p. 595]. However, in practice this is not necessarily a problem, as we have shown above.

The advantage of a PDA-like algorithm would be that the task of providing interfaces to the semantic and coordination modules would be made easier. However, it is possible that the techniques used in Tomita's algorithm for conquering the grammar's lexical ambiguity make this interface a nontrivial problem again, like it is in Lang's proposal for example [Lan74].

As Sikkel and Nijholt note [SN97], a chart parser is the canonical implementation of a parsing schema. This may be taken to mean that any known parsing algorithm for CF grammars may be implemented as some form of chart parser. Sikkel and Nijholt demonstrate this by giving a parsing schema corresponding to (generalizations of) algorithms based on PDAs, like the shift-reduce parser used presently by Delilah. An example implementation may be found in [Lan74]. This paper is about efficient processing of nondeterministic pushdown automata; nevertheless, it "considers exclusively Earley-type techniques".

Bibliography

[BGS60] Y. Bar-Hillel, C. Gaifman, and E. Shamir, "On categorial and phrase structure grammars", in: *Bull. Res. Council Israel*, Vol. 9F [1960], pp. 1–16.

[BH89] Hans Bennis and Teun Hoekstra, *Generatieve Grammatica*, Foris, 1989.

[BMvB88] Wojciech Buszkowski, Witold Marciszewski, Johan van Benthem (eds.), *Categorial grammar*, Series *Linguistic & literary studies in Eastern Europe (LLSEE)*, Vol. 25, Benjamins, 1988.

[Bar53] Y. Bar-Hillel, "A Quasi-Arithmetical Notation for Syntactic Description", in: *Language*, Vol. 29 [1953], pp. 47–58. Reprinted in: Y. Bar-Hillel, *Language and Information*, Addison-Wesley, 1964, pp. 61–74.

[Ben86] Johan van Benthem, *Essays in Logical Semantics*, Reidel, Dordrecht, 1986, Section 7.4.

[Cre93] Crit Cremers, *On Parsing Coordination Categorially*, HIL Diss. 5, Diss., Leiden 1993.

[DJL88] Pierre Deransart, Martin Jourdan, and Bernard Lorho, *Attribute grammars : definitions, systems, and bibliography*, LNCS Series, Vol. 323, Springer, 1988.

[Ear70] Jay Earley, "An efficient context-free parsing algorithm", in: *Communications of the ACM*, Vol. 13 [1970], No. 2, pp. 94–102.

[FT96] Alain Finkel and Isabelle Tellier, "A polynomial algorithm for the membership problem with categorial grammars", in: *Theoretical Computer Science*, Vol. 164 [1996], pp. 207–221.

[HM89] Mark Hepple and Glyn Morrill, "Parsing and derivational equivalence", in: *Proceedings of the 4th Conference of the European Chapter of the ACL* (EACL'89).

[Hep92] Mark Hepple, "Chart Parsing Lambek Grammars: Modal Extensions and Incrementality", in: *Proceedings of the 14th Int. Conf. on Comp. Ling.* (COLING'92), Nantes, France, 1992.

[Hijwww] A web-based interface to Delilah is available at http://fonetiek-6.LeidenUniv.nl/hijzlndr/delilah.html.

[JVW91] Aravind K. Joshi, K. Vijay-Shanker, and David J. Weir, "The Convergence of Mildly Context-Sensitive Grammar Formalisms", in: Peter Sells, Stuart M. Shieber, and Thomas Wasow (eds.), *Foundational Issues in Natural Language Processing*, MIT Press, 1991, pp. 31–82.

[Kom97] Nobo Komagata, "Efficient Parsing for CCGs With Generalized Type-Raised Categories", in: *International Workshop on Parsing Technologies '97*, pp. 135–146.

[Lam58] Joachim Lambek, "The Mathematics of Sentence Structure", in: *American Mathematical Monthly*, Vol. 65 [1958], pp. 154–169. Reprinted in [BMvB88].

[Lan74] Bernard Lang, "Deterministic Techniques for Efficient Non-Deterministic Parsers", in: J. Loeckx (ed.), *2nd Colloquium on Automata, Languages and Programming*, LNCS Series, Vol. 14, Springer, 1974.

[Per85] Fernando C.N. Pereira, "A new characterization of attachment preferences", in: David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky (eds.), *Natural language parsing: Psychological, computational, and theoretical perspectives*, Cambridge Univ. Press, 1985, pp. 307–319.

[SN97] K. Sikkel and A. Nijholt, "Parsing of Context-Free Languages", in: G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, Vol. 2: *Linear Modeling: Background and Application*, Springer, 1997, pp. 61–100.

[Ste85] M. Steedman, "Dependency and coordination in the grammar of Dutch and English", in: *Language*, Vol. 61 [1985], pp. 523–568.

[Ste86] M. Steedman, "Combinators and grammars.", in: R. Oehrle, E. Bach and D. Wheeler (eds.), *Categorial Grammars and Natural Language Structures*, Foris, 1986, pp. 417–442.

[Ste00] Mark Steedman, *The Syntactic Process*, MIT, to appear in 2000. This book develops CCGs as a linguistic theory at length. A draft version is available from http://www.cogsci.ed.ac.uk/~steedman/papers.html.

[Tom85] M. Tomita, *Efficient Parsing for Natural Languages*, Kluwer, 1985.

[Tom88] M. Tomita, "Graph-structured stack and natural language parsing", in: *Proceedings of the 26th Meeting of the Assoc. for Comp. Ling.* (ACL'88), pp. 248–257.

[VW90] K. Vijay-Shanker and David J. Weir, "Polynomial time parsing of Combinatory Categorial Grammars", in: *Proceedings of the 28th Meeting of the Assoc. for Comp. Ling.* (ACL'90), pp. 1–8.

[VW93] K. Vijay-Shanker and David J. Weir, "Parsing Some Constrained Grammar Formalisms", in: *Computational Linguistics*, Vol. 19 [1993], No. 4, pp. 591–636.

[VW94] K. Vijay-Shanker and David J. Weir, "The Equivalence of Four Extensions of Context-Free Grammars", in: *Math. Systems Theory*, Vol. 27 [1994], pp. 511–546.

[Win83] Terry Winograd, *Language as a Cognitive Process*, Vol. 1: *Syntax*, Addison-Wesley, 1983.

[You67] Daniel H. Younger, "Recognition and parsing of context-free languages in time n^3", in: *Information and Control*, Vol. 10 [1967], pp. 198–208.

Notice

 Correspondence should be addressed to the Department of Linguistics, P.O. Box 9515, 2300 RA Leiden, The Netherlands.

The SICStus Prolog package is available from the Swedish Institute for Computer Science. More information may be obtained via WWW at http://www.sics.se/isl/sicstus.html.

This report was typeset by LaTeX, using the report document class with the 10pt option, and the additional packages latexsym and url.