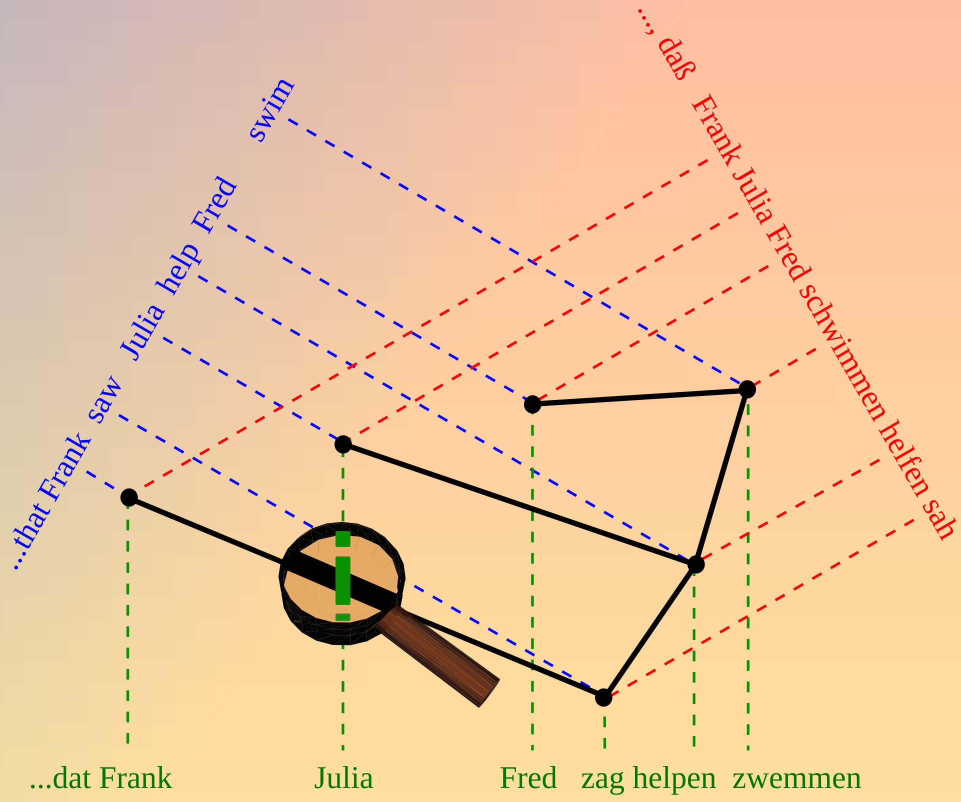
Surface without Structure

word order and tractability issues in natural language analysis


...that Frank saw Julia help Fred swim
..., daß Frank Julia Fred schwimmen helfen sah
...dat Frank
Julia
Fred
zag helpen
zwemmen

Annius Groenink

Surface without Structure

Word order and tractability issues in natural language analysis

Oppervlakte zonder Structuur

Woordvolgorde en computationele uitvoerbaarheid in natuurlijke- taalanalyse

(met een samenvatting in het Nederlands)

Proefschrift ter verkrijging van de graad van doctor
aan de Universiteit Utrecht,
op gezag van de Rector Magnificus, Prof. dr H.O. Voorma
ingevolge het besluit van het College van Decanen
in het openbaar te verdedigen
op vrijdag 7 november 1997
des middags te 16.15 uur

door

Annius Victor Groenink

geboren op 29 december 1971 te Twello (Voorst)

Promotoren: Prof. dr D.J.N. van Eijck
Onderzoeksinstituut voor Taal en Spraak
Universiteit Utrecht
Centrum voor Wiskunde en Informatica

Prof. dr W.C. Rounds
Deptartment of Electrical Engineering and Computer Science
University of Michigan

The research for this thesis was funded by the Stichting Informatica-Onderzoek Nederland (**SION**) of the Dutch national foundation for academic research **NWO**, under project no. 612-317-420: *incremental parser generation and context-sensitive disambiguation: a multidisciplinary perspective*, and carried out at the Centre for Mathematics and Computer Science (**CWI**) in Amsterdam.

Printed and bound at **CWI**, Amsterdam; cover: *focus on non-projectivity in Dutch*.

ISBN 90-73446-70-8

Contents

Prologue

This book is a report on four years of research for a PhD degree that I did in a project called INCREMENTAL PARSER GENERATION AND CONTEXT-SENSITIVE DISAMBIGUATION: A MULTIDISCIPLINARY PERSPECTIVE. The aim of this project is to build a bridge between knowledge about and techniques for language description in Software Engineering and Computational Linguistics. The train of thought to be explored was that in Computer Science, as well as in Linguistics, the languages in consideration have elements of non-context-freeness, but the approaches to dealing with such properties are rather different. Linguistics could benefit from the knowledge of efficient parsing techniques, such as the 'incremental' **GLR** parsing from [Rek92]. Some elements of non-context-freeness in computer languages are the offside rule in Miranda [Tur90] and freely definable operator precedences such as in **Prolog**. In Computer Science it is even common practice to work with systems that are not capable of describing any context-free language: in traditional **LALR**-driven analysis using tools like **YACC**, operator precedences are treated in a rather old-fashioned, *ad hoc* manner; it was thought that here Software Engineering could benefit from linguistic experience; a key concept in unifying the trans-context-free methods in the two fields was that of a DISAMBIGUATION phase defined on PARSE FORESTS that are the output of a context-free analysis. The idea then was to make a thorough assessment of the question precisely which types and which amounts of context-sensitivity are necessary to handle the non-context-free elements of both computer languages and natural languages, and to develop, based on this experience, a syntactic basis which would serve both as an improvement of the software engineering environment **ASF+SDF** [Kli93] and as a basis for linguistic analysis.

The project has been carried out in two lines of research; one concentrating on methods for syntax definition in a Software Engineering context, carried out by EELCO VISSER, reported on in his PhD thesis [Vis97], and the other, reported in this book, concentrating on problems of a linguistic nature.

Whereas in the Software Engineering part, a shift from the syntactical phase to a disambiguation phase turned out to be desirable, the linguistic counterpart showed a rather opposite movement; instead of analyzing the notion of 'disambiguation', my work can be characterized as eliminating any form of disambiguation that would be required by a system that is based on a form of CONTEXT-FREE BACKBONE, by increasing the power of the syntactic basis from context-free grammar to structurally more powerful mechanisms, eliminating among other things the problem of grammars not being OFF-LINE PARSABLE (section **7.3**). However, there is also an important analogy in the methods developed in Visser's perspective and mine; both the shift in the **ASF+SDF** group in several phases from an initially typically **LALR**-driven analysis to a scannerless context-free analysis conceptually followed by context-sensitive filtering operations defining such things as lexical keywords and precedence definitions, and the increase of the structural capacities of underlying linguistic formalisms studied in this thesis are aimed at increased RE-USABILITY or MODULARITY. In Software Engineering, one thus avoids having to rewrite considerable parts of a grammar when adapting an engineering tool to a slightly different dialect of a computer language (say **Cobol**); in Linguistics, one has the advantage that languages with similar deep structures, but essentially different surface orders, such as Dutch (SOV) *vs.* English (SVO) can optimally share a semantic phase based on simple λ expressions.[1] Several choices made in this thesis are motivated by the wish to build natural language systems relying on equational specification methods for post-syntactic processing, and altough this is probably not visible at the surface, the frequent and pleasant discussions I had with Eelco have certainly had an encouraging influence on my work.

*

It is good for the understanding of this text to let it begin with a brief chronological explanation of how my research has progressed; not least because in order to present the material I have gathered over the years in a continuous story-line, it turned out to be hardly necessary to change the order of the subjects as they presented themselves.

This **Prologue** then serves to acknowledge, in a concrete manner, the interest, help and guidance I received from various people, as well as to introduce the reader briefly to the points of departure of this work, before s/he proceeds to the introductory chapter **1**, which more or less necessarily starts with a discouraging amount of formal background material before getting to the point where these points of departure have been properly illustrated and an overview of the three parts of this book can be given in some detail.

*

In March 1994, my supervisor JAN VAN EIJCK arranged for me to go on a visit to STEPHEN PULMAN at **SRI** Cambridge, for a short project to get acquainted with work on natural language grammars. In Cambridge, I modified a unification grammar that Pulman wrote for the pan-European **FraCaS** project, to accept Dutch. Later I generalized this to a single grammar that accepts both English and Dutch.

This was a first point where the structural similarity of Indo-European languages, and hence the issue of re-usability and modularity showed its importance. An example was that I had to generalize the system that described auxiliary inversion to the more general Dutch case of verb second, which required a step of considerable orthogonalization, or in other words, required a number of *ad hoc* descriptions to be worked out in a more general manner.

A question that puzzled me henceforth was why movement of noun phrases was done through gap-threading, while verb movement was modelled with feature-driven selection of different context-free rules. One reason for this was easily identified—the simplicity of English allows for such a solution in the case of auxiliary inversion. When,

[1] Being aimed primarily at bridging gaps in the interface of various grammar systems to work in semantics by unifying syntactic structures underlying different languages, this thesis, with a small exception in section **10.4**, does *not* go into issues in semantics itself.

motivated by this worry, I tried to implement verb movement using gap threading, it became clear that such a move lifted not only the computational difficulty, but also the intuitive complexity of grammars to problematic levels.

An important aspect of feature grammars that also struck me, is that one realm of feature structures is used both for moving around structural representations, and to describe simple finite attributes such as agreement, tense and selectional properties, but that there is a dissimilarity in the way in which the feature system is used for these two different purposes: namely that between unboundedly long list constructions, and that of finite records of finite-range attributes.

Finally, working on the **FraCaS** grammar gave an insight into feature-style grammar writing, in particular gap-threading methods. I picked up the notion of *unprincipled feature hacking*, in part because by the time I didn't understand the finesse of the principles underlying the rules that Pulman's system silently obeyed—a danger of *rule based* grammars falling into the wrong hands (in casu mine).

Briefly summarized, the project provided the following four hints, the last of which is a consequence I did not draw immediately at the time:

A There is more structural similarity across (Indo-European) languages than many current approaches to syntax exploit.

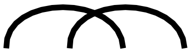
B For reasons of orthogonality, different forms of movement should not be described by different formal mechanisms. Because of its steep grammar complexity curve, slash-threading seems to be a cumbersome candidate for a universal mechanism underlying movement.

C Surface order should be separated from deep-structural (attribute) description; after doing so, every deep structure node will need to bear only boundedly much feature information.

D Rule-based grammars are not only lack explanatory capacity; they are also a burden in practice.

The hints **A**–**C**, together with the aim to find a mechanism that would enable one to write grammars (especially for Dutch) with an output interface that was indistinguishable from the output of a context-free parser, led, in the second half of 1994, to an early formulation of the LITERAL MOVEMENT GRAMMARS (**LMG**) that play a major rôle in this thesis. At this point I wish to thank JASPER KAMPERMAN for his enthusiastic reaction to a first outline of literal movement, without which I may not have guessed that it was worth further investigation.

The **LMG** framework claimed that the bulk of *movement* phenomena, an important example of which is the crossed dependency structure (1) in Dutch,



(1) ...dat Frank Julia koffie zag drinken
that coffee saw drink–INF

...that Frank saw Julia drink coffee

can be described by merely moving around *terminal strings*, rather than moving around, or sharing, complete *phrase structure trees*. The need to use unboundedly deep feature structures that move around complete structural analyses then disappears and is taken over by the structural backbone that is generated by a grammar close in style to a **CFG**.

A short example **LMG** fragment in the style of [Gro95b] is (2), which describes the behaviour of the RAISING VERBS responsible for the crossed dependencies in Dutch.

$$\begin{array}{lll} \mathrm{S} & \rightarrow & \ldots\texttt{dat}\ \mathrm{NP}\ m\ \mathrm{VP}(m) \\ \mathrm{VP}(nm) & \rightarrow & \mathrm{VR}\ (\mathrm{NP}/n)\ \mathrm{VP}(m) \\ \mathrm{VP}(n) & \rightarrow & \mathrm{VT}\ (\mathrm{NP}/n) \end{array} \tag{2}$$

Such an **LMG** behaves very much like a **CFG**, but a nonterminal can have ARGUMENTS that are instantiated in an **LMG** derivation as terminal strings, hence the name *literal movement*. In this grammar, the argument of the VP is the sequence of all object NPs. Hence, the VP constituent only yields a sequence of verbs, and postpones the realization of the objects until it is used in the S production, producing such sentences as (3).

$$\ldots \text{dat Frank } \overbrace{\text{Julia koffie}}^{m}\ [_{\mathrm{VP}(m)} \text{ zag drinken }] \tag{3}$$

SLASH ITEMS like (NP/n) in the example fragment generate the empty string, but instantiate variables that construct the terminal strings in argument positions. The best way to illustrate the slash items at this stage is to show the derivation of the VP in the example (3).

$$\frac{\mathrm{VR} \vdash \texttt{zag} \quad \mathrm{NP} \vdash \texttt{Julia} \quad \dfrac{\mathrm{VT} \vdash \texttt{drinken} \quad \mathrm{NP} \vdash \texttt{koffie}}{\mathrm{VP}(\texttt{koffie}) \vdash \texttt{drinken}}}{\mathrm{VP}(\texttt{Julia koffie}) \vdash \texttt{zag drinken}} \tag{4}$$

An important property of this fragment is that the tree analyses it produces are identical to the ones produced by a simple context-free grammar for English—the English VP order *saw Julia drink coffee* can be read at the top of derivation (4). This gave some depth to hint **A** formulated above. Hint **B** was also satisfied, because verb-second phenomena could be modelled analogously. By adding an extra argument to the VP that is instantiated with the finite verb (*zag* in this case), the grammars could produce the declarative (5) and interrogative (6) variants of the Dutch sentence.

(5) $\text{Frank } \overbrace{\text{zag}}^{v}\ \overbrace{\text{Julia koffie}}^{m}\ [_{\text{VP}(m,v)} \text{ drinken}]$
Frank saw Julia drink coffee

(6) $\overbrace{\text{Zag}}^{v} \text{ Frank } \overbrace{\text{Julia koffie}}^{m}\ [_{\text{VP}(m,v)} \text{ drinken}] \text{ ?}$
Did Frank see Julia drink coffee?

Shortly after these first designs of **LMG**, GERTJAN VAN NOORD suggested a selection of background reading in similar "light-weight" grammatical formalisms. The ability to describe topicalization and verb second however seemed to lift the utility of **LMG** above these similar approaches, such as HEAD GRAMMAR (**HG**), which only gave an account of Dutch subordinate clauses in isolation that was not straightforward to extend to sentence level.

In the following years I wrote several papers about **LMG**, and it was only much later, after several phases of simplification to the design of the **LMG** formalism, that I properly read the papers suggested by Van Noord, and found large similarities between **LMG** and existing TUPLE-BASED grammar formalisms such as LINEAR CONTEXT-FREE REWRITING SYSTEMS or MULTIPLE CONTEXT-FREE GRAMMARS (**LCFRS**, **MCFG**). The key notion connecting these formalisms to **LMG** is a switch from surface order generating *functions* to *relations*, or from a functional rule to something that looks much like a definite clause; in other words, a **Prolog** rule. The production-style **LMG** fragment (2) is then transformed to (7), and this is what **LMG**s will look like in this thesis—but this is merely a matter of presentation.[2]

(7)
$$\begin{array}{lll}
\text{S}(\ldots\texttt{dat}\ nmv) & \texttt{:-} & \text{NP}(n),\ \text{VP}(m,v). \\
\text{VP}(nm, vw) & \texttt{:-} & \text{VR}(v),\ \text{NP}(n),\ \text{VP}(m,w). \\
\text{VP}(n,v) & \texttt{:-} & \text{VT}(v),\ \text{NP}(n).
\end{array}$$

This definite clause notation abstracted a bit further away from the idea of *movement*, an abstraction the literature on **MCFG/LCFRS** had already made, to *placement* of one or more *clusters* the yield of a deep-structural constituent can be divided into. For example, a Dutch verb phrase can, in a very simple setting, be thought of as divided into a *noun cluster* (NC) and a *verb cluster* (VC). In the literature, one finds both analyses which consider the VC to be the core phrase, from which the objects have been extraposed to the left, and analyses which do the opposite: extrapose the verbs to the right. A sensible conclusion seems to be that neither of these should be given the

[2] The old production-style notation version of **LMG** will not be discussed in this thesis, but is not entirely uninteresting. Details of literal movement grammars with slash and colon items, and the definition of a property (left-bindingness) that allows for efficient left-to-right scanning parsers, can be found in my first conference paper [Gro95b]. An advantage of productions over the **Prolog**-clause style is that large parts of a grammar are as easy to read as a **CFG**. This can help translate, *e.g.*, **GB** accounts of linguistic phenomena into a literal movement paradigm (but I have chosen not to do so in chapter **9**).

honour of calling itself a ‘constituent’, but at best they are surface clusters of the one phrase that are equal-right citizens.

(8) . . . dat Frank $[_{\mathrm{NC}}$ Julia koffie $]$ $[_{\mathrm{VC}}$ zag drinken $]$

The relational view seemed to be responsible for an amount of extra strength w.r.t. known tuple-based formalisms such as **MCFG**; this was the ability to describe SHARING at string level: an **LMG** can describe the intersection between two context-free languages, simply by a clause like (9)—to be read as “*x* is recognized as an A if it is both a B and a C” (chapter **3**).

(9) $\mathrm{A}(x)$ **:-** $\mathrm{B}(x), \mathrm{C}(x).$

This ability was the issue of a discussion with DAVID WEIR in Sussex, who believed that the ability to describe intersections was an unfavourable property for a grammar formalism—it leads to undecidability of problems such as emptiness of the generated language, and it means that the resulting class cannot entirely satisfy the desirable properties of an ABSTRACT FAMILY OF LANGUAGES. More important for me at that time however was that Weir did not find my arguments for the tractability of recognition convincing.

As I consequently started to put myself under more pressure to find the proof that a large class of **LMG** could be recognized in polynomial time, it turned out that the wheel I seemed to have more or less re-invented did have a number of favourable properties. In a period when BILL ROUNDS (Michigan) was visiting **CWI**, I found that what a restriction of **LMG**, called SIMPLE **LMG**, added to the existing formalisms exactly made it the ultimately ‘undressed’, or grammar-like, representation of Rounds’ logical system **iLFP**, that describes precisely the languages recognizable in deterministic polynomial time (chapter **5**).

The next step was to prove that there was something in the space between the existing formalisms and **PTIME** that was worthwhile from a linguistic perspective, and such evidence I found in a well-known paper by ALEXIS MANASTER-RAMER [MR87]. At that same time, MARCUS KRACHT and JENS MICHAELIS (Potsdam, Berlin) were investigating counter-examples to the SEMILINEARITY (thought plausible) of Natural Language. I met Marcus at the **ESSLLI** summer school in August 1995, and later at the ***//*** workshop at **CWI**, where Marcus invited me for a short visit to Berlin. When I saw Marcus and Jens in February 1996, they helped me broaden my views as well as to properly work out the details of a number of arguments in favour of **LMG** vs other MILDLY CONTEXT-SENSITIVE grammar formalisms (chapter **8** and [MK96]).

I then started implementing a prototype of a parser based on simple **LMG**, and while writing some example grammars, I became more convinced of the ‘hints’ **A**–**C**, but got stuck, as among others Marcus Kracht had warned me for, on the *rule-based* approach to grammar writing.

Having worked out these subjects until a point where it seemed that I wasn’t able to squeeze out much more, I started worrying that this would be all, and then it seemed to have little original content. Jan van Eijck however convinced me that I would find the opposite if I simply started writing the book with what was available at this point. This was a good recipe, because thinking of my work in terms of a book helped making natural openings to new story-lines visible. Looking back at the resulting part **III** of this thesis, I wish I had started 'thinking in terms of the book' even earlier—not least because having such a concrete project at hand, Jan and I started having more regular, more in-depth, encouraging and rewarding discussions.

At the August 1996 **ESSLLI** summer school in Prague I went to a class on dependency grammar by HAJIČOVÁ, PETKEVIČ and SGALL, and had another conversation with Marcus Kracht. This, together with the excellent atmosphere of the summer school I already knew from the year before in Barcelona, set the stage for an open-minded study of the *principles and parameters* approach to language description, and gave clues as to how I might set up a principled account of language that formalized intuitions put in the back of my mind, such as the mentioned **A**–**C**, by tractability studies and the design of **LMG**.

Two further 'hints', or perhaps 'desiderata', played a rôle in developing such a principled framework. Work on simple **LMG** descriptions, which had polynomial-time recognition, invigorated a belief that linguistic structure should be tractable; another belief that needed to be given formal support was the idea that through literal movement, one could localize a suitably chosen set of basic morphological/selectional dependencies (excluding co-ordination phenomena, but including the relative clause).

E The structure of written language is computationally tractable.

F All relevant syntactic dependencies can be localized.

In parallel to this development I was working out ideas on FERNANDO PEREIRA's *extraposition grammar*, whose graph-shaped structural representations helped getting a grip on the seemingly *circular* dependencies introduced by *wh*-relative clauses. A similar phenomenon occurred when I proved that if modified head grammar is used to describe Dutch sentences with relative or *wh*-pronouns, it either has long-distance dependencies (which I wanted to localize), or it must attach the clause via the relative or interrogative pronoun directly to the verb that the pronoun is the complement of, and not, as is traditionally assumed, attach it at S or CP level. The conclusion seemed to be that the heaviest dependency involved in an object-relative clause such as *(the man) that I saw* is the relation verb (saw)–object (that), so this, if anything, is the "arc" that connects the main clause and the relative clause.

This provided a further motivation for putting forward the principled framework of chapter **10**. This framework, loosely based on POLLARD's **HG** and hence called FREE HEAD GRAMMAR, takes morphological and selectional DEPENDENCY to be the core of grammatical description, inspired by the literature on Dependency Grammar I recently read, by hint **F**, and by the methods of "manually" analyzing Latin and Greek I learned at high school.

Sentences of Latin clearly showed an underlying tree structure that is highly similar to that of English and Dutch, but obviously, the sentence could not be read from the trees by walking through the leaves from left to right. This was of course already not the case for the **LMG** descriptions of Dutch, so it was clear to me that this failure of a language to have a meaningful *constituent structure* was not a privilege of languages like Latin that are called NON-CONFIGURATIONAL in the literature.

Summarizing as before, I now had the following two additional hints:

G Every sentence should have a tree-structured representation, but sometimes a phrase must be allowed to be dominated by a subconstituent.

H It is fruitful to drop the assumption that there should be a tree structure at whose leaves the sentence can be read from left to right.

It was a challenging goal to try to use a cluster analysis, in the form of **FHG**, to describe some phrases from Latin poetry, as examples of extremely free word order. This blissfully turned out to work rather well. The fact that there are five pages on the syntactic analysis of Latin in this thesis seems to stress either, or both, the profetic or educational skills of my parents, who are classical language teachers at the high school I went to; when I started studying Mathematics in 1989 they said that one day I would not be able to resist the temptation to apply whatever knowledge I would gather, to language; moreover, my father always claimed that the 'type of intelligence' needed to appreciate classical languages was no different than that of someone with a good insight in the sciences.

The emphasis on *constituent structure* relativized in hint **H** had always struck me as being inspired by properties of English rather than of language in general, and a clean look at dependency in a framework where languages with a more free word order, such as Latin, are not expelled, made it interesting to investigate the effect of dropping the constituency principle for a while. It would at least give some more insight to a worry that has run through my research as an important motive.

J It seems that the major grammatical theories are *de facto* very much tuned to the description of English. Such a development must be avoided.

A proper introduction of a world of principles and parameters required a note on Government-Binding theory. Thus chapter **9** popped up naturally when I sat down to write chapter **10** that had been available on scribbler paper for a long time; this is the largest deviation from the 'chronological perspective' of this book; it may give the false suggestion that I now have a preference for the perspective sketched in chapter **10**; I also see clear benefits in an approach that remains closer to **GB** theory, which was one of the theories that were subject to concern **J**.

Amsterdam, September 1997

The author's address is

Keizersgracht 684
1017 ET Amsterdam
avg@cwi.nl/zfc@zfc.nl

Chapter 1

Phrase structure and dependency

PHRASE STRUCTURE and DEPENDENCY are the two most essential notions underlying the analysis of linguistic structure. Traditionally, phrase structure, abbreviated as PS, refers to a division of a sentence into PHRASES or CONSTITUENTS that can be represented in a LABELLED BRACKETING of the sentence, as in (1.1).

(1.1) $[_{\text{S}}$ $[_{\text{NP}}$ Frank] $[_{\text{AUX}}$ will]
$[_{\text{VP}}$ $[_{\text{V}}$ poor] $[_{\text{NP}}$ Julia] $[_{\text{NP}}$ $[_{\text{Det}}$ a] $[_{\text{N}}$ cup] $[_{\text{PP}}$ $[_{\text{P}}$ of] $[_{\text{NP}}$ $[_{\text{N}}$ tea]]]]]].

CONTEXT-FREE GRAMMAR (**CFG**) is a mathematical formalism that describes exactly these labelled bracketings of sentences. Important aspects of **CFG** are introduced in section **1.1**.

The choice which words are to be grouped into constituents, and the naming (S, NP, VP) of the constituents is directly influenced, if not defined, by the notion *dependency* between words and phrases, which is introduced in section **1.2**.

Traditional grammatical frameworks such as Government-Binding theory assume a tight interaction between dependency, phrase structure, and word order. This exercises a strong influence on the design of formalisms for grammatical description. At the core of the research reported in this thesis is a tendency to take a small step back with respect to a number of such traditional assumptions.

The meaning of the terms *constituent*, *phrase structure*, *dependency* and so forth differs considerably in various frameworks; most notably when theories built up on phrase structure (**GB**, chapter **9**, **HPSG** [PS94]) are compared with theories that take dependency as the central notion (dependency grammar [Mel88], word grammar [Hud90]). Section **1.3** develops generalized definitions of phrase structure and dependency that enable one to look at these concepts from a distance, and compare their use in different grammatical frameworks without running into terminological clashes.

Finally, section **1.4** gives an overview of parts **I**, **II** and **III** of this thesis, briefly outlining what will covered, and what new results are to be expected.

1.1 Context-free grammar

Although the reader should probably be acquainted with context-free grammar to be able to understand this book, I will define it here for the usual purpose of fixing notation and introducing a first example of a simple grammar, but also because the extensions to **CFG** I will give in chapters **2** and **3** depend on three different ways of looking at grammatical derivation.

1-1 definition. A CONTEXT-FREE GRAMMAR (**CFG**) is a tuple (N, T, S, P) where N and T are disjoint finite sets of NONTERMINAL SYMBOLS and TERMINAL SYMBOLS, $S \in N$ is the START SYMBOL, and P is a finite set of PRODUCTIONS of the form

$$A \;\rightarrow\; X_1 X_2 \cdots X_m$$

where $m \geq 0$, $A \in N$ and $X_i \in (N \cup T)$.

$*$

Some form of context-free grammars is often used at the core of linguistic theories; they are especially suitable for providing a 'rough structural characterization' of languages with an elementary word order, in which compound phrases such as *drank coffee* generally appear unaltered, and uninterrupted, in the complete sentence.

1-2 example: CFG for English. A large amount of structure underlying English syntax can be characterized using a context-free grammar. The following is a very simple example; $G = (N, T, \mathrm{C}^{\mathrm{sub}}, P)$, where

$$\begin{aligned} N &= \{\mathrm{C}^{\mathrm{sub}}, \mathrm{V}^0, \mathrm{V}^{\mathrm{I}}, \mathrm{V}^{\mathrm{T}}, \mathrm{V}^{\mathrm{R}}, \mathrm{N}^0, \mathrm{C}\}, \\ T &= \{\, \texttt{swim}, \texttt{drank}, \texttt{drink}, \texttt{saw}, \texttt{see}, \texttt{hear}, \\ &\quad\;\; \texttt{help}, \texttt{Frank}, \texttt{Julia}, \texttt{Fred}, \texttt{coffee}, \texttt{that} \,\} \end{aligned}$$

and the set of productions P is given in *figure 1.1*.

The nonterminal symbols have the following intuitive meanings: a V^{I} is an INTRANSITIVE VERB, a V^{T} is a TRANSITIVE VERB, a V^{R} a RAISING VERB,[1] an N^0 is a NOUN PHRASE, a V^0 is a complete sentence, and a $\mathrm{C}^{\mathrm{sub}}$ is a subordinate clause.

The rules can be read, informally, as follows: a sentence is made by concatenating a noun phrase and an intransitive verb (production [2]); *swim* is an intransitive verb (production [5]), but, an intransitive verb can also be made by concatenating a transitive verb and a noun phrase (production [3]). So *drank coffee*, just like *swim*, is an "intransitive verb".

1-3 remark. A more traditional naming of categories is CP for $\mathrm{C}^{\mathrm{sub}}$, S or IP for V^0, NP for N^0, VP and V for V^{I}. Multi-word verb phrases and a single intransitive verbs (sometimes called VI or IV) are not distinguished in this thesis, and similar for a

[1] Called raising because of some semantic and surface-order phenomena expressed in Government-Binding theory through forms of movement that will be discussed later (chapter **9**).

verbal complex taking one object and a transitive verb, *&c*. It is customary to present grammars as in *figure 1.1* by just listing the productions P; the start symbol is the one on the left hand side of the first rule in the grammar.

*

The way the grammar was interpreted in example **1-2** is informal; the traditional formal interpretation of a context-free grammar is to read its productions as REWRITE RULES that license the nonterminal symbol on the left hand side of the arrow to be replaced with the sequence on the right hand side of the arrow. It is customary to use the word SEMANTICS for a formal interpretation of a syntactical construction; and a grammar is itself a syntactical construction. This is not to be confused with the use of the word semantics as the interpretation or *meaning* of sentences in a natural language.

1-4 notation. I will use $A, B, C\ldots$ to denote arbitrary nonterminal symbols, a, b, c for arbitrary terminal symbols, X, Y, Z for symbols that are either nonterminal or terminal. The variables u, v, w will stand for *strings*, that is elements of T^*; finally, $\alpha, \beta\ldots$ will stand for arbitrary SEQUENCES of symbols, *i.e.* elements of $(N \cup T)^*$. I will use ε for

[1]	$\mathrm{C}^{\mathrm{sub}}$	→	$\mathrm{C}\ \mathrm{V}^0$
[2]	V^0	→	$\mathrm{N}^0\ \mathrm{V}^{\mathrm{I}}$
[3]	V^{I}	→	$\mathrm{V}^{\mathrm{T}}\ \mathrm{N}^0$
[4]	V^{I}	→	$\mathrm{V}^{\mathrm{R}}\ \mathrm{V}^0$
[5]	V^{I}	→	`swim`
[6]	V^{T}	→	`drank`
[7]	V^{T}	→	`drink`
[8]	V^{R}	→	`saw`
[9]	V^{R}	→	`see`
[10]	V^{R}	→	`hear`
[11]	V^{R}	→	`help`
[12]	N^0	→	`Frank`
[13]	N^0	→	`Julia`
[14]	N^0	→	`Fred`
[15]	N^0	→	`coffee`
[16]	C	→	`that`

Figure 1.1: Simple **CFG** for English.

the empty string or sequence. Concrete nonterminals in example grammars will be in roman, sometimes with a superscript. Concrete terminal symbols will be rendered in a typewriter font. For any string or sequence α, $|\alpha|$ will be the number of symbols in, or the length of α. When R is a production $A \to \alpha$, then A is called the LEFT HAND SIDE (LHS) of R, and α is its RIGHT HAND SIDE (RHS).

1-5 remark. In a context-free grammar for a natural language, the elements of T are usually the *words* of the language. The *strings* $w \in T^*$ are sentences (either correct or incorrect), and sometimes called TERMINAL WORDS. In the literature, it is also customary to call a string a SENTENCE only if it is recognized by the grammar.

I will avoid using this terminology. In a mathematical context, I will talk about TERMINAL symbols and STRINGS only. When talking about language, I will use WORD and SENTENCE in their ordinary informal meanings.

1-6 definition: rewriting semantics for CFG. Let $G = (N, T, S, P)$ be a context free grammar. Then the productions can be read as defining a relation $\Rightarrow$: the smallest relation such that, for all $\alpha, \beta, \gamma \in (N \cup T)^*$, $A \in N$:

$$\alpha A \gamma \Rightarrow \alpha\beta\gamma \text{ if } P \text{ contains } A \to \beta$$

Then define $\overset{*}{\Rightarrow}$ to be the *reflexive, transitive closure* of $\Rightarrow$, that is

1. $\alpha \overset{*}{\Rightarrow} \alpha$ for any sequence $\alpha \in (N \cup T)^*$
2. If $\alpha \Rightarrow \beta$ and $\beta \overset{*}{\Rightarrow} \gamma$ then $\alpha \overset{*}{\Rightarrow} \gamma$

The grammar G is now said to RECOGNIZE a string $w \in T^*$ if and only if $S \overset{*}{\Rightarrow} w$.

1-7 definition: language, CFL. Let T be an alphabet (a set). A LANGUAGE OVER T is a set $L \subseteq T^*$ of strings over T. The LANGUAGE GENERATED BY a **CFG** G is the language $L = \mathcal{L}(G)$ such that $w \in L$ if and only if G recognizes w. A language is called CONTEXT-FREE (a **CFL**) if it is recognized by a **CFG**.

*

The rewriting semantics is a rather *ambiguous* notion of analysis, because symbols can be rewritten in several different orders, even when this order does not seem to have any significance: (1.2) and (1.3) are different derivations, according the grammar of *figure 1.1*, of the same sentence.

(1.2)
$$\begin{aligned} \mathrm{V}^0 &\Rightarrow \mathrm{N}^0\ \mathrm{V}^\mathrm{I} \\ &\Rightarrow \texttt{Julia}\ \mathrm{V}^\mathrm{I} \\ &\Rightarrow \texttt{Julia}\ \mathrm{V}^\mathrm{T}\ \mathrm{N}^0 \\ &\Rightarrow \texttt{Julia drank}\ \mathrm{N}^0 \\ &\Rightarrow \texttt{Julia drank coffee} \end{aligned}$$

(1.3)
$$\begin{aligned} \mathrm{V}^0 &\Rightarrow \mathrm{N}^0\ \mathrm{V}^\mathrm{I} \\ &\Rightarrow \mathrm{N}^0\ \mathrm{V}^\mathrm{T}\ \mathrm{N}^0 \\ &\Rightarrow \mathrm{N}^0\ \mathrm{V}^\mathrm{T}\ \texttt{coffee} \\ &\Rightarrow \mathrm{N}^0\ \texttt{drank coffee} \\ &\Rightarrow \texttt{Julia drank coffee} \end{aligned}$$

Moreover, the derivations are *linear*, glossing over the possibility to view **CFG**s as assigning a TREE STRUCTURE to sentences (*figure 1.2*), or equivalently, writing them in LABELLED-BRACKET FORM (1.4).

(1.4) $[_{\mathrm{V}^0}\ [_{\mathrm{N}^0}\ \text{Julia}]\ [_{\mathrm{V}^\mathrm{I}}\ [_{\mathrm{V}^\mathrm{T}}\ \text{drank}]\ [_{\mathrm{N}^0}\ \text{coffee}]]]$

In this tree analysis, the two rewriting sequences (1.2) and (1.3) correspond to the same tree-shaped derivation. The formal notion of a tree analysis is the following alternative semantics for **CFG**, which is equivalent to the rewriting semantics (proposition **1-10**).

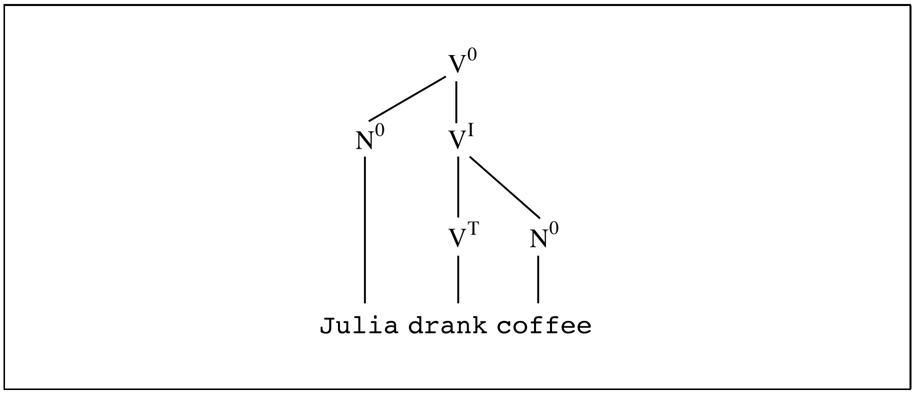


Figure 1.2: Tree analysis of *Julia drank coffee*.

1-8 deÆnition: derivational semantics for CFG. Let $G = (N,\ T,\ S,\ P)$ be a **CFG**. Then define the relation $\vdash$ between nonterminal symbols and terminal strings as follows, by induction on what is called the DEPTH of the derivation.

Base case Let $R \in P$ be a production in P whose RHS contains no nonterminal symbols:

$$A \rightarrow w$$

Then R is called a TERMINAL PRODUCTION and

$$A \vdash^1 w.$$

Inductive step If the RHS of a rule R contains at least one nonterminal symbol:

$$A \;\rightarrow\; w_0B_1w_1B_2w_2\cdots w_{m-1}B_mw_m$$

where $m \geq 1$, $w_i \in T^*$ and $B_i \in N$, R is called a NONTERMINAL PRODUCTION and if, for every $1 \leq k \leq m$,

$$B_k \vdash^{n_k} v_k$$

and $n = \max_k(n_k)$ is the highest value of the n_k,

$$A \vdash^{n+1} w_0v_1w_1v_2w_2\cdots w_{m-1}v_mw_m.$$

Finally, we write $A \vdash w$ if $A \vdash^n w$ for any n.

*

The sentence *Julia drank coffee* is a rather trivial one—the grammar from example **1-2** is capable of generating arbitrarily long sentences by repeating the chain of productions $\text{V}^0 \rightarrow \text{N}^0\ \text{V}^\text{I} \rightarrow \text{N}^0\ \text{V}^\text{R}\ \text{V}^0$. The simplest such sentence is shown in *figure 1.3*. But in principle any number of V^0s can be embedded, producing a sentence of the form (1.5).

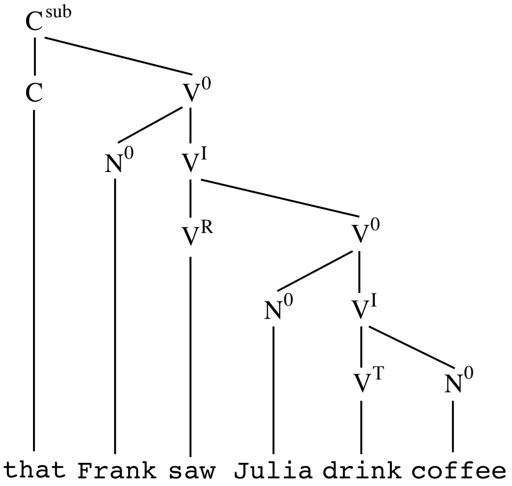


Figure 1.3: Example of verb phrase embedding.

(1.5) Frank saw Julia hear Frank see ⋯
⋯ Julia hear Frank see Julia drink coffee.

It is important to stress that a simple grammar as example **1-2** describes the *underlying structure* of English only; it does not look at morphology and thus approves of syntactically incorrect sentences like *Frank see coffee saw Julia*. Grammars are sometimes further refined by assigning different nonterminal symbols to finite and infinitive verbs and verb phrases. However, it then neglects the structural similarity between the finite and non-finite cases: rule [2] for example would then appear in three nearly identical forms: one for finite verbs, one for infinitives, and one for gerundives—and this is still a highly simplified example, that disregards things like person, number and case. Another refinement is to split N^0 into a category of ANIMATE NOUNS that can appear as the subject of verbs like *drink* and INANIMATE NOUNS like *coffee* that can't. For now, I will say that these properties are not *structural* in nature, so they should preferably not be considered when looking at the relation between underlying structure and word order; I will get back to this issue in chapter **7**.

A third formal interpretation of context-free grammars is in terms of LEAST FIXED POINTS; it is again equivalent to the rewriting interpretation, but gives information about all the nonterminal symbols simultaneously, and gives a link to the treatment of COMPUTATIONAL COMPLEXITY I will give in chapters **4** and **5**.

1-9 definition: fixed point semantics for CFG. Let $G = (N, T, S, P)$ be a **CFG**. Let $\mathcal{NA}$ be the set of ASSIGNMENTS to the nonterminals: functions ρ mapping a nonterminal to a set of strings over T. The set of productions P can then be viewed as an operator $[\![G]\!]$ taking an assignment as an argument and producing a new assignment, defined as follows:

$$(1.6)\qquad \begin{aligned} ([\![G]\!]\rho)(A) \;=\; \{\; & w_0 v_1 w_1 v_2 w_2 \cdots w_{m-1} v_m w_m \mid \\ & \forall k,\ 1 \le k \le m:\ v_k \in \rho(B_k), \\ & A \;\to w_0 B_1 w_1 B_2 w_2 \cdots w_{m-1} B_m w_m \in P \;\} \end{aligned}$$

Now define an order $(\mathcal{NA}, \sqsubseteq)$ by (1.7).

$$(1.7)\qquad \rho_1 \sqsubseteq \rho_2 \;\Leftrightarrow\; \forall A \in N.\ \rho_1(A) \subseteq \rho_2(A)$$

Then $(\mathcal{NA}, \sqsubseteq)$ is a COMPLETE PARTIAL ORDER (**cpo**), because it has a bottom element: the empty assignment ρ_0 given by (1.8), and contains all least upper bounds $\bigsqcup X$ to directed sets $X \in \mathcal{NA}$ given by (1.9).

$$(1.8)\qquad \rho_0(A) = \emptyset \text{ for all } A \in N$$

$$(1.9)\qquad \left(\bigsqcup X\right)(A) \;=\; \bigcup_{\rho \in X} (\rho(A))$$

It is easily seen that $[\![G]\!]$ is a continuous and monotonic operator on $(\mathcal{NA}, \sqsubseteq)$; this means that $\{\ [\![G]\!]^k \rho_0 \mid k \ge 0\ \}$ is a directed set and one can take the least upper bound of this set, the LEAST FIXED POINT of $[\![G]\!]$, to be the interpretation of the grammar:

$$\mathcal{I}_G \;=\; \bigsqcup_{k=0}^{\infty} [\![G]\!]^k \rho_0$$

This interpretation $\mathcal{I}_G$ is a function which takes a nonterminal and yields a set of strings; the language recognized by the grammar in the sense of definition **1-7** is then $\mathcal{I}_G(S)$.

*

Before proving formally that the three interpretations of **CFG** are equivalent, let's look at how the fixed-point interpretation works on the example grammar. The initial assignment ρ_0 assigns the empty set to each of the nonterminals; applying $[\![G]\!]$ to ρ_0 yields $\rho_1 = [\![G]\!]\rho_0$ where

$$\text{(1.10)}\quad \begin{aligned} \rho_1(\mathrm{V}^0) &= \emptyset \\ \rho_1(\mathrm{V}^\mathrm{I}) &= \{\,\texttt{swim}\,\} \\ \rho_1(\mathrm{V}^\mathrm{T}) &= \{\,\texttt{drank}, \texttt{drink}\,\} \\ \rho_1(\mathrm{V}^\mathrm{R}) &= \{\,\texttt{saw}, \texttt{see}, \texttt{hear}, \texttt{help}\,\} \\ \rho_1(\mathrm{N}^0) &= \{\,\texttt{Frank}, \texttt{Julia}, \texttt{Fred}, \texttt{coffee}\,\} \end{aligned}$$

Applying $[\![G]\!]$ again yields $\rho_2 = [\![G]\!]\rho_1$ where

$$\text{(1.11)}\quad \begin{aligned} \rho_2(\mathrm{V}^0) &= \{\,\texttt{Frank swim}, \texttt{Julia swim}, \ldots\,\} \\ \rho_2(\mathrm{V}^\mathrm{I}) &= \{\,\texttt{swim}, \texttt{drank coffee}, \texttt{drank Frank}, \ldots\,\} \end{aligned}$$

The assignments to V^T, V^R and N^0 don't change, because derivations for these nonterminals apply no more than a single production. A third phase yields the first complete sentences:

$$\text{(1.12)}\quad \begin{aligned} \rho_3(\mathrm{V}^0) &= \{\,\texttt{Frank swim}, \texttt{Julia swim}, \\ &\qquad \texttt{Frank drank coffee}, \\ &\qquad \texttt{Frank drank Frank}, \ldots\,\} \\ \rho_3(\mathrm{V}^\mathrm{I}) &= \{\,\texttt{swim}, \texttt{drank coffee},\texttt{drank Frank}, \\ &\qquad \texttt{saw Julia swim}, \ldots\,\} \end{aligned}$$

A fourth phase generates more complex sentences, and so forth.

$$\text{(1.13)}\quad \begin{aligned} \rho_4(\mathrm{V}^0) &= \{\,\texttt{Frank swim}, \texttt{Julia swim}, \\ &\qquad \texttt{Frank drank coffee}, \texttt{Frank drank Frank}, \\ &\qquad \texttt{Frank saw Julia swim}, \ldots\,\} \end{aligned}$$

So the fixed point interpretation sums up, per nonterminal, the derivations of depth 1, those of depth 2, of depth 3, *&c.*

In the context-free case, it is nearly trivial to prove that the three semantics are mutually equivalent; but recall that these three versions were given separately because in more complex grammar formalisms, one type of semantics is easier to define, and hence easier to use for proofs of a formalism's properties, than the other. Because the simple case of **CFG** serves as an example for these more complex proofs, I will go into relatively much detail here.

1-10 proposition. The fixed point, rewriting and derivational interpretations of **CFG** are equivalent.

Proof.

(i) If $A \vdash^n w$ then $A \overset{*}{\Rightarrow} w$.

Suppose $A \vdash^n w$. If $n = 1$, then there is a production

$$A \;\rightarrow\; w$$

So it follows immediately that $A \overset{*}{\Rightarrow} w$. If $n \geq 1$, then $A \vdash^{n+1} w$, means that there is a production

$$A \;\rightarrow\; w_0 B_1 w_1 B_2 w_2 \cdots w_{m-1} B_m w_m$$

and for each $1 \leq k \leq m$ there is a $n_k \leq n$ such that

$$B_k \vdash^{n_k} v_k.$$

and $w = v_{\mathit{1}} \cdots v_m$; now by induction, $B_k \overset{*}{\Rightarrow} v_k$. It is easily seen that $\overset{*}{\Rightarrow}$ is closed under composition and substitution; so

$$\begin{array}{lll} A & \overset{*}{\Rightarrow} & B_1 \cdots B_m \\ & \overset{*}{\Rightarrow} & v_1 B_2 \cdots B_m \\ & \overset{*}{\Rightarrow} & v_1 v_2 B_3 \cdots B_m \\ & \vdots & \\ & \overset{*}{\Rightarrow} & v_1 \cdots v_{m-1} B_m \\ & \overset{*}{\Rightarrow} & v_1 \cdots\cdots v_m \end{array}$$

i.e.

$$A \overset{*}{\Rightarrow} w.$$

(ii) if $S \overset{*}{\Rightarrow} w$ then $S \vdash w$.

To prove this, write $\alpha \overset{n}{\Rightarrow} \beta$ if $\alpha \overbrace{\Rightarrow \;\cdots \Rightarrow}^{n} \beta$, and prove the following stronger result by induction on n: if $\alpha \overset{n}{\Rightarrow} w$ and

$$\alpha = w_0 B_1 w_1 B_2 w_2 \cdots w_{m-1} B_m w_m,$$

then for each $1 \leq k \leq m$, there is a p such that $B_k \vdash^p v_k$, and

$$w = w_0 v_1 w_1 v_2 w_2 \cdots w_{m-1} v_m w_m.$$

(iii) $w \in ([\![G]\!]^k \rho_0)(A)$ if and only if $A \vdash^k w$.

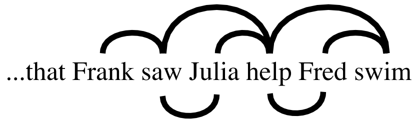
This follows immediately by careful examination of the definitions. □

1.2 Dependency

Before I proceed to developing the meta-theoretical framework and list of points of departure of the next sections, I briefly look at how context-free grammars interact with the linguistic notion of DEPENDENCY in their application to languages such as German and Dutch whose surface structure seems to be slightly more complex than that of English.

A verb and its complements (to take a representative example) are said to be in a relation of IMMEDIATE DEPENDENCY, because the verb assigns accusative case to its object, may assign infinitive or gerundive morphology to its verbal complement, *&c*. Typically, one says that the object and subject *depend on* the verb, but not vice versa. For now, I will ignore this sense of direction in the notion of dependency.[2]

Sentence (1.14) shows the major dependencies in a sentence described by the grammar for English in the previous section.

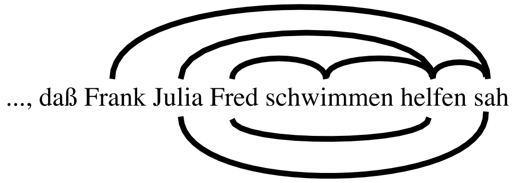


(1.14) ...that Frank saw Julia help Fred swim

It is not a coincidence that one can, with a bit of fantasy, recognize a ‘tree’ in the dependency arcs drawn *above* the sentence in (1.14), and that the arcs correspond to lines in the derivations according to the context-free grammar of *figure 1.1* (this derivation is shown on page 33).

This correspondence between dependency arcs and branches of a context-free derivation is a way of interrelating context-free grammars for different languages, or in more general terms, for relating the underlying structure of languages without looking at their word order.

1-11 example: German, long surface distance, dependency domain.
In subordinate clauses in GERMAN and DUTCH, all the objects in the verb phrase typically precede all the verbs. Sentence (1.15) is the German equivalent of the English sentence (1.14).



(1.15) ..., daß Frank Julia Fred schwimmen helfen sah

In German, as in English, there can in principle be any number of dependent verb-object pairs of type (*sah*, *Julia*). But in contrast to the English case, the dependencies

[2] Disregarding the directionality of immediate dependency does not make sense when looking at the transitivized notion of dependency, because every pair of two phrases or words in a sentence is usually connected through a chain of immediate dependency arcs. However, in those parts of this thesis where I am interested in a BOUNDED DEPENDENCY DOMAIN, the transitivization is limited and it is legitimate to look at undirectional non-immediate dependency.

in German can stretch over an arbitrarily large number of words. I will then say that the language has LONG DISTANCE DEPENDENCIES with respect to the SURFACE (WORD) ORDER.

Because the verb-object pairs are *embedded*, the German equivalent can be generated by a context-free grammar just like the English one by reversing the order of the V^I productions; see *figure 1.4*.

[1]	C^{sub}	→	$C\ V^0$
[2]	V^0	→	$N^0\ V^I$
[3]	V^I	→	$N^0\ V^T$
[4]	V^I	→	$V^0\ V^R$
[5]	V^I	→	`schwimmen`
[6]	V^T	→	`trank`
[7]	V^T	→	`trinken`
[8]	V^R	→	`sah`
[9]	V^R	→	`sehen`
[10]	V^R	→	`hören`
[11]	V^R	→	`helfen`
[12]	N^0	→	`Frank`
[13]	N^0	→	`Julia`
[14]	N^0	→	`Fred`
[15]	N^0	→	`Kaffee`
[16]	C	→	`daß`

Figure 1.4: **CFG** for simple German subordinate clauses.

The German grammar correctly keeps dependent words ‘close to each other’ in its derivation trees; that is there are no *structural* long-distance dependencies. This is formally characterized in the following definition: the German **CFG** analysis has a BOUNDED DEPENDENCY DOMAIN.

1-12 definition. A syntactic description of a fragment of natural language (for example a context-free grammar) is said to have a BOUNDED DEPENDENCY DOMAIN if it assigns some graph-shaped structural analysis to a sentence, and there is a (typically small) number *d*, such that for any sentence, immediately dependent words are connected in the structural representation by a chain of no more than *d* edges.

*

For simple fragments of English and German, I have now showed that context-free grammars can properly capture EMBEDDING such as the V^R construction, CONSTITUENCY or PHRASE STRUCTURE, and the right DEPENDENCY relations. Nonetheless it has been argued many times that **CFG** are inadequate for giving a complete description of any particular natural language.

All such arguments in the literature employ one of the following two typical ways of proving the non-context-freeness of natural language: a WEAK GENERATIVE CAPACITY (**wgc**) argument aims at showing that there is no context-free grammar that generates *precisely* the correct sentences of a given language (say English), without looking at how the grammar analyzes the constructions of the language. A STRONG GENERATIVE CAPACITY (**sgc**) argument assumes an amount of knowledge about the structure of the language, and shows that there is no grammar whose structural analyses respect that knowledge. With **wgc** reasoning non-context-freeness can be decided for a smaller class of languages, but in a way often thought more convincing, because an **sgc** argument assumes theoretical knowledge that one has to *believe* in order to find the argument convincing.

It is perhaps confusing that a weak generative capacity argument has a stronger content than a strong generative capacity argument. The terminology **wgc** and **sgc** has the following definition as a clarifying historical background.

1-13 deÆnition: weak/strong equivalence. Two grammars are WEAKLY EQUIVALENT if they generate the same languages in the sense of definition **1-7**. They are STRONGLY EQUIVALENT if in addition to that, they assign equivalent structural analyses.

*

When context-free grammars are under investigation, both types of argument usually lean on the well-known PUMPING LEMMA, that will be generalized later (**3-10**, **3-11**, **8-8**).

1-14 theorem: pumping lemma for CFG. Let L be a context-free language. Then there is a constant c_0 such that for any $w \in L$ with $|w| > c_0$, there are strings u_0, u_1, u_2 and v_1, v_2 such that v_1 and v_2 are not longer than c_0, at least one of v_1 and v_2 is is not empty, $w = u_0 v_1 u_1 v_2 u_2$ and for any $p \geq 0$, $u_0 v_1^p u_1 v_2^p u_2 \in L$.

Proof. The full proof is in [HU79]. Let L be recognized by a context-free grammar G. Grammar G can be transformed into a CHOMSKY NORMAL FORM grammar $G' = (N, T, S, P)$ that has no EMPTY PRODUCTIONS (productions whose right hand side is ε) except possibly $S \rightarrow \varepsilon$, and no productions with more than 2 symbols on their RHS. Take $c_0 = 2^{|N|}$ to be two to the power of the number of nonterminals in G'. Then if a sentence w is in L and longer than c_0, at least one branch in its G'-derivation in must be longer than the number of nonterminals, so one nonterminal A must appear on this branch more than once. So, using the rewriting semantics, w can be derived as follows:

$$(1.16)\quad \begin{aligned} S &\overset{*}{\Rightarrow} u_0\, A\, u_2 \\ &\overset{*}{\Rightarrow} u_0\, v_1\, A\, v_2\, u_2 \\ &\overset{*}{\Rightarrow} u_0\, v_1\, u_1\, v_2\, u_2 \\ &= w \end{aligned}$$

Now observe that the recursive chain

$$(1.17)\quad A \overset{*}{\Rightarrow} v_1\, A\, v_2$$

in this rewriting sequence can be eliminated ($p = 0$), or iterated ($p > 1$). □

DUTCH is used abundantly in arguments that natural language is beyond the capacity of context-free grammars, both weakly and strongly. Some weak generative capacity arguments however also work in German, such as the following argument done for Dutch in [MR87].

1-15 deÆnition: regular languages. Let some alphabet T be given, and let $a \in T$. Then

- ▷ the SINGLETON a briefly denotes the language containing just the string a;
- ▷ let L_1 and L_2 be languages over T, then the CONCATENATION L_1L_2 is the language containing all concatenations w_1w_2 of any $w_1 \in L_1$ and $w_2 \in L_2$;
- ▷ the UNION $L_1 \cup L_2$ denotes the language which contains all strings that are either in L_1 or in L_2;
- ▷ the ITERATION L^* denotes the language $\{ w^n \mid n \geq 0,\ w \in L \}$ (where w^n stands for n repetitions of the string w).

A language that can be defined using the operations singleton, concatenation, union and iteration is called a REGULAR LANGUAGE. An example is the language $\mathtt{a}^*(\mathtt{b}\cup\mathtt{c})^*\mathtt{d}$ that contains sentences such as `aaaabcbbccbddd`.

It is customary to write L^+ to abbreviate $LL^* = \{ w^n \mid n \geq 1,\ w \in L \}$. The regular languages form a proper subclass of the context-free languages.

1-16 lemma: intersection. The class of context-free languages is closed under intersection with regular languages, that is: if a context-free language is intersected with a regular language, the result is again a context-free language [HU79].

1-17 proposition: linguistic limits of CFG, wgc and German [MR87]. There is no context-free grammar that generates the set of syntactically correct German sentences.

Proof. While the dependency structure of simple German subordinate clauses can be described by a context-free grammar, binary conjunctions as in (1.18) are a construction that cannot be generated by a **CFG**.

(1.18) . . . , daß Frank Julia Fred schwimmen helfen ließ
that swim help let-3SG
und ertrinken lassen sah
and drown let-INF saw

This is proved as follows. Intersecting German with the regular language

(1.19) $$R = \texttt{..., daß Frank Julia}\, N^{*}\, \texttt{schwimmen}\, V^{*}\, \texttt{ließ}$$
$$\texttt{und ertrinken}\, V^{*}\, \texttt{sah}$$

where
$$N = \{\, \texttt{Fred}, \texttt{Dieter}, \texttt{Ute}, \texttt{Beate} \,\}$$
$$V = \{\, \texttt{beibringen}, \texttt{sehen}, \texttt{hören}, \texttt{helfen}, \texttt{lassen} \,\}$$

yields the set

(1.20) $$L = \{\, \texttt{..., daß Frank Julia}\, N^{k}\, \texttt{schwimmen}\, V^{k}\, \texttt{ließ} \; \texttt{und ertrinken}\, V^{k}\, \texttt{sah} \mid k \geq 0 \,\}$$

Now suppose German is context-free. Then so is *L*, by lemma **1-16**. So the pumping lemma **1-14** applies. Look at the sentence in *L* that has $k = c_0$; then the pumping lemma says that new sentences in *L* can be produced from it by pumping two substrings shorter than c_0; it is easy to see that this is impossible; at least 3 substrings will need to be pumped. This is a contradiction, so German is not context-free. □

The English verb phrase structure does not seem to allow for a similar construction, because the second clause cannot leave out the objects, but replaces them with *him*/*her*.

(1.21) . . . that Frank let Julia help Fred swim and saw *her* let *him* drown

The structure of Dutch is even 'harder' for context-free grammars. In 1976, HUYBRECHTS [Huy76] published an argument against the weak context-freeness of Dutch using a technique similar to the above for German, projecting[3] a simple, co-ordination free crossed dependency sentence onto the 2-copy language $\{\, ww \mid w \in \{\texttt{a}, \texttt{b}\}^{*} \,\}$, which is known not to be context-free (use the pumping lemma). This argument was cumbersome, because the dependencies in Dutch are practically invisible in the morphological realization, so effectively, one would end up with something like $\{\, ww \mid w \in a^{*} \,\}$, which is context-free.

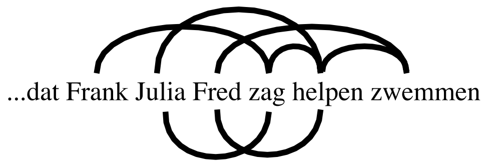
Later arguments either use conjunctions as in **1-17**, or are based on strong generative capacity (see *e.g.* Huybrechts refinement [Huy84], or the following proposition).

1-18 proposition: linguistic limits of CFG; sgc and Dutch.

There is no context-free grammar that describes the set of syntactically correct Dutch sentences and whose derivation trees have a bounded dependency domain.

[3] Through intersection with a regular language and a homomorphism.

Proof. The Dutch equivalent of raising verb sentence shows a form of CROSSED DEPENDENCY: in



(1.22) ...dat Frank Julia Fred zag helpen zwemmen

Frank is linked to *zag*, and all further objects in the sentence appear between *Frank* and *zag*. Since there can be any number of objects, Dutch has inherently long distance dependencies w.r.t. the surface order. But the Dutch verb phrase doesn't have the embedding structure of the German sentences; the pumping lemma can be used to show that in this case there is no context-free grammar that reflects the proper dependencies in its derivational analysis, *i.e.*, that the grammar cannot have a bounded dependency domain.

Suppose there is a **CFG** for Dutch. Then this grammar has a pumping lemma[4] constant c_0. Let s be a sentence longer than c_0 from the following fragment:

(1.23)
$$\begin{array}{rcl} R & = & \ldots\texttt{dat Frank}\, N^k\, \texttt{Julia zag}\, V^k\, \texttt{zwemmen} \\ \text{where} & & \\ N & = & \{\, \texttt{Fred}, \texttt{Pieter}, \texttt{Ada}, \texttt{Bea} \,\} \\ V & = & \{\, \texttt{leren}, \texttt{zien}, \texttt{horen}, \texttt{helpen}, \texttt{laten} \,\} \end{array}$$

then the grammar will recognize correct Dutch sentences obtained from s by pumping two substrings. The fragment is constructed in such a way that correct Dutch sentences obtained by pumping from a sentence in the fragment must again belong to the fragment.[5] Indeed, the two pumped strings must be of the form N^p and V^p.

The construction of the pumping lemma shows us that these strings are generated in an *embedding* structure; the overall derivation looks like

(1.24)
$$\begin{array}{rcl} S & \overset{*}{\Rightarrow} & \texttt{dat Frank}\, u_0\, A\, u_3 \\ & \overset{*}{\Rightarrow} & \texttt{dat Frank}\, u_0\, N^p\, A\, V^p\, u_3 \\ & \overset{*}{\Rightarrow} & \texttt{dat Frank}\, u_0\, N^p\, u_1\, \texttt{Julia zag}\, u_2\, V^p\, u_3 \end{array}$$

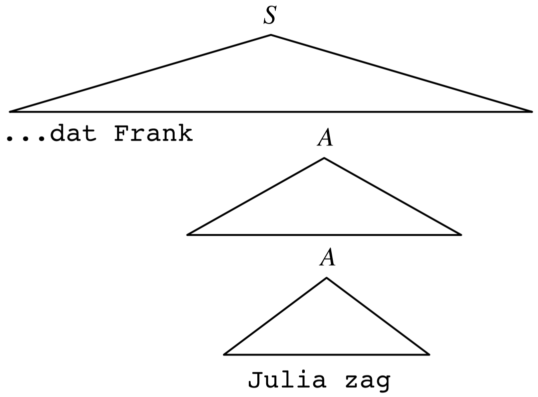
So every repetition of the recursive chain $A \overset{*}{\Rightarrow} N^pAV^p$ will increase the distance

[4] Since the grammar is not necessarily in CNF, this constant will be slightly different than that of lemma 1-14.

[5] Manaster-Ramer points out that in Dutch, the number of objects is in some cases allowed to be smaller than the number of verbs. For details see [MR87], p. 230*ff*.

between the dependent words *Frank* and *zag* in the derivation tree (1.25).

(1.25)



It can now be concluded that there is no context-free analysis of Dutch that has a bounded dependency domain. □

Chapter **8** includes a discussion of other arguments against the weak and strong adequacy of various grammatical formalisms, including **CFG**. This thesis I will investigate straightforward extensions of context-free grammar in which some (chapter **3**), or all (chapter **10**) long-distance dependencies structurally inherent to **CFG** can be localized.

1.3 Some meta-theory

Computational linguists have always attached great value to drawing tree-shaped analyses from which the derived sentence can be read at the bottom going straight from left to right. This is partially due to the rôle context-free grammar has played as a fundamental basis for grammatical description since the early days of generative linguistics. On the other hand, this tendency has been able to survive because linguistic research has always had a considerable focus on English, which has a relatively high correlation between surface order and dependency.

The hardness of the Dutch cross-serial construction (1.22) is often characterized in works on DEPENDENCY GRAMMAR as the property of being NON-PROJECTIVE according to the definitions made in this section.

1-19 definitions: dependency, phrase structure, node. I coarsely divide syntactic theories into DEPENDENCY BASED METHODS and PHRASE STRUCTURE BASED METHODS. A dependency based theory takes *words* as the basic entities, whereas a phrase structure based theory recognizes *phrases*, that combine words or smaller phrases, as additional entities. Some phrase structure based theories also assume the existence of "empty" entities which play certain functional, non-structural, rôles.

For every known grammatical theory, the basic entities will appear in some form as the vertices of a graph- or tree-shaped structure. Abstracting away from the type of theory, I will use the word NODE to refer to a basic entity.

*

Dependency-based frameworks typically arise when so-called *non-configurational* languages are to be described, that is, languages with a very weak interaction between dependency and word order—think of Latin. At the other end, phrase structure based methods perform best on languages where dependency and word order are very tightly related, hence especially on English. Dutch and German are somewhere in the middle of this spectrum, and they often require "untypical" use of the syntactical constructions offered in either type of grammatical frameworks.

1-20 definitions: head, dependent, hierarchy. When two nodes are in an immediate dependency relation to each other, one is often designated the HEAD and the other the DEPENDENT or COMPLEMENT. A complement is strictly required by the head, whereas dependent refers to the more optional case, including, *e.g.* adjectives, adverbs, prepositional phases, *&c.*

I will call a theory STRICTLY HIERARCHICAL if in the analysis of a single full sentence, each node has *at most one* head, and there is only one node, the TOP NODE, that has no head.

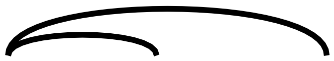
A theory is STRICTLY SELECTIONAL if every dependent is a complement.

*

Context-free grammar qualifies as a strictly hierarchical, phrase structure based syntactical theory. The dependency pictures drawn in the previous section were not strictly hierarchical, because they contained *cycles*.

The directionality of immediate dependency from definition **1-20** is often motivated by the fact that one word SELECTS for a word of another type, and not vice versa. If a transitive verb *kissed* is in a dependency relation with its object *Julia*, then the noun is there because the verb SELECTS for a noun phrase, that is, the verb can appear in a correct sentence only when combined with an object. Therefore, the verb is said to be the head in the dependency relation, and the object is the complement. When there is a more *optional* selection, *e.g.* between a noun and an adjective, one often allows oneself to speak of the adjective as a more general case of dependent of the head noun, but which is not a complement. Another model of the noun–adjective case is the strictly selectional approach of calling the adjective a head and the noun a complement—in this case, the noun will often have more than one head, and the resulting theory cannot be strictly hierarchical.[6]

There has been some interest in the literature [Hud90] [Nic78] for looking seriously at the option of allowing a word or phrase to be governed by more than one head. The dependency pictures from the previous section can be viewed as an example of multi-headedness. Another example is the following relative clause in Latin taken from OVID, Metamorphoses (Pyramus & Thisbe):



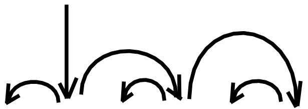
(1.26)

...altera	quas	Oriens	habuit	praelata	puellis
the other	that	the East	had	preferred	girls
F–NOM–SG	F–ACC–PL			F–NOM–SG	F–ABL–PL

‘‘...the other the preferred one among the girls known in the East’’
(Ov. *Met.* IV 56.)

The relative pronoun *quas* receives its accusative case from the verb *habuit*, and its feminine plural from *puellis*.[7]

The following, directional version of the dependency pictures is free of cycles and multi-headedness. Sentence (1.27) shows a generally accepted subset, including directions, of the dependencies in the English subordinate clause (I leave out the relations to the word *that* in all examples). The arcs drawn *below* the sentence in the previous section are considered secondary, and can in fact all be thought of as the composition of two other arcs (the object of a raising verb is the subject of its infinitive verb complement). By a convention due to HUDSON [Hud], a vertical arrow points at the top node.

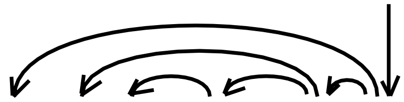


(1.27) (...that) Frank saw Julia help Fred swim

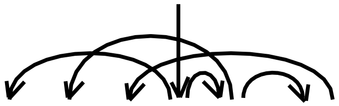
[6] In chapter **10**, a stage is reached where all dependents are complements.

[7] An alternative view, taken in section **10.4**, is that the relative pronoun is an adjunct, and adjuncts are heads, so *quas* is the head of *puellis*.

The German (1.28) and Dutch (1.29) cases can be assigned identical dependency relations; the Dutch dependencies however are "tangled",[8] which makes it impossible to draw a tree analysis such as I did for English and German in the previous section. This is formalized in definition **1-21**.



(1.28) (...daß) Frank Julia Fred schwimmen helfen sah



(1.29) (...dat) Frank Julia Fred zag helpen zwemmen

1-21 deÆnition: projectivity (Hays, Lecerf) [Mel88][Hay64]. Given a selection of dependency relations, a sentence is called PROJECTIVE if, when drawing all arcs linking dependent words as *arrows* from head to dependent, *above* the sentence,

1. no arc is crossing another arc
2. no arc is covering the top node

If one draws, as above, a vertical arrow pointing at the top node, condition 1 implies condition 2, because an arc covers the top node if and only if it crosses the vertical arrow pointing at the top node. Vice versa, if condition 2 is satisfied, a vertical arrow to the top node can be drawn that doesn't cross any arc.

A fragment of natural language (*i.e.*, a language according to definition **1-7**) is projective if all its member sentences are projective.

*

The majority of dependency-based theories assumes some form of the projectivity property, which takes away a great deal of the benefit of a pure dependency approach: easy description of non-projective phenomena such as free word order or crossed dependencies. One of the points of the work in this thesis is that projectivity *with respect to the surface order* is slightly beside the essence of the intuition one wants to capture. Still, one of the aims of this work is to bring the dependency and phrase structure views together; but since I will use the 'literal movement' paradigm to model phenomena like crossed dependencies, projectivity is not precisely the notion that will make this link.

For now, let's stay with the unproblematic case: the grammar for English. The dependency arcs readily suggest a tree representation in which the sentence can be read at the bottom of the tree going left to right. Dependency structure (1.27) is a tree (*figure 1.5a*), and easily transformed to a derivation tree of a context-free grammar (*figure*

[8] The term *tangled* is due to HUDSON, [Hud].

1.5b). By convention, phrase–head arcs like V^0–V^R and V^0–V^I in a tree representation will be drawn vertically. Chains of phrase-head relations are called PROJECTIONS.

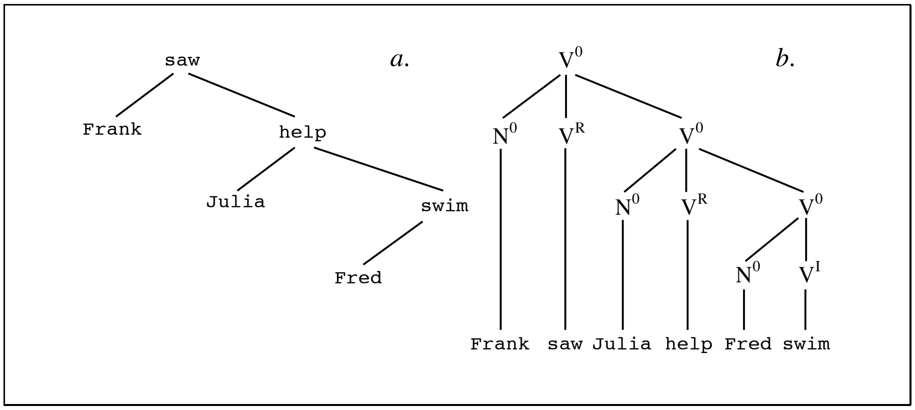


Figure 1.5: From dependency structures to constituent trees (1)

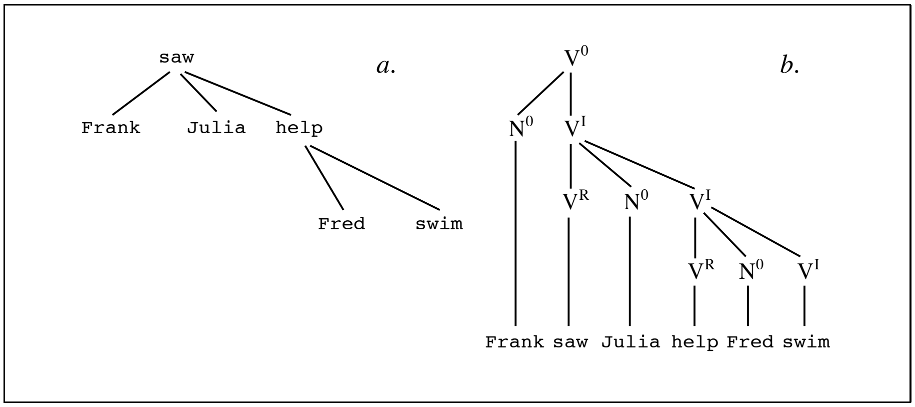
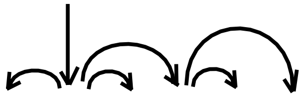


Figure 1.6: From dependency structures to constituent trees (2)

Note that by no means one should take the dependencies as drawn in (1.27) for granted. For example, one might be happier with a direct dependency between a raising verb like *saw* and *help* and its object, because arguably it is *saw* that assigns accusative case to its object *Julia*, and not the infinitive verb *help* to its subject. Such a view on dependencies is highlighted in (1.30). In this case however, verbs have different numbers of complements depending on their morphology—the finite verb *saw* would select for a subject, but the infinite verbs would not.



(1.30) (...that) Frank saw Julia help Fred swim

Figure 1.6 is a free translation of these alternative dependencies (*a.)* to a context-free derivation tree (*b.*); in the constituent paradigm one is free to put in an extra verbal level V^I between the V^0 and the finite V^R so that each type of verbal *constituent* has a unique number of complements. The grammar from section **1.1** captures this generalization; its dependencies are derived from (1.27), but another verbal level is added between the two complements of a V^R; see *figure 1.7.* Arguments in favour of dependency based theories tend to consider the high number of nodes, and the relatively high number of arcs separating heads from their complements as a disadvantage. On the other hand, a structure as in *figure 1.7* has the advantage that properties such as case marking and word order constraints can be easily generalized.

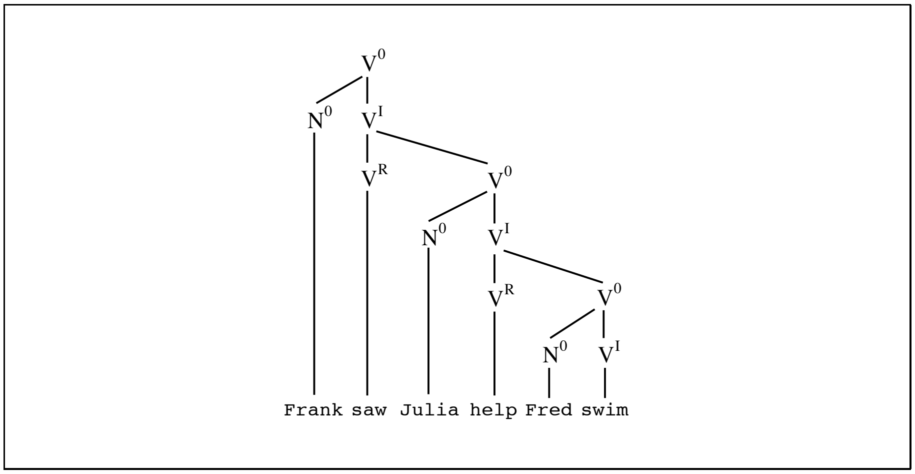


Figure 1.7: Tree according to the context-free grammar on page 15.

As noted before, the dependency structures for German and Dutch are identical to the English ones. By just changing the left-to-right ordering of the branches of the tree representations, a constituent analysis can be drawn for the German sentence; if the dependencies are taken from (1.27) and extended with the same extra level as in the English example, the resulting analysis is as given by the context-free grammar in *figure 1.4*.

Such a development is clearly impossible for the Dutch sentence; the same tangling arcs as in (1.29) will appear in the tentative context-free derivation. So given a set of dependencies, one can draw a constituent analysis if and only if the sentence is projective.

*

In phrase-structure based theories, trees serve two, sometimes conflicting purposes; they capture *dependency* as well as *phrase structure* or *constituency*. Dependency is a relatively straightforward notion, but phrase structure and constituency are a bit ambiguous. PROJECTIVE GRAMMAR FORMALISMS assume that a *phrase* or *constituent*, *i.e.* a logical group of words, is a *continuous* substring of the sentence; but other formalisms may refer to a ‘phrase’ or ‘constituent’ as a group of words that can be interleaved with words from other phrases. Therefore I will avoid these terms, and rather distinguish three different notions of CLUSTERS.

1-22 definitions.

A SURFACE CLUSTER is a word or logical word group that appears unfragmented in the complete sentence; surface clusters can often be combined to form larger surface clusters.

A SURFACE STRUCTURE is a tree from which the sentence can be read by listing the words at its leaves from left to right. In the literature this is often, but not always, called *phrase structure* or *constituent structure*.

A DEPENDENCY CLUSTER is the word group obtained by starting with one node (the HEAD of the dependency cluster), and repeatedly following the immediate dependency arrows departing from each node.

A DEPENDENCY STRUCTURE is the minimal tree containing all the arcs connecting the words in a dependency cluster.

A DEEP CLUSTER is a word or logical word group that coincides with a theory’s notion of a *phrase* or *constituent*. A deep cluster does not necessarily appear unfragmented in the complete sentence, and is not necessarily representable as a (single) string of words. When fragmented, this is sometimes called a *discontinuous constituent* in the literature. Like surface clusters, deep clusters are built of smaller deep clusters.

A DEEP STRUCTURE is the tree obtained by repetitively splitting up a deep cluster into smaller deep clusters.

*

In most theories, some of these notions will coincide. A phrase structure based theory will often identify deep structure with surface structure (the *c*-structure in **LFG**). Because every language will have long-distance dependencies w.r.t. any assigned surface structure, these notions can *not* co-incide with that of a dependency structure. TREE ADJOINING GRAMMAR (**TAG**) is an example of a theory that recognizes surface and deep structures that are clearly distinct. Theories in the GOVERNMENT-BINDING family will not just have a deep structure and a surface structure, but will have longer series of tree structures, one derived from the other. **GB** theory is discussed in chapter **9**.

In the theories developed in this thesis, except in chapter **9** on **GB** theory, deep structure is related to dependency structure in varying degrees of tightness, and surface structure does not play a rôle. Instead, I focus on defining operations, applied recursively at each node of the deep structure, that spell out a sentence.

I will now introduce some terminology to classify the different formalisms in terms of what types of analyses they make.

1-23 deÆnitions: projective formalism. A PROJECTIVE GRAMMAR FORMALISM is a formalism assigning to each sentence a surface structure (possibly among other structures).

A projective grammar formalism can be viewed as a formalism whose grammatical descriptions have, in some sense or other, a context-free grammar at their basis. This is often called the CONTEXT-FREE BACKBONE of a grammar.

*

Non-projective phenomena such as the crossed dependencies in Dutch and Swiss German make that a language cannot be described trivially by projective formalisms, that is not without carrying information over unboundedly long chains of arcs in the deep structure or surface structure.

The analysis of the Dutch cross-serial phrase in the **LFG** formalism [BKPZ82], shown in *figure 1.8*, illustrates how information needs to be carried over arbitrarily long distances in the constituent structure.

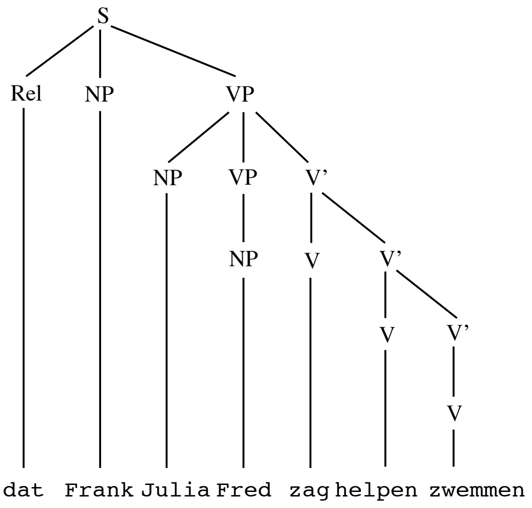


Figure 1.8: **LFG** analysis of the Dutch cross-serial phrase

A formalism based on phrases with a non-continuous representation is MODIFIED HEAD GRAMMAR (**MHG**), discussed in chapter **3**. In **MHG**, a deep cluster is as much as possible also a dependency cluster, and is represented as a string with a "hole", typically left or right of its head, and in addition to the normal way of combining it with one or more other phrases (concatenation), there is an additional operation, WRAPPING.

An example is the deep phrase *easy to please*. It can appear unfragmented (1.31) or fragmented (1.32), so it is not a surface cluster.

(1.31) Kim is **easy** · **to please**.

(1.32) Kim is an **easy** person **to please**.

So *easy* · *to please* can be thought of as having a hole between *easy* and *to please*, and one can say that in the second sentence, it is WRAPPED around the noun *person*.

Wrapping is also capable of solving the crossed-dependency problem in a straightforward manner.[9] Think of the Dutch verb phrase as having a hole between the OBJECT CLUSTER and the VERB CLUSTER it typically consists of.

(1.33) Zag jij **Fred** · **zwemmen**?
Did you see Fred swim?

(1.34) Of Julia **Fred** zag **zwemmen**?
Whether Julia saw Fred swim?

1-24 deÆnitions. The following terminology will be used informally: a language that can be felicitously described using a projective formalism is a CONFIGURATIONAL LANGUAGE.

A grammar formalism that is non-projective, but phrase structure based and preserving a bounded dependency domain is called MILDLY PROJECTIVE. A language that can be described by a mildly projective grammar is called MILDLY CONFIGURATIONAL.

A language that is neither configurational nor mildly configurational is called NON-CONFIGURATIONAL.[10]

*

It would be tempting to call English configurational, Dutch mildly configurational, and Latin non-configurational.

Chapters **3** and **10** of this thesis will be devoted to the development of mildly projective phrase-structure based analyses.

[9] In an isolated fragment containing subordinate clauses only. See chapter **3** and section **8.2** for a full discussion.

[10] The notion of *non*-configurationality is used primarily in Government-Binding contexts to pigeonhole languages that have failed descriptive attempts through the transformational analyses known from the **GB**/**EST** frameworks.

1.4 Overview of the thesis

Now that some formal material and some meta-theoretical notions have been introduced, I can feasibly discuss the three different perspectives on the points of departure **A-J** listed in the **Prologue** that are offered in parts **I**, **II** and **III** of this thesis, and briefly summarize which constructions and results in this thesis are summaries of previously published research and which are new.

I. Formal Structure

In recent years there has been considerable interest in 'light' grammar formalisms aimed at describing linguistic *structure* only, whose descriptive capacity is minimally larger than that of a context free grammar, but which give simple and well-motivated descriptions of non-projective word order phenomena (hints **B** and **C**). A well known example is TREE ADJOINING GRAMMAR (**TAG**, see *e.g.* [Wei88]); less common are LINEAR INDEXED GRAMMAR (**LIG**), HEAD GRAMMAR (**HG**, [Pol84]) and COMBINATORY CATEGORIAL GRAMMAR (**CCG**). **HG**, **LIG**, **CCG** and **TAG** form a mutually equivalent group [VSW94] in a hierarchy of languages described by a TUPLE-BASED formalism called LINEAR CONTEXT-FREE REWRITING SYSTEMS (**LCFRS** or **MCFG**, [Wei88]); this class is further extended with PARALLEL MULTIPLE CONTEXT-FREE GRAMMARS (**PMCFG**) in [KNSK92].

An older formalism with similar purposes, but so far not compared in the literature to the other formalisms I mentioned is EXTRAPOSITION GRAMMAR (**XG**, [Per81]). **XG** can be thought of as formalizing the cyclic dependencies that may arise when describing a relative clause, whereas tuple-based formalisms such as **HG** and **MHG**, are aimed at eliminating long distance dependency altogether.

Chapter **2** opens part **I** by introducing **XG**. New in this chapter are descriptions of Dutch, which shed light on more complex forms of extraposition than covered in [Per81]. In order to extend the coverage of these descriptions, a modification is made to the interpretation of **XG** grammars.

Chapter **3** starts with an extensive overview of tuple-based formalisms from the literature, which are an illustration of hint **H**, putting together results from various sources, and adding an occasional detail that is not covered in the literature, such as the difficulty of describing full Dutch sentences in **MHG**, and whether **PMCFG** describes a set of Dutch and German data that MANASTER-RAMER showed to be beyond the capacity of tree adjoining grammar (developing further the line started in proposition **1-17**). Then, the various tuple grammar formalisms are developed into a new general form called LITERAL MOVEMENT GRAMMAR (**LMG**) that will play a central rôle in this thesis. A well-behaved subclass of this generic formalism is called SIMPLE **LMG**, and essentially extends **MCFG** with the ability to *share* parts of the input string between different branches in a derivation. Tuple grammars have the property that they can describe languages with very different surface orders based on identical tree structures (hint **A**).

II. Computational Tractability

An important property of simple grammatical formalisms (with, in a linguistic context, **CFG** as the extreme example) is that they allow efficient implementation of recognition and parsing procedures (hint **E**). Part **II** of this thesis looks at everything related to computer implementation, in theory and in practice.

In Computer Science, it is customary to express the theoretical performance of algorithms in terms of their COMPUTATIONAL COMPLEXITY: a function expressing how the time and space requirements of the algorithm depend on the size n of the *input* parameters of the algorithm. Given an abstract problem, it is often possible to make predictions about the minimum and maximum complexity of any possible algorithm that solves it. One is usually more interested in these statements about the *complexity of a problem* rather than the complexity of a particular algorithm. A well known example of a complexity statement is

(1.35) Fixed recognition for context-free languages can be performed
in $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space

where the symbol $\mathcal{O}$ is a notation indicating the *order of magnitude* by which the execution time and required space of recognition algorithms grow in terms of the size of input sentence and the size of the grammar.

Statements like (1.35) are explained in more detail in chapter **4**, which also gives various ways for getting to similar results for classes of stronger formalisms like **HG**, **MCFG**, **LMG** and **XG**: two models of computation are introduced, RANDOM ACCESS MACHINES and ALTERNATING TURING MACHINES, and ROUNDS' logical language **iLFP**. New in chapter **4** are the definition of INDEX **LMG** and PREDICATE DESCENT ALGORITHMS, which formally capture the well-known complexity heuristics based on counting the number of positions in the analyzed sentence that play a rôle when checking a grammar rule. Index **LMG** can be thought of as a 'minimal form' of **iLFP** formulae, and the relationship between these two is investigated briefly.

Chapter **5** redevelops a proof cycle carried out in [Rou88] and [CKS81] to demonstrate that simple **LMG** describe precisely the languages that can be recognized in time polynomial in the size of the input. It also includes discussions of how to obtain polynomial time bounds for recognition of arbitrary grammars in the simple **LMG** spectrum, briefly looks at the problem of UNIVERSAL RECOGNITION that has been argued to be intractable, and at stronger forms that have an exponential time recognition.

Chapter **6**, in which I try to obtain similar results for classes of **XG**, is completely original. The standard interpretation in Pereira's sense is first proven to be undecidable. Restrictions on the format and interpretation of **XG** started in chapter **2** are developed further, and lead to the definition of classes that have polynomial-time recognition.

The practical chapter **7** is the first of a series of significantly less formally minded chapters. It looks at an implementation of an **LMG** based parser and practically feasible methods of defining attributes such as morphology and semantic interpretation over packed tree representations output by a parser.

III. Principles

Part **III** investigates how a framework might be built that preserves the intuitions about underlying structure from part **I**, and the tractability results in part **II**, but in a way that respects as many motivated linguistic principles as possible. The chapters in the third part, and in a sense also chapter **7**, are of a much more *tentative* nature than the previous chapters—they aim at a broad, liberal-minded investigation of various approaches to language description without going into detailed formal presentations.

In chapter **8**, the discussion of part **I** is picked up in a more meta-theoretical manner. One of the motivations for the light grammar formalisms of part **I** is that in order to study the nature of linguistic structure in a "theory-independent" fashion, it is valuable to investigate precisely how much formal power is required for an adequate structural description of natural languages. A number of examples from the literature are listed, making up for the rather limited view of the examples on partial verb clause conjunctions in chapter **3**, notions of CONSTANT GROWTH and MILD CONTEXT-SENSITIVITY are introduced. New are revised definitions of mild context-sensitivity and an investigation of the formal power of **MHG** w.r.t. the combination of topicalization, relative clauses and crossed dependencies, while demanding a bounded dependency domain.

Chapters **9** and **10** are inspired by hint **D**. In chapter **9**, I introduce the notion of a PRINCIPLE BASED or EXPLANATORY theory; that is a grammatical theory that is aimed primarily at *explaining* linguistic phenomena, as opposed to merely giving practical descriptions. The **GB** framework, also called the EXTENDED STANDARD THEORY (**EST**) or more generally, the PRINCIPLES AND PARAMETERS (**P&P**) framework, and its successor, the MINIMALIST PROGRAM [Cho96][11] are well known for their broad coverage of linguistic phenomena. Some theories have tried to achieve an equivalent coverage (most notably **HPSG**, [PS94]), but in the field of *descriptive linguistics*, **GB** theory remains the standard framework. New to a certain extent, in this chapter, is only the final section proposing a modification of **GB** that abandons surface structure, and makes use of the LITERAL MOVEMENT construction from chapter **3** instead.

The final chapter **10** pays attention to hints **F** and **G**: it tries to combine research in mildly context-sensitive formalisms, head grammar, dependency grammar and so forth; like the description of **GB** in terms of literal movement, it is largely programmatic in nature, and keeps a large number of design decisions open. It includes a study of the capacity of a syntactic basis slightly extending that of Pollard's **HG** to describe demanding surface order phenomena in Dutch and Latin, and a proposal for an unorthodox way of looking at relative clause attachment.

*

Part **III** *is followed by two reference diagrams (pp. 214, 215), which may help reading this thesis with its wealth of abbreviations, and an* **Epilogue** *that connects this thesis to* **CG** *and* **TAG**, *and gives a broad concluding discussion.*

[11] This book, *The Minimalist Program*, also contains an up-to-date historical overview and an introduction to the **EST**. A more basic introduction is [Hae91].

Part I

Formal Structure

Chapter 2
Extraposition grammar

Of the grammar formalisms defined in this thesis, EXTRAPOSITION GRAMMAR is most close to **CFG**, so it is sensible to use it to open part **I**.

XG, first defined by FERNANDO PEREIRA in [Per81], is a grammar formalism that augments context-free grammar with a dedicated construction modelling LEFTWARD EXTRAPOSITION. Pereira writes that leftward extraposition is a widely used model for describing interrogative sentences and relative clauses in most Western European languages, and these constructions are so essential, even in small real-world applications, that one would like to be able "to express them in a clear and concise manner".

Pereira's paper concentrates on leftward extraposition of noun phrases, such as

(2.1) The mouse [that$_i$ the cat chased ε_i] squeaks.

In the first section of this chapter, I will define **XG** in the form from [Per81] and explain how it can be used to describe such leftward extraposition in English. I then observe that the long-distance dependencies in these sentences have some similarity to those of Dutch, and it will turn out that indeed **XG** give a simple account of Dutch cross-serial relative clauses. But to describe full Dutch sentences, it is convenient to make some modifications to the semantics of **XG** grammars. This is done in section **2.2** and **2.3**.

XG will return in chapter **6**, where it turns out that the modified interpretations of **XG** proposed in this chapter have favourable computational properties.

2.1 Definition and examples

The following definition gives a slightly simplified[1] characterization of the original form of **XG** as in [Per81].

2-1 definition. An EXTRAPOSITION GRAMMAR (**XG**) is a tuple (N, T, S, P) where N and T are disjoint sets of nonterminal and terminal symbols, $S \in N$ is the start symbol, and P is a finite set of productions of the forms

1. $A \rightarrow X_1 X_2 \cdots X_n$ (a CONTEXT-FREE RULE)
2. $A \ldots B \rightarrow X_1 X_2 \cdots X_n$ (an ELLIPSIS RULE)

where $A, B \in N$ and $X_i \in (N \cup T)$.

*

In contrast to the case of **CFG**, derivational and fixed-point semantics are essentially more difficult to define for **XG**, so the interpretation of **XG** given here is an analog of the rewriting semantics **1-6** for **CFG**.

2-2 definition: rewriting semantics for XG. Let $\overline{\alpha}, \overline{\beta}$ and $\overline{\gamma}$ be bracketed sequences of nonterminal and terminal symbols, that is $\overline{\alpha}, \overline{\beta}, \overline{\gamma} \in (N \cup T \cup \{<, >\})^*$, and let a rule of type 1, 2 be in P, then the following hold, respectively:

1. $\overline{\alpha} A \overline{\beta} \Rightarrow \overline{\alpha} X_1 X_2 \cdots X_n \overline{\beta}$
2. $\overline{\alpha} A \overline{\gamma} B \overline{\beta} \Rightarrow \overline{\alpha} X_1 X_2 \cdots X_n {<}\overline{\gamma}{>} \overline{\beta}$

provided that the brackets in $\overline{\gamma}$ are BALANCED. Now G recognizes a string w if there is a bracketing $\overline{w}$ of w such that $S \overset{*}{\Rightarrow} \overline{w}$.

2-3 notation. I will use the notation of **1-4**, with the following additions: $\overline{u}, \overline{v}, \overline{w}$ are (not necessarily balanced) bracketings of terminal strings u, v and w; $\overline{\alpha}, \overline{\beta}, \overline{\gamma}$ are bracketings of sequences α, β, γ of nonterminal and terminal symbols.

2-4 example: English relative clauses. The grammar in *figure 2.1*, taken from [Per81] and slightly modified to fit in the category naming conventions of this book, derives simple English sentences with relative clauses like

(2.2) The mouse that$_i$ the cat chased ε_i squeaks

The traditional analysis of this sentence is that the object of *chased* has been moved, or EXTRAPOSED, leftward. The co-indexed relative pronoun *that$_i$* and the empty element

[1] In the **XG** described in [Per81], the rules of the second type are of the more general form $A_1 \ldots A_2 \ldots \cdots \ldots A_n \rightarrow X_1 X_2 \cdots X_n$. It is not hard to prove directly that the 'bilinear' version I discuss here is weakly equivalent to Pereira's definition; however, the result is for free here once I have proved that the bilinear version describes all r.e. languages.

[1]	V^0	→	$N^0 \ V^I$
[2]	V^I	→	$V^T \ N^0$
[3]	N^0	→	$N^0 \ C^{rel}$
[4]	N^0	→	N^{trace}
[5]	C^{rel}	→	$C^{marker} \ V^0$
[6]	$C^{marker} \ldots N^{trace}$	→	`that`
[7]	V^I	→	`squeaks`
[8]	V^T	→	`chased`
[9]	V^T	→	`likes`
[10]	N^0	→	`the cat`
[11]	N^0	→	`the mouse`
[12]	N^0	→	`fish`

Figure 2.1: Simple grammar for English with relative clauses.

ε_i are called FILLER and TRACE, respectively. The relative clause *that the cat chased* is derived as follows:

$$
\begin{array}{llll}
(2.3) \quad C^{rel} & \Rightarrow & C^{marker} \ V^0 & \text{by rule [5]} \\
 & \Rightarrow & C^{marker} \ N^0 \ V^I & \text{by rule [1]} \\
 & \Rightarrow & C^{marker} \ N^0 \ V^T \ N^0 & \text{by rule [2]} \\
 & \Rightarrow & C^{marker} \ N^0 \ V^T \ N^{trace} & \text{by rule [4]} \\
 & \Rightarrow & \texttt{that} \ <N^0 \ V^T> & \text{by rule [6]} \\
 & \overset{*}{\Rightarrow} & \texttt{that} <\texttt{the cat chased}> &
\end{array}
$$

The derivation graph in *figure 2.2* shows more intuitively how the ellipsis rules establish a link between a relative marker and its trace.

*

A similar technique can be used to describe the crossed dependency structure of Dutch, except that instead of a single noun phrase, a whole series, consisting of *all* the objects in the verb phrase, is extraposed. The **LFG** analysis (*figure 1.8* on page 35) mentioned in the introduction is an example of a noun phrase extraposition model.

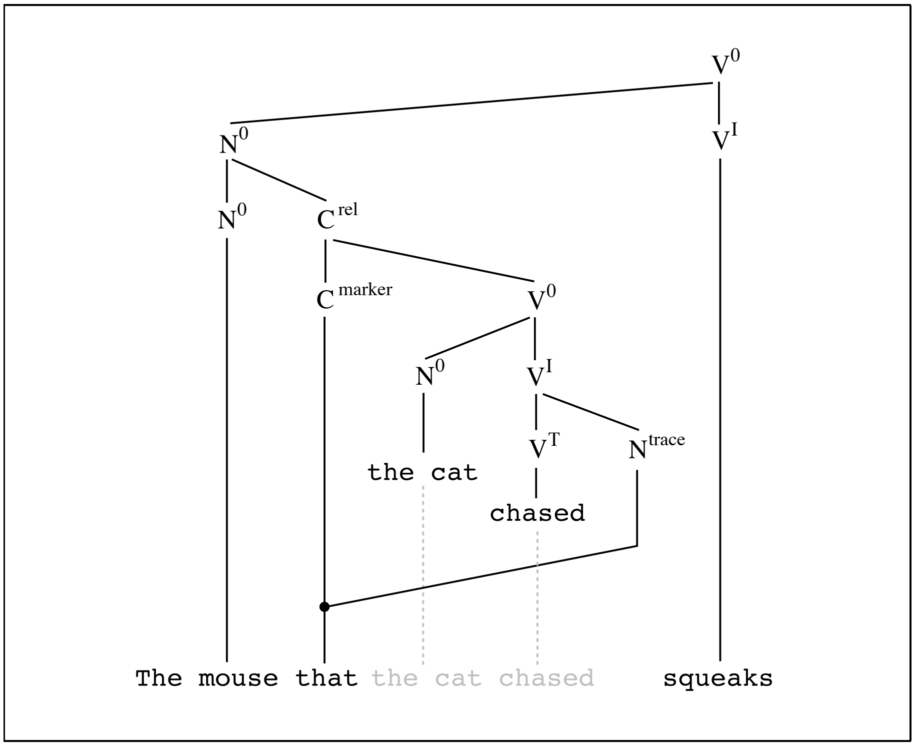


Figure 2.2: Derivation of an English relative clause

[1]	C^{sub}	→	`dat` NC V^0
[2]	V^0	→	N^{trace} V^I
[3]	V^I	→	V^T N^{trace}
[4]	V^I	→	V^R V^0
[5]	NC . . . N^{trace}	→	N^0 NC
[6]	NC	→	ε

Figure 2.3: **XG** for Dutch subordinate clauses.

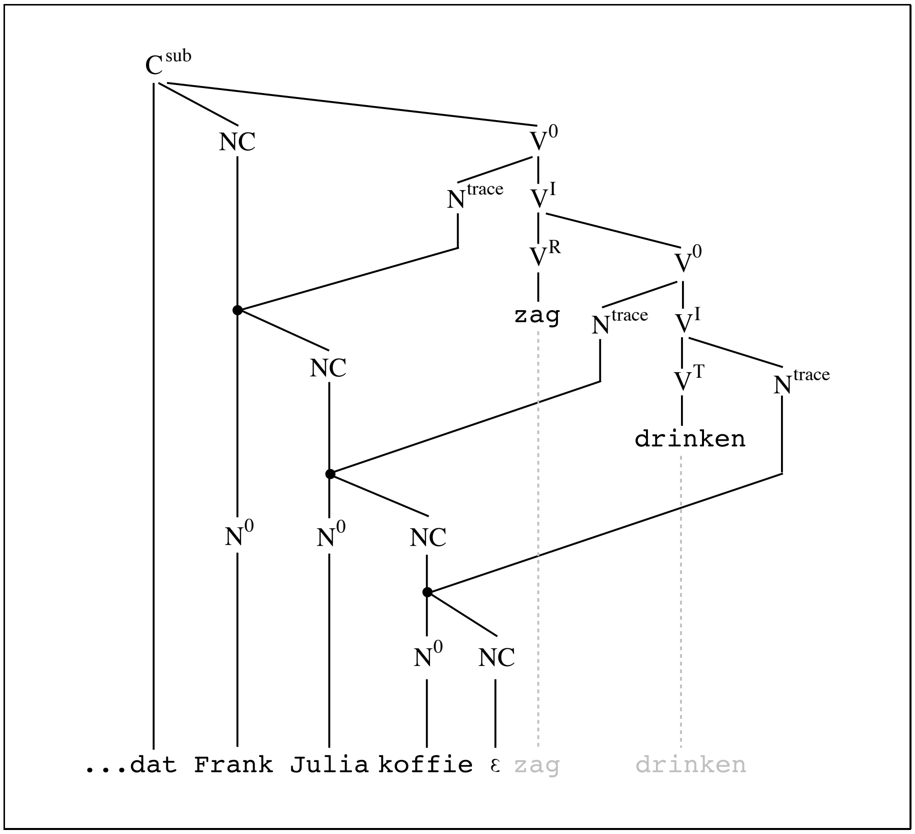


Figure 2.4: Derivation of a Dutch verb phrase.[2]

2-5 example: Dutch crossed dependencies. *Figure 2.3* shows an example grammar that gives a description of Dutch cross-serial subordinate clauses, as in sentence (2.4).

(2.4) . . . dat $[_{\text{NC}}$ Frank$_i$ Julia$_j$ koffie$_k$ $]$ $[_{\text{V}^0}$ ε_i zag ε_j drinken ε_k $]$.
(. . . that Frank saw Julia drink coffee)

The structure this grammar assigns to the verbal clause (V^0) is identical to that of the context-free grammar for English (*figure 1.1* on page 15). However, at subordinate clause level (C^{sub}), it generates a NOUN CLUSTER (NC) that in turn will generate the objects whose traces appear in the actual verb clause V^0. The nominal cluster splits off a noun phrase whenever there is a trace to be eliminated to its right.

[2] A **GB** theorist may find the **XG** analysis unconventional, because to model the *verb raising* construction, the objects in the verb phrase are raised and moved to leftward rather than raising the verbs and moving those rightward. However, as is obvious to a more naive observer of this figure, it is the verbs *zag* and *drinken* that are actually lifted up above the rest of the sentence.

The verb clause derives a string similar to the context-free case, but instead of actual noun phrases, it generates traces only:

(2.5) $$V^0 \overset{*}{\Rightarrow} N^{\text{trace}}\ \texttt{zag}\ N^{\text{trace}}\ \texttt{drinken}\ N^{\text{trace}}$$

Now, the verb clause is embedded into a C^{sub} as follows:

(2.6) $$\begin{array}{llll} C^{\text{sub}} & \Rightarrow & \ldots\texttt{dat}\ NC\ V^0 & \text{by [1]} \\ & \overset{*}{\Rightarrow} & \ldots\texttt{dat}\ NC\ N^{\text{trace}}\ \texttt{zag}\ N^{\text{trace}}\ \texttt{drinken}\ N^{\text{trace}} & (2.5) \\ & \Rightarrow & \ldots\texttt{dat}\ N^0\ NC <> \texttt{zag}\ N^{\text{trace}}\,\texttt{drinken}\ N^{\text{trace}} & \text{by [5]} \\ & \Rightarrow & \ldots\texttt{dat}\ N^0\ N^0\ NC <<> \texttt{zag}> \texttt{drinken}\ N^{\text{trace}} & \text{by [5]} \\ & \Rightarrow & \ldots\texttt{dat}\ N^0\ N^0\ N^0\ NC <<<> \texttt{zag}> \texttt{drinken}> & \text{by [5]} \\ & \Rightarrow & \ldots\texttt{dat}\ N^0\ N^0\ N^0 <<<> \texttt{zag}> \texttt{drinken}> & \text{by [6]} \\ & \overset{*}{\Rightarrow} & \ldots\texttt{dat Frank Julia koffie} <<<> \texttt{zag}> \texttt{drinken}> & \end{array}$$

The corresponding derivation graph is shown in *figure 2.4*. The derivation graph of this example shows intuitively how the balancedness constraint in definition **2-2** works in practice. A sequence is between brackets in the textual derivation, whenever it is enclosed in a CYCLE in the derivation graph. Such cycles may be contained in each other, but the lines of one cycle may not cross those of another cycle. So in terms of derivation graphs, the balancedness constraint in the interpretation of an ellipsis rule ensures that the graphs have a top-down, left-to-right ordered, planar representation.[3]

The balancedness constraint thus enforces that the fillers are co-indexed with the right traces; only one such order is correct in Dutch. An attempt to swap the dependencies between the N^0s and the traces as in (2.7), linking the object of a transitive verb to the first noun phrase in the NC, and the subject to the second, will fail because the balancedness constraint makes that one is then at a dead end in the derivation. The corresponding tentative derivation graph would have crossing lines between the two object fillers N^0 and their traces N^{trace}.

(2.7) $$\begin{array}{lll} C^{\text{sub}} & \overset{*}{\Rightarrow} & NC\ N^{\text{trace}}\ V^T\ N^{\text{trace}} \\ & \overset{*}{\Rightarrow} & N^0\ NC < N^{\text{trace}}\ V^T > \\ & \Rightarrow & \mathbf{??} \end{array}$$

[3] Note that in this planar representation, the lines that are stippled in the figures do not count—these are not part of the derivation graph but added for reasons of aesthetics.

2.2 Dutch sentence order and loose balancedness

It is in the spirit of the ideas expressed in the **Prologue** (hint **B**) and chapter **1** to try to extend the **XG** account of verb clauses to cover placement of the inflected verb in Dutch, using the same leftward extraposition mechanism. Look, for example, at the interrogative sentence (2.8).

(2.8) Zag Frank Julia koffie drinken?
"Did Frank see Julia drink coffee?"

Here *zag* can be thought of as moving leftward out of the verb clause, crossing over the noun cluster, which itself may be linked to traces right of the verb trace. This cannot be done without violating the balancedness constraint, because the finite verb (the chain V^{R}—`zag` in *figure 2.4*) is enclosed in a cycle. Sentence (2.9) adds another complication: *wh*-MOVEMENT (fronting of interrogative pronouns) and TOPICALIZATION (fronting of arbitrary phrases) allow a noun phrase to be moved leftward over the extraposed finite verb.

(2.9) Wie zag jij koffie drinken?
"Who did you see drink coffee?"

[1]	$\mathrm{C}^{\mathrm{sub}}$	→	$\mathtt{dat}\ \mathrm{NC}\ \mathrm{V}^{0}$
[2]	$\mathrm{C}^{\mathrm{wh\text{-}decl}}$	→	$\mathrm{N}^{\mathrm{topic}}\ \mathrm{V}^{\mathrm{topic}}\ \mathrm{NC}\ \mathrm{V}^{0}$
[3]	$\mathrm{C}^{\mathrm{ques}}$	→	$\mathrm{V}^{\mathrm{topic}}\ \mathrm{NC}\ \mathrm{V}^{0}$
[4]	V^{0}	→	$\mathrm{N}^{\mathrm{trace}}\ \mathrm{V}^{\mathrm{I}}$
[5]	V^{I}	→	$\mathrm{V}^{\mathrm{T}}\ \mathrm{N}^{\mathrm{trace}}$
[6]	V^{I}	→	$\mathrm{V}^{\mathrm{R}}\ \mathrm{V}^{0}$
[7]	$\mathrm{V}^{\alpha+\mathrm{fin}}$	→	$\mathrm{V}^{\mathrm{trace}\text{-}\alpha+\mathrm{fin}}$
[8]	$\mathrm{N}^{\mathrm{topic}}\ldots\mathrm{N}^{\mathrm{trace}}$	→	N^{0}
[9]	$\mathrm{V}^{\mathrm{topic}}\ldots\mathrm{V}^{\mathrm{trace}\text{-}\alpha}$	→	V^{α}
[10]	$\mathrm{V}^{\mathrm{topic}}\ldots\mathrm{N}^{\mathrm{trace}}$	→	$\mathrm{N}^{\mathrm{trace}}\ \mathrm{V}^{\mathrm{topic}}$
[11]	$\mathrm{NC}\ldots\mathrm{V}^{\mathrm{trace}\text{-}\alpha}$	→	$\mathrm{V}^{\mathrm{trace}\text{-}\alpha}\ \mathrm{NC}$
[12]	$\mathrm{NC}\ldots\mathrm{N}^{\mathrm{trace}}$	→	$\mathrm{N}^{0}\ \mathrm{NC}$
[13]	$\mathrm{NC}\ldots\mathrm{N}^{\mathrm{trace}}$	→	$\mathrm{N}^{\mathrm{trace}}\ \mathrm{NC}$
[14]	NC	→	ε

Figure 2.5: Successive-cyclic XG for examples (2.8) and (2.9).

The technique used in the grammar of *figure 2.5* exploits, in a naive fashion, the idea of *successive-cyclic movement* known from Government-Binding theory. One speaks of successive-cyclic movement if a phrase moves over a large distance in more than one step; in the current example this is done by allowing the optional verb traces to "step over" the brackets (rule schemas [10] and [11]) introduced in the derivation to prevent the lines connecting nominal filler–trace pairs to cross. Similarly, an N-trace can be eliminated in the NC either by generating a real noun phrase (rule [12]) or by generating a new trace (rule [13]). At most one N-trace, and at most one verb trace are filled at the top level (rules [1]–[3]). Any of the arguments in the verb phrase, including the subject, is allowed to be fronted; the initial position of the subject in a normal declarative sentence is also considered a form of topicalization.

Note that this grammar contains a number of RULE SCHEMATA that actually refer to a *set* of rules. For every verb type, say T for transitive, there will be nonterminals V^{T+fin} and V^{T-fin}, and when V^T appears on the RHS of a rule without further specification (rule [5]), this rule is actually a schema that contains a rule for each of V^{T+fin} and V^{T-fin}.

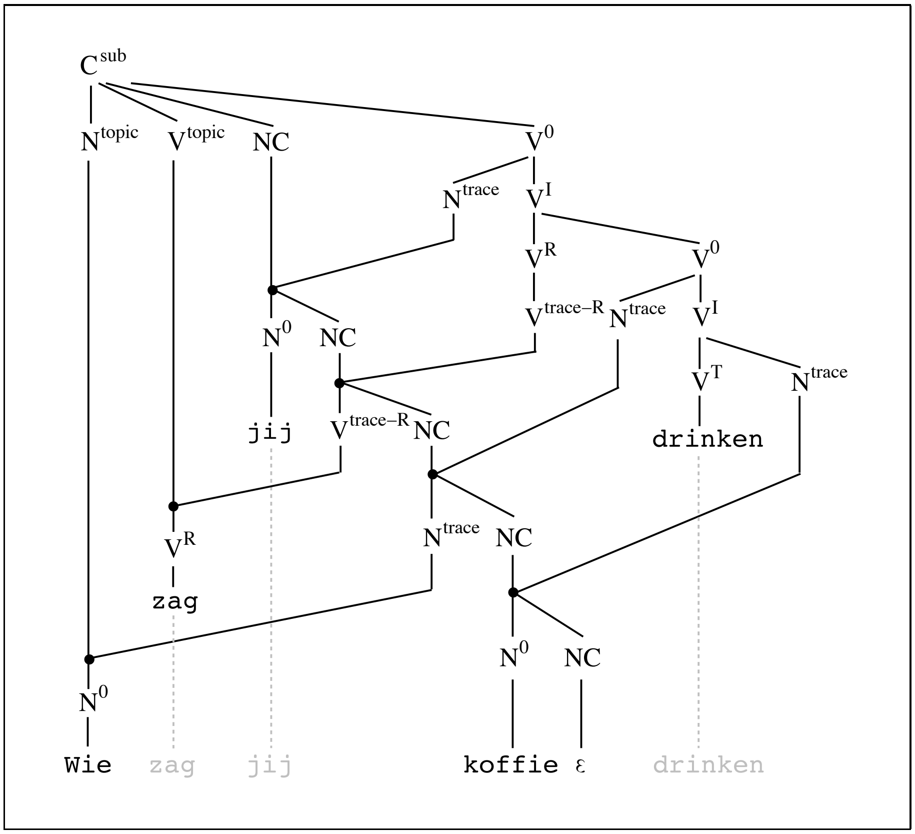


Figure 2.6: **XG** derivation of *Wie zag Frank koffie drinken?*

*

Another, perhaps better, solution to the problem of crossing threads of movement is to modify the semantics of **XG** so as to allow for lines belonging to different filler–trace pairs to cross, rather than devising complex rule systems to bend the laws of the grammar framework.

Hence the following revised rewriting semantics for **XG**.

2-6 definition: loose rewriting semantics for XG.

Let $\overline{\alpha}, \overline{\beta}, \overline{\gamma} \in (N \cup T \cup \{ <_A,\ {}_A> \mid A \in P \})^*$ be sequences of nonterminal and terminal symbols augmented with LABELLED BRACKETS, and let a rule of type 1, 2 be in P, then the following hold, respectively:

1. $\overline{\alpha}A\overline{\beta} \Rightarrow \overline{\alpha}X_1X_2\cdots X_n\overline{\beta}$
2. $\overline{\alpha}A\overline{\gamma}B\overline{\beta} \Rightarrow \overline{\alpha}X_1X_2\cdots X_n<_B\overline{\gamma}\ {}_B>\overline{\beta}$

provided that $\overline{\gamma}$ is balanced w.r.t. each of the labelled bracket pairs $<_A,\ {}_A>$, individually. That is, $<_A<_B\ {}_A>_B>$ is a correct bracketing.

*

The loose interpretation has two advantages. In the current context it is valuable, because it eliminates the need for the successive-cyclic movement that complicated the grammar in *figure 2.5*. In chapter **6** it will turn out to be the key to obtaining computationally tractable classes of extraposition grammars.

Under loose interpretation, rules [10] and [11] disappear altogether; in the corresponding derivation graph, the chain V^{R}—$\mathrm{V}^{\text{trace-R}}$—$\mathrm{V}^{\mathrm{R}}$—`zag` is not interconnected with the NC chain, but is allowed to cross it. The noun traces however are not allowed to cross each other, therefore rule [13] is to remain. A full example is deferred to the next section, where a new type of **XG** rule is introduced.

2.3 An island rule

One of the virtues of the **XG** formalism mentioned in [Per81] is, at first glance, lost under the loose interpretation: the capacity to elegantly describe EXTRAPOSITION ISLANDS by introducing extra brackets.

Figure 2.7 shows how the grammar for English relative clauses of section **2.1** derives sentences such as (2.10) that violate the COMPLEX-NP CONSTRAINT, that forbids a filler from outside a relative clause to be co-indexed with a trace inside it. These relative clauses, enclosed in square brackets in (2.10), are called ISLANDS to extraposition.

(2.10) The mouse [that$_i$ the cat [that$_j$ ε_j chased ε_i] likes fish] squeaks.

The derivation of (2.10) in *figure 2.7* clearly violates the complex-NP constraint. Pereira solves this by replacing the C$^{\text{rel}}$ production [5] by the rules [5] and [5′] shown in *figure 2.8*. This makes it impossible to derive the incorrect sentence (2.10), because the brackets introduced by C$^{\text{marker}}$—N$^{\text{trace}}$ and Begin$^{\text{island}}$—End$^{\text{island}}$ are not balanced. The derivation in *figure 2.9* shows how the pairs Begin$^{\text{island}}$—End$^{\text{island}}$ enclose relative clauses in cycles that cannot be crossed and hence block extraposition.

*

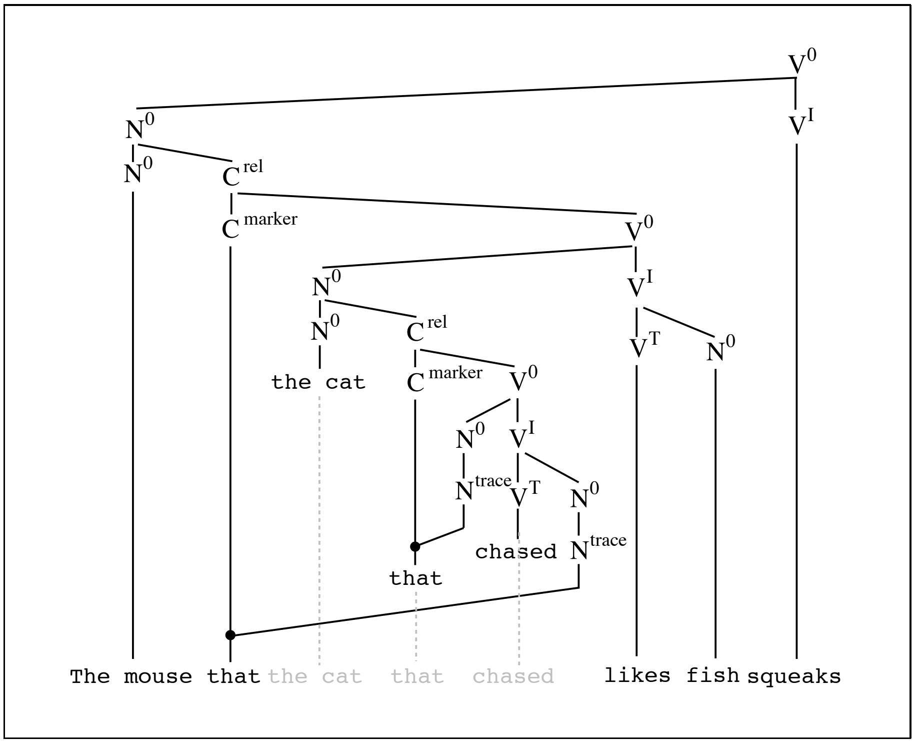


Figure 2.7: Derivation of a sentence violating the complex-NP constraint.

Replace rule [5] in *figure 2.1* with

[5] $\mathrm{C}^{\mathrm{rel}} \rightarrow \mathrm{Begin}^{\mathrm{island}}\ \mathrm{C}^{\mathrm{marker}}\ \mathrm{V}^{0}\ \mathrm{End}^{\mathrm{island}}$
[5′] $\mathrm{Begin}^{\mathrm{island}} \ldots \mathrm{End}^{\mathrm{island}} \rightarrow \varepsilon$

Figure 2.8: Bracketing construction to enforce the complex-NP constraint.

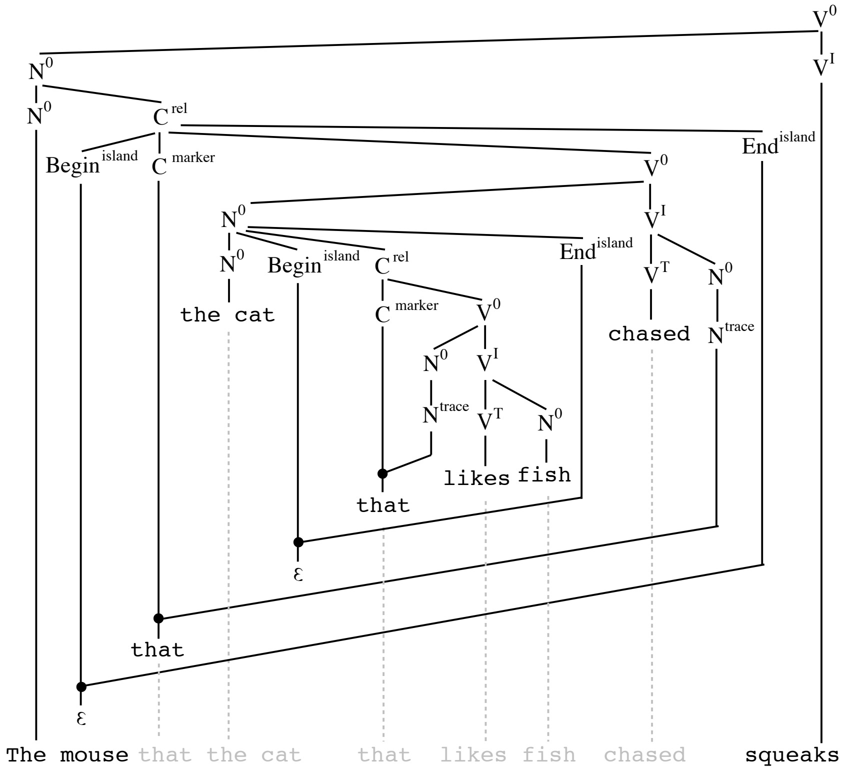


Figure 2.9: Implementation of the complex NP-constraint using multiple brackets

If brackets introduced by different filler–trace pairs are allowed to be unbalanced, as in the *loose interpretation* proposed in the previous section, Pereira's island model will not work. Another revision of the interpretation could solve this issue—one could introduce *classes* of brackets that need to be matched against each other, *i.e.* the brackets introduced by $\mathrm{C}^{\mathrm{marker}}$—$\mathrm{N}^{\mathrm{trace}}$ and $\mathrm{Begin}^{\mathrm{island}}$—$\mathrm{End}^{\mathrm{island}}$ would be required to be properly nested, but a verbal filler–trace pair should be allowed to cross.

Instead of Pereira's method, I propose to add a construction to loosely interpreted **XG** that is dedicated to implementing island restrictions; this rule will preserve the promised tractability results of chapter **6**.

2-7 definition: XG island rule. Extend **XG** with the following rule type:

3. $A(B_1, B_2, \ldots, B_n) \rightarrow X_1 X_2 \cdots X_n$ (an ISLAND RULE)

with the following semantics:

$$\overline{\alpha} A \overline{\beta} \Rightarrow \overline{\alpha} <_{B_1} <_{B_2} \cdots <_{B_n} X_1 X_2 \cdots X_n \; {}_{B_n}\!> \cdots {}_{B_2}\!> {}_{B_1}\!> \overline{\beta}$$

*

An island production allows a nonterminal A to be rewritten only if it then produces a string in which the brackets $<_B$ and $>_B$ are balanced; all extraposition of elements of type $B_1, \ldots, B_n$ will fail because the 'island' is enclosed in brackets.

Such brackets are added explicitly in Pereira's grammar (*figures 2.8 and 2.9*); now they are part of the definition of the island rule. Implementing the complex-NP constraint now amounts to changing the production for $\mathrm{C}^{\mathrm{rel}}$, to say that all $\mathrm{N}^{\mathrm{trace}}$ within a relativized sentence must be matched within that sentence (*figure 2.10*).

Replace rule [5] in *figure 2.1* with

[5] $\mathrm{C}^{\mathrm{rel}}(\mathrm{N}^{\mathrm{trace}}) \quad \rightarrow \quad \mathrm{C}^{\mathrm{marker}} \;\; \mathrm{V}^{0}$

Figure 2.10: Enforcing the complex-NP constraint with an island rule.

*

As said earlier, the Dutch grammar of *figure 2.5* can be reformulated in **XG** under loose interpretation. The graph structures produced will be similar, but the successive-cyclic effect is no longer necessary, and the graph will have less nodes and the grammar has less rules. Such a grammar is shown in *figure 2.11*; a derivation is shown in *figure 2.12*. The island rule eliminates the need to allow only finite verbs to form traces, by making the embedded verbal clause an island to verb traces.

Conclusions to chapter 2

The essential quality of the **XG** approach in the framework of this thesis is that it gives a model of extraposition without a hint of TRANSFORMATION of structures, such as in **GB** theory (chapter **9**). There is one clearly defined sentential structure—be it more complex than a tree—which has elements of both deep structure and surface structure. In fact, the structure in *figure 2.4* is rather similar to the **LFG** analysis (*figure 1.8*), except that the graph shape of the derivations makes the long-distance dependencies explicit (and thus, in a sense, eliminates them).

To put **XG** in the meta-terminology of definition **1-22**, an **XG** derivation graph is a *deep structure* that can be turned into a *surface structure* by eliminating the filler-trace arcs, and into a *dependency structure* by keeping the filler-trace arcs but removing the other arc that connects the filler to its 'structural parent'.

The only other appearance **XG** has made in the literature after Pereira's paper is, as far as I have been able to track down, in [Sta87] which talks about *switch rules* that look remarkably much like a simple form of head wrapping (see chapter **3**). STABLER's approach in this article is symmetric, and thus gives equivalent models for both leftward and rightward extraposition.

This chapter used a considerable amount of material from Pereira's original paper [Per81] on **XG**. The descriptions of Dutch, the loose semantics and the island rule are original.

[1]	C^{sub}	→	`dat` NC V^0
[2]	$C^{\text{wh-decl}}$	→	N^{topic} V^{topic} NC V^0
[3]	C^{ques}	→	V^{topic} NC V^0
[4]	V^0	→	N^{trace} V^{I}
[5]	V^{I}	→	V^{T} N^{trace}
[6]	V^{I}	→	V^{R} V^{island}
[7]	$V^{\text{island}}(V^{\alpha\text{-trace}})$	→	V^0
[8]	N^{trace}	→	$N^{\text{topic-trace}}$
[9]	V^{α}	→	$V^{\text{trace-}\alpha}$
[10]	$N^{\text{topic}}\ldots N^{\text{topic-trace}}$	→	N^0
[11]	$V^{\text{topic}}\ldots V^{\text{trace-}\alpha}$	→	V^{α}
[12]	$\text{NC}\ldots N^{\text{trace}}$	→	N^0 NC
[13]	NC	→	ε

Figure 2.11: Loose/island **XG** for examples (2.8) and (2.9).

I will get back to **XG** in chapter **6** where I will prove that generic **XG** describe any recursively enumerable language, but that the notion of loose interpretation defined in this chapter yields tractable recognition. As was already shown here, the descriptive qualities do not necessarily decrease when different threads of extraposition are allowed to cross each other.

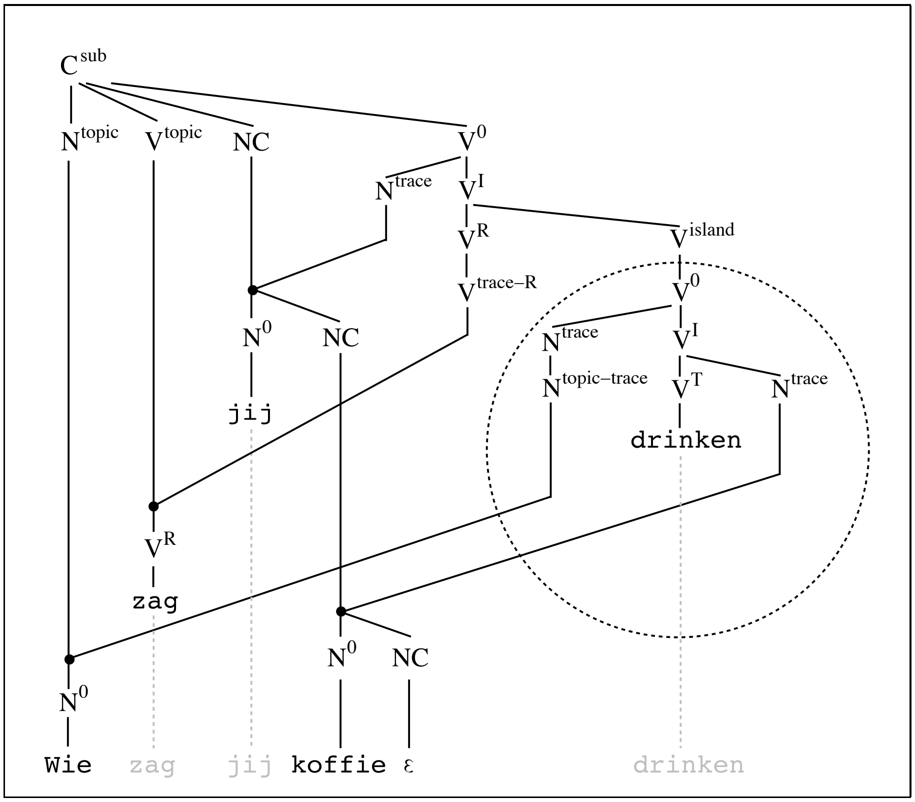


Figure 2.12: Loose/island **XG** derivation of sentence (2.9).

Chapter 3
Tuple-based extensions of context-free grammar

While extraposition grammars are a formalization of the cyclic dependencies that arise in the description of relative clauses, TUPLE-BASED GRAMMARS aim at avoiding structural long-distance dependency altogether, by splitting up the strings generated at the nodes in a derivation tree into two or more SURFACE CLUSTERS that can end up at different places in the full sentence.

In the terminology of definition **1-22**, the derivation graph of a tuple grammar is a deep structure and often also a dependency structure—this depends on how relative clauses are modelled, and will be discussed in more detail in part **III**. Although *surface clusters* can be recognized in the elements of the tuples of strings generated at the nodes in a derivation, these are not part of a distinct *surface structure*.

The simplest tuple-based grammar formalism is MODIFIED HEAD GRAMMAR (**MHG**) and has already been illustrated briefly in chapter **1** (page 35). It is a mathematically motivated simplification of the more linguistically oriented HEAD GRAMMAR of [Pol84] and generates the same class of languages as TREE ADJOINING GRAMMAR (**TAG**), LINEAR INDEXED GRAMMAR (**LIG**) and COMBINATORY CATEGORIAL GRAMMAR (**CCG**).[1] Whereas a context-free grammar generates strings over an alphabet T, and nonterminal nodes combine the strings generated at daughter nodes through concatenation, **MHG** generates *pairs* of strings $\langle w_1, w_2\rangle \in T^* \times T^*$, and in addition to two concatenation operations

$$\begin{array}{lllll} \textbf{HC} & \textit{head-complement}(\langle u_1,u_2\rangle\,,\ \langle v_1,v_2\rangle) & = & \langle u_1,\ u_2v_1v_2\rangle \\ \textbf{CH} & \textit{complement-head}(\langle u_1,u_2\rangle\,,\ \langle v_1,v_2\rangle) & = & \langle u_1u_2v_1,\ v_2\rangle \end{array}$$

an **MHG** rule can *wrap* one pair of strings inside the other:

$$\textbf{HW} \qquad \textit{head-wrap}(\langle u_1,u_2\rangle\,,\ \langle v_1,v_2\rangle) \ = \ \langle v_1u_1,\ u_2v_2\rangle$$

The pairs of strings are also called HEADED STRINGS, for by convention, the first terminal symbol in the second string is the lexical *head* of the construction.[2]

[1] For definitions and the mutual weak equivalence of these four formalisms see [VSW94].

[2] It is important to distinguish two interpretations of the word *head* here: a phrase is divided into two *daughter phrases*, one of which is the *head daughter* and the other the *complement* or *dependent daughter*. A *lexical head* is a single terminal symbol, and is the terminal string produced by the head daughter if this daughter node is a leaf, or to be the lexical head of the head daughter if it is nonterminal. So in the phrase $[_{\mathrm{C}}$the$][_{\mathrm{H}}$blue line with rounded endpoints$]$ the head daughter is the phrase *blue line with rounded endpoints*, and the lexical head is the noun *line*.

[1]	$\mathrm{C}^{\mathrm{sub}}$	$\rightarrow$	*head-complement*$(\mathrm{C},\ \mathrm{V}^0)$
[2]	V^0	$\rightarrow$	*complement-head*$(\mathrm{N}^0,\ \mathrm{V}^{\mathrm{I}})$
[3]	V^{I}	$\rightarrow$	*complement-head*$(\mathrm{N}^0,\ \mathrm{V}^{\mathrm{T}})$
[4]	V^{I}	$\rightarrow$	*head-wrap*$(\mathrm{V}^{\mathrm{R}},\ \mathrm{V}^0)$
[5]	V^{T}	$\rightarrow$	$\langle \varepsilon, \texttt{dronk} \rangle$
[6]	V^{T}	$\rightarrow$	$\langle \varepsilon, \texttt{drinken} \rangle$
[7]	V^{R}	$\rightarrow$	$\langle \varepsilon, \texttt{zag} \rangle$
[8]	N^0	$\rightarrow$	$\langle \varepsilon, \texttt{Frank} \rangle$
[9]	N^0	$\rightarrow$	$\langle \varepsilon, \texttt{Julia} \rangle$
[10]	N^0	$\rightarrow$	$\langle \varepsilon, \texttt{koffie} \rangle$
[11]	C	$\rightarrow$	$\langle \varepsilon, \texttt{dat} \rangle$

Figure 3.1: **MHG** for Dutch subordinate clauses.

The grammars for English and German from chapter **1** can be given an **MHG** equivalent for Dutch that has the same underlying structure as the **CFG**s, except that unlike in the **XG** case, its left-right branching is not identical to the English or German case, but a rather odd mixture of complements left and right of the head. The grammar is shown in *figure 3.1*, and a derivation in *figure 3.2*.

As in the extraposition grammars in the previous chapter, the **MHG** splits up the verbal clause V^0 into a noun cluster and a verb cluster; but in this grammar, these clusters cannot be traced back in the derivation structure—rather can they be found back as the left and right components of the tuples generated by verbal projections. The previously fairly arbitrary choice whether to dislocate the verbs to the right, or the nominal complements to the left, is now irrelevant because the **MHG** description is symmetric in this respect.

Modified head grammars have been generalized to work with tuples of arbitrary size; the resulting formalism, PARALLEL MULTIPLE CONTEXT-FREE GRAMMAR (**PMCFG**, [KNSK92]) is defined formally in section **3.1**. Within **PMCFG**, several subclasses can be recognized, *viz.* **MHG**, LINEAR CONTEXT-FREE REWRITING SYSTEMS (**LCFRS**, [Wei88]) and LINEAR **MCFG**. Some formal properties of these subclasses are treated in sections **3.2**, **3.3** and **3.4**. I then introduce my own extension of the tuple-based grammars, LITERAL MOVEMENT GRAMMAR (**LMG**), and give a classification of all the formalisms presented in this chapter, along with some further formal properties.

3.1 Multiple context-free grammar

To formally define **MHG**, it is best to look at the general case of grammars that make use, instead of concatenation in a **CFG** production, of arbitrary functions manipulating TUPLES of terminal strings. Such grammars are called MULTIPLE CONTEXT-FREE GRAMMARS, and are discussed in [SMFK91] and [KNSK92].

3-1 definition. A PARALLEL[3] MULTIPLE CONTEXT-FREE GRAMMAR (**PMCFG**) is a tuple (N, T, a, S, P) where N and T are disjoint sets of nonterminal and terminal symbols; $S \in N$ is the start symbol; $a : N \rightarrow \mathbb{Z}^+$ assigns an ARITY to each of the nonterminals and P is a finite set of *productions* of the form

$$A \rightarrow f(B_1, \ldots, B_m)$$

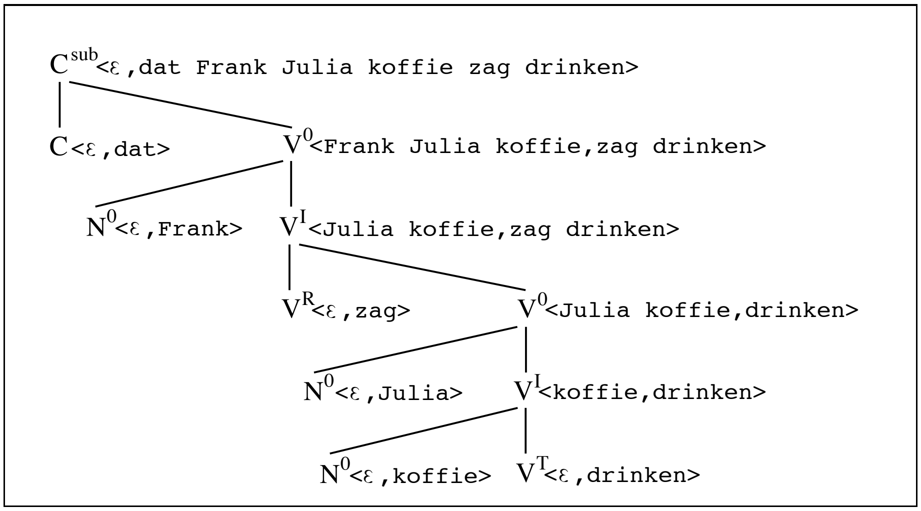
where $m \geq 0$, $A, B_1, \ldots, B_m \in N$, and the YIELD FUNCTION f is a function over tuples of terminal strings, that is, $f : (T^*)^{a(B_1)} \times \cdots \times (T^*)^{a(B_m)} \rightarrow (T^*)^{a(A)}$ can be defined symbolically as

$$f(\langle x_1^1, \ldots, x_{a(B_1)}^1 \rangle, \ \ldots, \ \langle x_1^m, \ldots, x_{a(B_m)}^m \rangle) \ = \ \langle t_1, \ldots, t_{a(A)} \rangle$$

where t_k are strings over terminal symbols and the variables x_j^i. When $m = 0$, the function f is a constant, and I will simply write

$$A \rightarrow \left\langle w_1, \ldots, w_{a(A)} \right\rangle .$$

[3] See **3-6** for why the general case is called *parallel*.



C^sub<ε,dat Frank Julia koffie zag drinken>
C<ε,dat>
V^0<Frank Julia koffie,zag drinken>
N^0<ε,Frank>
V^I<Julia koffie,zag drinken>
V^R<ε,zag>
V^0<Julia koffie,drinken>
N^0<ε,Julia>
V^I<koffie,drinken>
N^0<ε,koffie>
V^T<ε,drinken>

Figure 3.2: **MHG** derivation of a Dutch subordinate clause.

3-2 classiÆcation: yield functions. Let f be a yield function defined by

$$f(\langle x_1^1, \ldots, x_{a_1}^1 \rangle, \ \ldots, \ \langle x_1^m, \ldots, x_{a_m}^m \rangle) \ = \ \langle t_1, \ldots, t_a \rangle$$

1. f is NONERASING if every x_j^i appears at least once in $t_1, \ldots, t_a$. The class of nonerasing yield functions is denoted by $\mathcal{NE}$.
2. f is LINEAR if every x_j^i appears at most once in $t_1, \ldots, t_a$. The class of linear yield functions is denoted by $\mathcal{LIN}$.
3. $\mathcal{HGY} = \{$ *head-complement*, *complement-head*, *head-wrap* $\}$ is the set of HEAD GRAMMAR YIELD FUNCTIONS defined by

HC	*head-complement*$(\langle u_1, u_2 \rangle, \langle v_1, v_2 \rangle)$	$=$	$\langle u_1, \ u_2 v_1 v_2 \rangle$
CH	*complement-head*$(\langle u_1, u_2 \rangle, \langle v_1, v_2 \rangle)$	$=$	$\langle u_1 u_2 v_1, \ v_2 \rangle$
HW	*head-wrap*$(\langle u_1, u_2 \rangle, \langle v_1, v_2 \rangle)$	$=$	$\langle v_1 u_1, \ u_2 v_2 \rangle$

Note that $\mathcal{HGY} \subseteq \mathcal{LIN} \cap \mathcal{NE}$.

*

A semantics for **PMCFG** is best given in a fashion similar to the derivational semantics **1-8** for **CFG**. A fixed point semantics will be discussed for the more general case of **LMG** in chapter **5**.

3-3 deÆnition: derivational semantics for PMCFG. Let $G = (N, T, a, S, P)$ be a **PMCFG**. Then the nonterminals of G recognize tuples of strings, as follows:

Base case If P contains a production $A \to \langle v_1, v_2, \ldots, v_{a(A)} \rangle$, then

$$A \vdash \langle v_1, v_2, \ldots, v_{a(A)} \rangle$$

Inductive step If P contains a production $A \to f(B_1, B_2, \ldots, B_m)$, for each $1 \leq k \leq m$, $B_k \vdash \langle \ v_1^k, v_2^k, \ldots v_{a(B_k)}^k \ \rangle$ and

$$f(\langle v_1^1, \ldots, v_{a(B_1)}^1 \rangle, \ \ldots, \ \langle v_1^m, \ldots, v_{a(B_m)}^m \rangle) \ = \ \langle w_1, \ldots, w_{a(A)} \rangle$$

then

$$A \vdash \langle w_1, \ldots, w_{a(A)} \rangle .$$

Now G is said to RECOGNIZE w if $w = w_1 \cdots w_{a(S)}$ and $S \vdash \langle w_1, \ldots, w_{a(S)} \rangle$.

3-4 classiÆcation: grammar types. Let $G = (N, T, a, S, P)$ be a **PMCFG**. Then

1. If $\mathcal{F}$ is a class of yield functions, then G is an $\mathcal{F}$-**PMCFG** if all yield functions in P are included in $\mathcal{F}$.

2. An $\mathcal{HGY}$-**PMCFG** is called a MODIFIED HEAD GRAMMAR (**MHG**).
3. A $\mathcal{LIN}$-**PMCFG** is called a LINEAR **(P)MCFG**.
4. A $(\mathcal{LIN} \cap \mathcal{NE})$-**PMCFG** is called a LINEAR CONTEXT-FREE REWRITING SYSTEM (**LCFRS**).
5. G is an n-**PMCFG** if for all $A \in N$, $a(A) \leq n$.

*

Non-erasingness is defined because it is psychologically-descriptively relevant, but the following proposition shows that it does not influence weak generative capacity.

3-5 proposition. For every (linear) n-**PMCFL** there is a weakly equivalent (linear) nonerasing n-**PMCFL**.

Proof. [SMFK91] Repeat the following step until the grammar is non-erasing: if a yield function discards a component of its k-th argument B_k, translate the productions for B_k to productions for a nonterminal B'_k that does not have this component. □

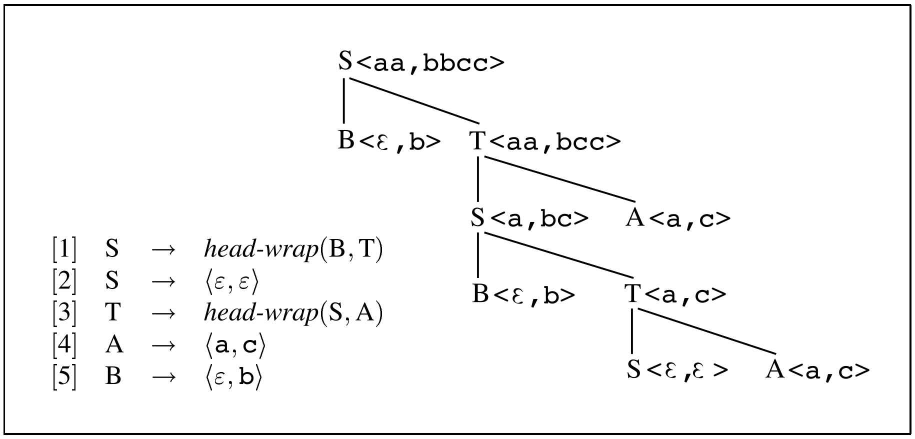
3-6 remark: naming in the literature. Presumably for historical reasons, linear **MCFG** are simply called *mcfg* in the literature [SMFK91] [KNSK92]. I will specifically add the predicate 'linear' to stress that linear **MCFG** are a special case of **PMCFG** and not vice versa. The version which is also nonerasing is also called *mcfg with the information-lossless property* but **LCFRS** is a more widespread name [Wei88]. There are several, tightly related formulations of head grammar. In [Pol84], head grammar refers to an elementary grammar formalism that manipulates *headed strings*, where the head can be either left or right of the 'split point', but also to a larger, 'extended' framework which adds **GPSG**-style features and a linear precedence operator to basic **HG**. The formal equivalent, modified head grammar, was introduced in [Roa87]; its weak equivalence to Pollard's **HG** formalism is also discussed in [SMFK91]. Some features of Pollard's extended **HG** are discussed in chapter **10**.

3.2 ModiÆed head grammar

The introduction to this chapter already showed an **MHG** generating Dutch cross-serial subordinate clauses, which were argued in chapter **1** to be beyond the strong generative capacity of context-free grammar. The following example shows that **MHG** also have a greater weak generative capacity than **CFG**.

3-7 proposition. The class **MHL** of languages recognized by an **MHG** strictly subsumes **CFL**.

Proof. Clearly **CFL** is contained in **MHL**. To see the strict inclusion, consider the 3-counting language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$, which cannot be generated by a context-free grammar; this is proved using the pumping lemma **1-14**. However, the language is recognized by the **MHG** in *figure 3.3*. □



$$\begin{array}{llll}
[1] & \mathrm{S} & \rightarrow & \textit{head-wrap}(\mathrm{B},\mathrm{T}) \\
[2] & \mathrm{S} & \rightarrow & \langle \varepsilon, \varepsilon \rangle \\
[3] & \mathrm{T} & \rightarrow & \textit{head-wrap}(\mathrm{S},\mathrm{A}) \\
[4] & \mathrm{A} & \rightarrow & \langle \mathtt{a}, \mathtt{c} \rangle \\
[5] & \mathrm{B} & \rightarrow & \langle \varepsilon, \mathtt{b} \rangle
\end{array}$$

Figure 3.3: **MHG** for the 3-counting language and a derivation.

It would be desirable to give a description of Dutch in **MHG** along the lines of the **XG** examples in the previous chapter. This can be done to a certain extent, but rather clumsily: the grammar in *figure 3.1* can be extended to generate declarative and verb initial interrogative clauses by dividing the nonterminals for V^{I} and V^{0} into three types each: subordinate, declarative and interrogative. Clearly, a more structural, less feature-driven approach would be preferable.

It is formally proven in chapter **8** that when *wh*-questions and topicalization are added, a fragment is obtained that can be described by an **MHG** only when either (**i**) turning the derivations entirely up-side-down or (**ii**) accepting an unbounded dependency domain.

Therefore in this chapter such examples will be given in linear **MCFG**. An objection that could be raised to making this step to tuples of an arity greater than two, is that in head grammar, and to a smaller degree in **MHG**, the division of constituents into two components has a linguistically motivated foundation—it is not psychologically implausible that the human mind is capable of splitting an already 'derived' constituent up into two pieces using the *head* of the constituent as a handle. Such a linguistically flavoured motivation is harder to obtain for the general case of tuple grammars, but one such motivation will be investigated in chapter **9**.

On the other hand, the motivation for head grammars given in Pollard's [Pol84] does not go further than the following statement: "many discontinuity phenomena can be described by [such] head wrapping operations". Pollard then proceeds to substantiate this claim.

Further in this chapter I will show that a fragment of partial verb phrase conjunctions as used in proposition **1-17** is beyond the weak generative capacity of linear **MCFG** and hence a magno fortiori beyond that of **HG**. A generalization of head grammars is discussed at length in chapter **10**. In the **Epilogue**, section **E.2**, I will briefly touch on the relationship between **HG** and **TAG** in the context of describing mildly configurational languages.

3.3 Linear MCFG

Linear **MCFG**, appearing most often in its non-erasing form that is better known as **LCFRS**, traditionally play a rather formal rôle in the literature on mathematical linguistics. They are equivalent to multi-component tree adjoining grammars (**MC-TAG**, [Wei88]), and form the top of a hierarchy of classes of languages described by grammar formalisms of growing generative capacity [Wei92]. This hierarchy can be formulated in many different ways, as increasingly complex head-wrapping grammars, Dyck grammars or variants of **CCG** or tree adjoining formalisms. The classes in this hierarchy have favourable properties—they are full abstract families of languages (**AFL**, [Gin75]) and correspond to classes of string automata—but their formulation has, as far as I can see, no concrete appeal to any linguistic intuition.

In the context of this thesis, it is more interesting to look at a differently cut hierarchy, which (**i**) extends beyond **LCFRL** to at least **PTIME** and (**ii**) whose member grammar classes are chosen more from a language-engineering point of view than from that of a mathematical linguist.

*

Before I proceed to some formal properties, let's look at how linear **MCFG** can be used to describe Dutch sentences.

3-8 example: Dutch sentence types. If one aims to describe Dutch using a surface cluster model similar to **MHG**, then the following three sentential forms—verb final in (3.1a) and two different forms of verb second[4] in (3.1b) and (3.1c)—are to be given the same structural description except for the top node of category C^{α}, $\alpha =$ **sub**, **decl-wh**, **ques**, and in such a way that the V^0 directly under these clausal phrases C^{α} generate the same tuples of strings.

(3.1) a. . . . dat Frank Julia koffie zag drinken
". . . that Frank saw Julia drink coffee"

b. Frank zag Julia koffie drinken
"Frank saw Julia drink coffee"

c. Zag Frank Julia koffie drinken?
"Did Frank see Julia drink coffee?"

Note now that the only substring of length greater than 1 that appears unfragmented in all of these three sentences is *Julia koffie*. So the minimum number of clusters needed for V^0 is 4.

Figure 3.4 shows a linear and nonerasing 4-**MCFG** that describes (3.1a–c), and a derivation of (3.1b) is shown in *figure 3.5*. The grammar strictly refines the **MHG** from the previous section, in that it still divides the verb phrase into an NC and a VC; but it splits up both clusters further into two components. A verbal clause is now a four-tuple $\langle s, o, h, v\rangle$ consisting of a subject s, the rest of the noun cluster o, the head

[4] Conventionally, a verb in sentence-initial position is also regarded as a result of the 'verb second' construction—this is widely accepted, but rather theory-internal, **GB** terminology.

$$
\begin{array}{llll}
[1] & \mathrm{C}^{\mathrm{sub}} & \to f_1(\mathrm{V}^0) & \text{where } f_1(\langle s,o,h,v\rangle) = \langle\, \texttt{dat}\ \overbrace{s\ o}^{\mathrm{NC}}\ \overbrace{h\ v}^{\mathrm{VC}}\,\rangle \\
[2] & \mathrm{C}^{\mathrm{decl\text{-}wh}} & \to f_2(\mathrm{V}^0) & \text{where } f_2(\langle s,o,h,v\rangle) = \langle s\ h\ o\ v\rangle \\
[3] & \mathrm{C}^{\mathrm{ques}} & \to f_3(\mathrm{V}^0) & \text{where } f_3(\langle s,o,h,v\rangle) = \langle h\ s\ o\ v\rangle \\
\\
[4] & \mathrm{V}^0 & \to f_4(\mathrm{N}^0,\mathrm{V}^{\mathrm{I}}) & \text{where } f_4(\langle s\rangle,\ \langle o,h,v\rangle) = \langle s,o,h,v\rangle \\
[5] & \mathrm{V}^{\mathrm{I}} & \to f_5(\mathrm{V}^{\mathrm{T}},\mathrm{N}^0) & \text{where } f_5(\langle v\rangle,\ \langle o\rangle) = \langle o,\ v,\ \varepsilon\rangle \\
[6] & \mathrm{V}^{\mathrm{I}} & \to f_6(\mathrm{V}^{\mathrm{R}},\mathrm{V}^0) & \text{where } f_6(\langle r\rangle,\ \langle s,o,h,v\rangle) = \langle s\ o,\ r,\ h\ v\rangle \\
\\
[7] & \mathrm{N}^0 & \to \langle\texttt{Frank}\rangle \mid \langle\texttt{Julia}\rangle \mid \langle\texttt{koffie}\rangle & \\
[8] & \mathrm{V}^{\mathrm{I}} & \to \langle\varepsilon,\ \texttt{zwemmen},\ \varepsilon\rangle & \\
[9] & \mathrm{V}^{\mathrm{T}} & \to \langle\texttt{drinken}\rangle & \\
[10] & \mathrm{V}^{\mathrm{R}} & \to \langle\texttt{zag}\rangle &
\end{array}
$$

Figure 3.4: Linear **MCFG** for full Dutch sentences.

verb h, and the rest of the verb cluster v. An intransitive verb phrase V^{I} is a verbal clause without a subject, so a 3-tuple. The correspondence of the variable names to the SVO/SOV/VSO (subject-verb-object, *&c.*) terminology, known from the literature in generative linguistics, helps reading the grammar.

Note also that as in the **XG** examples in the previous chapter, but contrary to the **MHG** example, the left-right branching has been made identical to the context-free structure assigned to English in chapter **1**. While the three yield functions of **MHG**

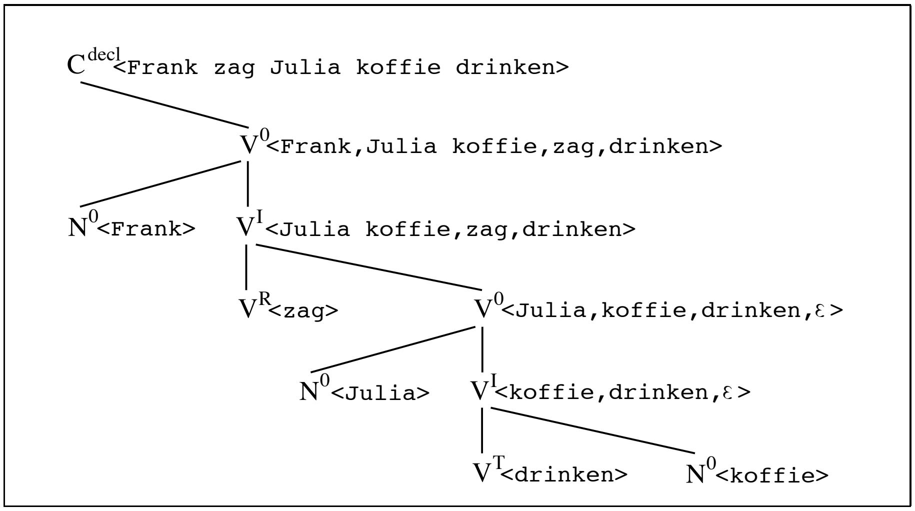

C^decl<Frank zag Julia koffie drinken>
V^0<Frank,Julia koffie,zag,drinken>
N^0<Frank>
V^I<Julia koffie,zag,drinken>
V^R<zag>
V^0<Julia,koffie,drinken,ε>
N^0<Julia>
V^I<koffie,drinken,ε>
V^T<drinken>
N^0<koffie>

Figure 3.5: Derivation of a declarative sentence.

have some repercussion on the relationship between surface order and branching in the derivation tree, this relationship is entirely free in **MCFG**. The grammars can be thought of as two columns—the column left of *where* specifies the structure, *i.e.* immediate dominance, and the right column specifies the surface order. The same structure can be used to underlie languages with totally different word orders, just by modifying the functions in the right column; similarly, the order of the daughter nodes in a production can be swapped in every possible way, when the functions in the right column are modified accordingly. Hence the traditional SVO/SOV/VSO distinction that refers to the left-right order of branching in the *structure* is no longer applicable to the case of **MCFG**.

*

One remaining problem in the example is that a *wh*-question fronting an object from the $\mathrm{V^I}$ rather than the subject is not generated by this grammar. To do this, one must either construct a 5-**MCFG** or differentiate rules driven by a $+/-$**topic** feature.

Formal properties

The above example serves only as supporting a conjecture that **MCFG** provide, w.r.t. **MHG**, additional power of linguistic value. The following results are of a more formal nature.

Counting and copy languages are often used as an indication of the structural capacities of a class of grammars—it is argued that there are parallels between such simple languages and the crossed dependencies in Dutch (see ref. to [Huy84] on page 26, or [Rad91] for more advanced examples).

3-9 example: counting and copy ability.

(i) A class of grammars is said to be ABLE TO COUNT UP TO n if it contains a grammar generating the language $\{\ \mathsf{a}_1^k\mathsf{a}_2^k\cdots\mathsf{a}_n^k \mid k \geq 0\ \}$, where $\mathsf{a}_1,\ldots,\mathsf{a}_n$ are n different terminal symbols. The grammar in *figure 3.6* shows how the class of linear n-**MCFG** can count up to at least $2n$.

$$\begin{array}{llll}
[1] & \mathrm{S} & \rightarrow & f(\mathrm{A}) \\
[2] & \mathrm{A} & \rightarrow & g(\mathrm{A}) \\
[3] & \mathrm{A} & \rightarrow & \langle\varepsilon,\varepsilon\ldots,\varepsilon\rangle
\end{array}$$

$$\begin{array}{rcl}
\text{where}\, f(\langle x_1,x_2,\ldots,x_n\rangle) & = & \langle x_1x_2\cdots x_n\rangle \\
g(\langle x_1,x_2,\ldots,x_n\rangle) & = & \langle \mathsf{a}_1x_1\mathsf{a}_2,\ \mathsf{a}_3x_2\mathsf{a}_4,\ \ldots,\ \mathsf{a}_{2n-1}x_n\mathsf{a}_{2n}\rangle
\end{array}$$

Figure 3.6: n-**MCFG** counting up to $2n$.

(ii) An n-COPY LANGUAGE is the language $\{\ w^n \mid w \in L\ \}$ where L is context-free. For such L there is a context-free grammar $G = (N,T,S,P)$. Let a positive integer n be given. Construct an **MCFG** $G' = (N, T, a, S, P')$ as follows: $a(A) = n$ for all $A \in N$; for each of the productions $R \in P$, if R is a terminal rule $A \to w$, then add to P':

$$A \;\to\; \langle\, \overbrace{w, w, \ldots, w}^{n} \,\rangle$$

and if R is nonterminal:

$$A \;\to\; w_0 B_1 w_1 B_2 w_2 \cdots w_{m-1} B_m w_m$$

add the **MCFG** production

$$A \;\to\; f(B_1, B_2, \ldots, B_m)$$

where

$$\begin{aligned} f(\langle v_1^1, \ldots, v_n^1 \rangle, \ldots, \langle v_n^1, \ldots, v_n^m \rangle) \;=\; \langle\; & w_0 v_1^1 w_1 \cdots w_{m-1} v_1^m w_m, \\ & \ldots, \\ & w_0 v_n^1 w_1 \cdots w_{m-1} v_n^m w_m \;\rangle \end{aligned}$$

Then G' recognizes the language $\{\ w^n \mid w \in L\ \}$. Note that by putting $n = 1$, it is now rigorously shown that every context-free grammar has a weakly equivalent **MCFG**.

*

Now for the converse **3-12** of these examples, which makes use of a generalization of the pumping lemma **1-14** for context-free grammars.[5]

3-10 deÆnition: k-pumpability. A language L is UNIVERSALLY k-PUMPABLE if there are constants c_0 and k such that for any $w \in L$ with $|w| > c_0$, there are strings $u_0, \ldots, u_k$ and $v_1, \ldots, v_k$ such that $w = u_0 v_1 u_1 v_2 u_2 \cdots u_{k-1} v_k u_k$, for each i: $0 \leq |v_i| < c_0$, at least one of the v_i is not the empty string and for any $p \geq 0$, $u_0 v_1^p u_1 v_2^p u_2 \cdots u_{k-1} v_k^p u_k \in L$.

3-11 theorem: pumping lemma for linear MCFL. Let L be recognized by a linear n-**MCFG**. Then L is universally $2n$-pumpable.

Proof. Two straightforward modifications need to be made to the proof of **lemma 3.2** in [SMFK91], p200, which is itself a nontrivial elaboration of the proof of the context-free pumping lemma **1-14**. The existential statement of pumpability needs to be replaced with a universal one ([SMFK91] actually proved this universal version; see in [Rad91], footnote 10 on page 286), and the argument about the size of the v_k needs to be refined. □

[5] The property defined here is called *universal pumpability* because in chapter **8**, definition **8-8**, a version of pumpability is defined for which the number k of pumpable substrings is not fixed.

3-12 corollaries. Let n be a positive integer. Then **(i)** linear n-**MCFG** can count exactly up to $2n$, and **(ii)** n is the highest number m such that linear n-**MCFG** can generate $\{w^m \mid w \in L\}$ for each context-free language L.

*

To conclude this section, I will give the analogue of the partial verb clause conjunctions in **1-17** for linear **MCFG** (*c.q.* **MC-TAG**), briefly mentioned but not worked out in detail by Manaster-Ramer in [MR87]. It is done for German, because the argument for Dutch suffers from the possibility of dropping embedded subjects of raising verbs, which does not seem to be allowed in standard German (see the footnote in chapter **1** on page 27).

3-13 definition. Let two finite terminal alphabets T_1 and T_2 be given. A HOMOMORPHISM from T_1 to T_2 is a map $h : T_1^* \rightarrow T_2^*$ satisfying the constraints $h(uv) = h(u)h(v)$. In other words, a homomorphism is the unique extension of a function of single terminal symbols, *i.e.*, a map $h' : T_1 \rightarrow T_2^*$.

3-14 lemma: closure properties [SMFK91]. For each n, the class of (linear) n-**PMCFL** is a full abstract family of languages (**AFL**), *i.e.* closed under set union, concatenation, iteration, intersection with a regular set, arbitrary homomorphism and inverse homomorphism.[6]

3-15 proposition: linguistic limits of linear MCFG, wgc/German. There is no linear **MCFG** that generates the set of syntactically correct German sentences.

Proof. Extend the the proof of **1-17** to cover arbitrary numbers of conjuncts. Suppose that the set D of syntactically correct German sentences is a linear **MCFL**. Let the sets N, T, and V, the regular language R and the homomorphism h be defined by (3.2)–(3.6),

$$N = \{\,\texttt{Fred}, \texttt{Dieter}, \texttt{Ute}, \texttt{Beate}\,\} \tag{3.2}$$

$$T = \{\,\texttt{küssen}, \texttt{umarmen}, \texttt{einladen}\,\} \tag{3.3}$$

$$V = \{\,\texttt{hören}, \texttt{helfen}, \texttt{lassen}\,\} \tag{3.4}$$

$$\begin{aligned} R = {} & \texttt{Hat Jürgen gesagt daß} \\ & \texttt{Antje Frank } N^* \texttt{ Julia } (\,T\,V^* \texttt{ sah, })^+ \texttt{ und} \\ & \texttt{alle ihr einen schönen Geburtstag wünschten?} \end{aligned} \tag{3.5}$$

$$h(a) = \begin{cases} \varepsilon, & \text{if } a \in \{\,\texttt{Hat}, \texttt{Jürgen}, \texttt{gesagt}, \texttt{daß}, \\ & \qquad \texttt{alle}, \texttt{ihr}, \texttt{einen}, \texttt{schönen}, \\ & \qquad \texttt{Geburtstag}, \texttt{wünschten}\}, \\ \texttt{und}, & \text{if } a \text{ is a comma or question mark,} \\ \texttt{verführen}, & \text{if } a \in T, \\ a, & \text{otherwise} \end{cases} \tag{3.6}$$

[6] The proof given in [SMFK91] seems to be convincing only for the case of linear **MCFG**. See page 80 at the end of this chapter for details.

then by lemma **3-14** the fragment L, defined by (3.7), must be described by a linear **MCFL**.

(3.7) $$L = h(D \cap R) = \{\, \texttt{Antje Frank} \;\; N^n \;\; \texttt{Julia} \;\; (\texttt{verführen} \;\; V^n \;\; \texttt{sah und})^+ \mid n \geq 1 \,\}$$

In this example, one can't simply say that for a sentence with k conjuncts, $k+1$ substrings need to be pumped, because one can always repeat a single conjunct. However, linear **MCFG** satisfy the pumping property of definition **3-10**, which also states that the size of the pumped substrings has a fixed upper bound c_0. If the fragment is a linear **MCFL**, then there is be a k such that it is k-pumpable. In sentence (3.8), the conjuncts are longer than c_0 so they cannot be pumped as a whole; so new sentences within the fragment can indeed be created only by pumping $k+1$ sequences: substrings of $\texttt{Ute}^{c_0}$ and each of the verb sequences $\texttt{lassen}^{c_0}$.

(3.8) $$\texttt{Antje Frank Ute}^{c_0} \;\; \texttt{Julia} \;\; (\texttt{verführen lassen}^{c_0} \; \texttt{sah und})^k$$

So the fragment is not k-pumpable, and it follows that German cannot be described by a linear **MCFG**. □

3.4 Parallel MCFG

Generic **PMCFG** extends linear **MCFG** in providing a mechanism for REDUPLICATION. As such it can give accounts for a number of phenomena that involve counting. Linguistic examples of such phenomena are Chinese number names [Rad91], *respectively*-constructs, and Old Georgian genitive suffix stacking [MK96]; these are discussed in chapter **8**.

3-16 proposition. Linear **MCFL** is strictly included in **PMCFL**.

Proof. The **PMCFG** in *figure 3.7* generates the language $\mathtt{a}^{2^n}$, which is not k-pumpable for any k. □

$$\begin{array}{llll} [1] & \mathrm{S} & \rightarrow & f(\mathrm{S}) \quad \text{where}\, f(\langle x\rangle) \;=\; \langle xx\rangle \\ [2] & \mathrm{S} & \rightarrow & \langle \mathtt{a}\rangle \end{array}$$

Figure 3.7: **PMCFG** generating a non-pumpable language.

Because no formulation of the pumping lemma is known for **PMCFG**, the argument in **3-15** that linear **MCFG** cannot generate German structure will fail for **PMCFG**. The following example even shows, be it in a very abstract an indirect way, that there is an **PMCFG** that generates the fragment L used in **3-15**.

3-17 example: Partial verb clause conjunctions are a PMCFL.

The grammar in *figure 3.8* describes a fragment F that is slightly more simple than the fragment L used in **3-15**.

$$(3.9) \qquad F = \{\, \texttt{Antje Frank} \quad \texttt{Ute}^n \quad \texttt{Julia} \quad (\texttt{verführen} \quad \texttt{lassen}^n \quad \texttt{sah und})^+ \mid n \geq 0 \,\}$$

The fragment can be described so easily by a **PMCFG** because it allows only one infinitive raising verb, *lassen*, so that the sequence $\texttt{lassen}^k$ can be generated once and then be reduplicated. But because **PMCFG** is closed under inverse homomorphism, we can map the simple fragment back to one with a richer selection of verbs. It is, however, not straightforward to write a concrete grammar generating F, and the construction of the proof that **PMCFG** is closed under h^{-1} does not seem to be of much help (a direct proof is rarely given because laborious).

$$(3.10) \qquad h(a) \;=\; \begin{cases} \texttt{Ute}, & \text{if } a \in \{\texttt{Fred}, \texttt{Dieter}, \texttt{Beate}\} \\ \texttt{lassen}, & \text{if } a \in \{\texttt{hören}, \texttt{helfen}\} \\ a, & \text{otherwise} \end{cases}$$

$$\begin{aligned} (3.11) \qquad h^{-1}(F) \;&=\; \{\, \texttt{Antje Frank} \quad N^n \quad \texttt{Julia} \quad (\texttt{verführen} \quad V^n \quad \texttt{sah und})^+ \mid n \geq 1 \,\} \\ &=\; L \end{aligned}$$

3.5 Literal movement grammar

The notation of **MCFG** makes that these grammars are rather difficult to read. There is an amount of redundancy in the idea that the grammar productions refer to a function, and this yield function is specified separately. LITERAL MOVEMENT GRAMMAR circumvents this problem by moving the variables over terminal strings straight into the production using a *definite clause*-style notation.

The program in (3.13) is an example of how a **CFG** (3.12) is typically translated into the logic programming language **Prolog**.

(3.12)
$$\begin{array}{lll} \mathrm{S} & \rightarrow & \mathrm{NP}\ \mathrm{VP} \\ \mathrm{NP} & \rightarrow & \texttt{Julia} \\ \mathrm{VP} & \rightarrow & \texttt{slept} \end{array}$$

(3.13)
```
s(Z) :- np(X), vp(Y), append(X, Y, Z).
np([julia]).
vp([slept]).
```

The **Prolog** program says, briefly, that one can infer `s(Z)` if `Z` is a list that is the concatenation of two lists `X` and `Y`, and `np(X)` and `vp(Y)` are already known. Furthermore, `np([julia])` and `vp([slept])` are facts. So from the facts, `s([julia, slept])` can be inferred. Square brackets are used to construct lists. The predicate “append” is a **Prolog** built-in.

I will use a notation, illustrated in (3.14) and (3.15), for grammars that is half way between the conventions for writing down context-free grammars and **Prolog**—I don’t use the **Prolog** conventions for variables, and polish away the use of **append** by allowing associative concatenation in predicate arguments.

(3.14)
$$\begin{array}{l} \mathrm{S}(xy)\ \texttt{:-}\ \mathrm{NP}(x), \mathrm{VP}(y). \\ \mathrm{NP}(\texttt{Julia}). \\ \mathrm{VP}(\texttt{slept}). \end{array}$$

The **LMG** in (3.16) is a one-to-one equivalent of a 2-**MCFG** for $\texttt{a}^n\texttt{b}^n\texttt{c}^n$. While it

$$\begin{array}{llll} [1] & \mathrm{S} \rightarrow e(\mathrm{C}), & e(\langle n,\ v,\ c\rangle) & = & \langle \texttt{Antje Frank}\ n\ c\rangle \\ \\ [2] & \mathrm{C} \rightarrow f(\mathrm{C}), & f(\langle n,\ v,\ c\rangle) & = & \langle n,\ v,\ v\ \texttt{und}\ c\rangle \\ [3] & \mathrm{C} \rightarrow g(\mathrm{V}), & g(\langle n,\ v\rangle) & = & \langle n,\ v\ \texttt{sah},\ \varepsilon\rangle \\ \\ [4] & \mathrm{V} \rightarrow h(\mathrm{V}), & h(\langle n,\ v\rangle) & = & \langle \texttt{Ute}\ n,\ v\ \texttt{lassen}\rangle \\ [5] & \mathrm{V} \rightarrow \langle \texttt{Julia},\ \texttt{verführen}\rangle & & & \end{array}$$

Figure 3.8: **PMCFG** for partial verb clause conjunctions.

derives a string $\mathtt{a}^n\mathtt{b}^n$ just like the context-free grammar (3.15), a separate component $\mathtt{c}^n$ is generated in parallel.

(3.15) $\mathrm{A}(\mathtt{a}x\mathtt{b})$ **:-** $\mathrm{A}(x)$.
$\mathrm{A}(\varepsilon)$.

(3.16) $\mathrm{A}(\mathtt{a}x\mathtt{b}, \mathtt{c}y)$ **:-** $\mathrm{A}(x, y)$.
$\mathrm{A}(\varepsilon, \varepsilon)$.

The free definite clause style notation of **LMG** allows the grammar writer to do everything allowed in a **PMCFG**, and considerably more.

Formal definition

Literal movement grammars and **PMCFG** are defined in a similar fashion, but in **LMG**, the *yield functions* are integrated into the productions and the *arity function* on nonterminals becomes redundant.

3-18 definition. A (generic) LITERAL MOVEMENT GRAMMAR (**LMG**) is a tuple $G = (N, T, V, S, P)$ where N, T and V are mutually disjoint sets of nonterminal symbols, terminal symbols and VARIABLE SYMBOLS respectively, $S \in N$, and P is a finite set of CLAUSES

$$\phi \;\; \textbf{:-} \;\; \psi_1, \psi_2, \ldots, \psi_m.$$

where $m \geq 0$ and each of $\phi, \psi_1 \ldots \psi_m$ is a PREDICATE

$$A(t_1, \ldots, t_p)$$

where $p \geq 1$, $A \in N$ and $t_i \in (T \cup V)^*$. When $m = 0$, the clause is TERMINAL, and the symbol **:-** is usually left out.

A predicate $A(t_1, \ldots, t_p)$ is SIMPLE if $t_1, \ldots, t_p$ are single variable symbols.

*

An **LMG** clause is INSTANTIATED by substituting a string $w \in T^*$ for each of the variables occurring in the clause. E.g. the rule $\mathrm{S}(xy)$ **:-** $\mathrm{NP}(x), \mathrm{VP}(y)$ is instantiated to rules over terminals only such as

S(`Julia pinched Fred`) **:-** NP(`Julia`), VP(`pinched Fred`).
S(`Julia pinched Fred`) **:-** NP(`Julia pinched`), VP(`Fred`).

3-19 definition: derivational semantics. Let $G = (N, T, V, S, P)$ be an **LMG**. Then G DERIVES instantiated predicates as in the following inductive definition: let w_i, $v_j^i \in T^*$;

Base case[7] If

$$A(w_1,\dots,w_p).$$

is an instantiation of a terminal clause, then $G \vdash A(w_1,\dots,w_p)$.

Inductive step If, $m \geq 1$ and

$$A(w_1,\dots,w_p) \;\;\texttt{:-}\;\; B_1(v^1_1,\dots,v^1_{p_1}),\dots,B_m(v^m_1,\dots,v^m_{p_m}).$$

is an instantiation of a (non-terminal) clause in P, and for each $1 \leq k \leq m$, $G \vdash B_k(v^k_1,\dots,v^k_{p_k})$, then $G \vdash A(w_1,\dots,w_p)$.

Now G is said to RECOGNIZE w if $w = w_1 \cdots w_p$ and $G \vdash S(w_1,\dots,w_p)$.

3-20 remark. This definition of **LMG** does not assign an *arity* to a nonterminal symbol; this is to simplify the definitions, especially in section when looking at fixed point interpretations in chapters **4** and **5**. In principle, a nonterminal can appear with different arities in the same **LMG**.

3-21 examples. Although it is a rather simple example ($m = 1$), let's check how the **LMG** in (3.16) derives `aabbcc`:

$$\begin{array}{llll}
 & G \vdash \mathrm{A}(\varepsilon,\ \varepsilon) & \text{by } \mathrm{A}(\varepsilon,\varepsilon). & \\
\Rightarrow & G \vdash \mathrm{A}(\texttt{ab},\ \texttt{c}) & \text{by } \mathrm{A}(\texttt{a}x\texttt{b},\ \texttt{c}y)\ \texttt{:-}\ \mathrm{A}(x,y). & (x = y = \varepsilon) \\
\Rightarrow & G \vdash \mathrm{A}(\texttt{aabb},\ \texttt{cc}) & \text{by } \mathrm{A}(\texttt{a}x\texttt{b},\ \texttt{c}y)\ \texttt{:-}\ \mathrm{A}(x,y). & (x = \texttt{ab},\ y = \texttt{c})
\end{array}$$

Figures 3.9–3.11 show **LMG** equivalents of an **MHG**, a linear **MCFG** and a **PMCFG**, respectively. The grammar in *figure 3.9* is a bit easier to verify than the **MHG** itself, but the existence of the much simpler example (3.16) shows that the rules allowed in **MHG** make it more cumbersome than a more liberal 2-component **LMG** format.

$$\begin{array}{llll}
[1] & \mathrm{S}(y_1x_1, x_2y_2) & \texttt{:-} & \mathrm{B}(x_1,x_2), \mathrm{T}(y_1,y_2) \\
[2] & \mathrm{S}(\varepsilon,\varepsilon) & & \\
[3] & \mathrm{T}(y_1x_1, x_2y_2) & \texttt{:-} & \mathrm{S}(x_1,x_2), \mathrm{A}(y_1,y_2) \\
[4] & \mathrm{A}(\texttt{a},\texttt{c}) & & \\
[5] & \mathrm{B}(\varepsilon,\texttt{b}) & &
\end{array}$$

Figure 3.9: **LMG** equivalent for the **MHG** in *figure 3.3*.

[7] Note that a more elegant, but less elementary definition would use that fact that the inductive step includes the base case if one puts $m \geq 0$.

[1] $\mathrm{C}^{\mathrm{sub}}(\texttt{dat}\ s\ o\ h\ v)$ **:-** $\mathrm{V}^0(s,o,h,v)$.
[2] $\mathrm{C}^{\text{decl-wh}}(s\ h\ o\ v)$ **:-** $\mathrm{V}^0(s,o,h,v)$.
[3] $\mathrm{C}^{\mathrm{ques}}(h\ s\ o\ v)$ **:-** $\mathrm{V}^0(s,o,h,v)$.

[4] $\mathrm{V}^0(s,\ o,\ h,\ v)$ **:-** $\mathrm{N}^0(s), \mathrm{V}^{\mathrm{I}}(o,h,v)$.
[5] $\mathrm{V}^{\mathrm{I}}(o,\ v,\ \varepsilon)$ **:-** $\mathrm{V}^{\mathrm{T}}(v), \mathrm{N}^0(o)$.
[6] $\mathrm{V}^{\mathrm{I}}(s\ o,\ r,\ h\ v)$ **:-** $\mathrm{V}^{\mathrm{R}}(r), \mathrm{V}^0(s,o,h,v)$.

[7] $\mathrm{N}^0(\texttt{Frank})$.
[7′] $\mathrm{N}^0(\texttt{Julia})$.
[7″] $\mathrm{N}^0(\texttt{koffie})$.
[8] $\mathrm{V}^{\mathrm{I}}(\varepsilon,\ \texttt{zwemmen},\ \varepsilon)$.
[9] $\mathrm{V}^{\mathrm{T}}(\texttt{drinken})$.
[10] $\mathrm{V}^{\mathrm{R}}(\texttt{zag})$.

Figure 3.10: **LMG** equivalent of the linear **MCFG** in *figure 3.4*.

Formal properties and classification

From a series of formal results, it will turn out that **LMG** should be regarded as a framework for the definition of more restricted formalisms, rather than as a concrete tool for language description. The most important such result is that **LMG** in their general form generate any *recursively enumerable* language; preparations for this theorem are made here, while the conclusion is drawn in chapter **5** after some complexity theory has been introduced. A remarkable subclass of **LMG** is that of the SIMPLE **LMG**, which is analogously defined here, and shown to describe precisely the class of tractably recognizable languages in chapter **5**.

The constructions used to prove the first two results already indicate that generic **LMG** is a dramatically powerful formalism.

3-22 proposition. The class **LML** of languages recognized by generic **LMG** is closed under arbitrary homomorphism.

Proof. Let L be an **LML** over T, and h be a homomorphism. Suppose L is recognized by the **LMG** $G = (N, T, V, S, P)$, and assume w.l.o.g. that the start symbol S appears

[1] $\mathrm{S}(xx)$ **:-** $\mathrm{S}(x)$.
[2] $\mathrm{S}(\texttt{a})$.

Figure 3.11: **LMG** equivalent for the **PMCFG** in *figure 3.7*.

only with arity 1. Then construct **LMG** $G' = (N \cup \{\Sigma, H\},\ T \cup h(T),\ V,\ \Sigma,\ P')$, such that $\Sigma, H \notin N$ and

$$\begin{array}{lll} P' & = & P \cup \\ & & \{\ \Sigma(y)\ \mathrel{:-}\ S(x),\ \mathrm{H}(x,\ y).\ , \\ & & \ \ \mathrm{H}(\varepsilon, \varepsilon).\ \} \cup \\ & & \{\ \mathrm{H}(a\,x,\ w\,y)\ \mathrel{:-}\ \mathrm{H}(x,\ y).\ \mid a \in T,\ w = h(a)\ \} \end{array} \tag{3.17}$$

Then G' generates $h(L)$. □

After seeing the **Prolog** construction, one may be tempted to think that the class of languages recognized by **LMG**s whose predicates all occur with arity 1 is the class of context-free grammars. On the contrary:

3-23 definition. An n-**LMG** is an **LMG** in which no nonterminal symbol appears with an arity greater than n.

3-24 proposition. 1–**LML** = **LML**.

Proof. The idea is to replace the commas in **LMG** clauses by terminal symbols. Let $G = (N, T, V, S, P)$ be an **LMG**, let V be finite, and order V as $\{x_1, x_2, \ldots, x_n\}$. Let the symbols ITS and c not be in N, T or V. Then let $G' = (N \cup \{\mathrm{ITS}\},\ T \cup \{\mathsf{c}\},\ V, S, P')$ where

$$\begin{array}{llllll} P' & = & \{\ A(t_1\ \mathsf{c}\ t_2\ \mathsf{c}\ \cdots\ \mathsf{c}\ t_p) & \mathrel{:-} & B_1(s^1_1\ \mathsf{c}\ s^1_2\ \mathsf{c}\ \cdots\ \mathsf{c}\ s^1_{p_1}), \\ & & & & \ldots \\ & & & & B_m(s^m_1\ \mathsf{c}\ s^m_2\ \mathsf{c}\ \cdots\ \mathsf{c}\ s^m_{p_m}), \\ & & & & \mathrm{ITS}(x_1),\ \mathrm{ITS}(x_2),\ \ldots,\ \mathrm{ITS}(x_n). \\ & & \quad \mid P \text{ contains a production} & & \\ & & \quad\ A(t_1,\ t_2,\ \ldots,\ t_p) & \mathrel{:-} & B_1(s^1_1,\ s^1_2,\ \ldots,\ s^1_{p_1}), \\ & & & & \ldots \\ & & & & B_m(s^m_1,\ s^m_2\ \ldots,\ s^m_{p_m})\ \} \cup \\ & & \{\ \mathrm{ITS}(a\,x)\ \mathrel{:-}\ \mathrm{ITS}(x).\ \mid a \in T\ \} \cup & & \\ & & \{\ \mathrm{ITS}(\varepsilon).\ \} & & \end{array} \tag{3.18}$$

Intuitively, the symbol c stands for "comma" and ITS stands for "is a word in T^*", in other words, for "does not contain the comma symbol". Now G' recognizes $\mathcal{L}(G)$. □

So let's proceed to a classification of the **PMCFG** hierarchy, in terms of **LMG**. For a full classification of the different formalisms in their predicate **LMG** representations, some terminology needs to be introduced.[8]

[8] The terms *bottom-up* and *top-down* in definition **3-25** reflect the behaviour in the perspective of derivation trees, where the root (S-) node is the top of the derivation. Although this choice of terminology should certainly not be read as hinting at a relationship to parsing or procedural semantics, the analogy to the top-down/bottom-up terminology is useful, because other options like 'leftward' and 'rightward' give rise to confusion—left and right are reversed in **MCFG** notation.

Formalism	*Increasing conditions on* **LMG** *form*
Generic **LMG**	—
Simple **LMG**	Bottom-up nonerasing, non-combinatorial
(Nonerasing) **PMCFG**	Top-down linear, top-down nonerasing
LCFRS	Bottom-up linear
MHG	Pairs only, restricted operations
CFG	Singletons

Figure 3.12: Classification of tuple grammars in terms of **LMG** clause restrictions.

3-25 classification: clause types. Let R be an **LMG** clause:

$$A(t_1,\ldots,t_p)\ \ \mathbf{:-}\ \ B_1(s^1_1,\ldots,s^1_{p_1}),\ \ldots,\ B_m(s^m_1,\ldots,s^m_{p_m}).$$

then

▷ $R \in \mathcal{BU}\text{-}\mathcal{LIN}$, or R is bottom-up linear if no variable x appears more than once in $t_1,\ldots,t_p$.

▷ $R \in \mathcal{TD}\text{-}\mathcal{LIN}$, or R is top-down linear if no variable x appears more than once in $s^1_1,\ldots,s^m_{p_m}$.

▷ $R \in \mathcal{BU}\text{-}\mathcal{NE}$, or R is bottom-up nonerasing if each variable x occurring in one of the s^j_k also occurs in at least one of the t_i.

▷ $R \in \mathcal{TD}\text{-}\mathcal{NE}$, or R is top-down nonerasing if each variable x occurring in one of the t_i also appears in one of the s^j_k.

▷ $R \in \mathcal{NC}$, or R is non-combinatorial if each of the s^j_k consists of a single variable, *i.e.* all predicates on the RHS are simple.

▷ R is simple if it is in $\mathcal{S} = \mathcal{BU}\text{-}\mathcal{NE} \cap \mathcal{BU}\text{-}\mathcal{LIN} \cap \mathcal{NC}$.

3-26 classification: grammar types. An **LMG** (N,T,S,P) is an $\mathcal{F}$-**LMG** if for all $R \in P, R \in \mathcal{F}$. Furthermore, $\mathcal{F}$-**LML** will denote the class of languages recognized by $\mathcal{F}$-**LMG**.

A $\mathcal{BU}\text{-}\mathcal{LIN}$-**LMG** is called bottom-up linear, a $\mathcal{TD}\text{-}\mathcal{LIN}$-**LMG** is top-down linear, $\mathcal{BU}\text{-}\mathcal{NE}$-**LMG** is bottom-up nonerasing, a $\mathcal{TD}\text{-}\mathcal{NE}$-**LMG** is top-down nonerasing and an $\mathcal{S}$-**LMG** is a simple **LMG**.

3-27 proposition. A language L is described by a **PMCFG** if and only if it is described by a top-down linear, top-down nonerasing, non-combinatorial **LMG**.

Proof. Suppose w.l.o.g. that each nonterminal A in a **LMG** appears with only one arity $a(A)$ (add new names for nonterminals that appear in more than one arity). Then there is a 1–1 correspondence between **LMG** rules and **PMCFG** rules: rule (3.19) is mapped to (3.20) and vice versa. The translation from **PMCFG** to **LMG** is trivial; the reverse translation is valid because of non-combinatoriality, top-down linearity and non-erasingness (otherwise f would not have been a function).

(3.19) $A \rightarrow f(B_1, \dots, B_m)$
where $f(\left\langle x^1_1, \dots, x^1_{p_1} \right\rangle, \dots, \left\langle x^m_1, \dots, x^m_{p_m} \right\rangle) = \langle t_1, \dots, t_p \rangle$

(3.20) $A(t_1, \dots, t_p)$ **:-** $B_1(x^1_1, \dots, x^1_{p_1}), \dots, B_m(x^m_1, \dots, x^m_{p_m})$.

□

The classes **MHG**, linear **MCFG**, *&c.* can now all be classified in terms of clause type classification **3-25**. Such a classification is shown in *figure 3.12*.

The last clause type defined, that of *simple* **LMG**, will play an important rôle in this thesis; simple **LMG** will turn out to generate precisely the languages recognizable in polynomial time, and have some favourable linguistic properties as well.

The key property of simple **LMG** is that, while its power is sufficiently restricted, it is capable of modelling *sharing* of strings generated in different branches of a derivation.

3-28 proposition: intersection. Let $\mathcal{F}$ be a class of **LMG** clauses including $\mathcal{S}$. Then $\mathcal{F}$-**LML** is closed under intersection.

Proof. Let $G_1 = (N_1, T_1, V_1, S_1, P_1)$ and $G_2 = (N_2, T_2, V_2, S_2, P_2)$. Assume w.l.o.g. that $N_1 \cap N_2 = \emptyset$ and neither contains the symbol S; moreover let S_1 and S_2 be used only with arity 1. Let G be grammar $G = (N_1 \cup N_2 \cup \{\mathrm{S}\}, T_1 \cup T_2, V_1 \cup V_2, \mathrm{S}, P_1 \cup P_2 \cup \{R\})$, where R is the clause

(3.21) $\mathrm{S}(x)$ **:-** $S_1(x), S_2(x)$.

("$\mathrm{S}(x)$ can be derived if one can derive both $S_1(x)$ and $S_2(x)$.") Then G generates $\mathcal{L}(G_1) \cap \mathcal{L}(G_2)$. □

3-29 proposition. **PMCFL** is strictly contained in $\mathcal{S}$-**LML**

Proof. (**i**) Let L be described by a **PMCFG**. Then it is also described by a nonerasing **PMCFG**. Translate this grammar to an **LMG** G using proposition **3-27**. The clauses in G are now bottom-up and top-down nonerasing, and top-down linear.

This means that the clauses of G are not necessarily simple, because they may not be bottom-up linear; that is, a variable x may appear on the LHS of a clause more than once. Look at such a clause; an example is (3.22).

(3.22) $\mathrm{A}(x, yy)$ **:-** $\mathrm{B}(x), \mathrm{C}(y)$.

Contrary to **PMCFG**, simple **LMG** allows variables to appear on the *right hand side* of a clause more than once. So clause (3.22) can be replaced with the simple clause (3.23) and the simple clause schema (3.24) to yield a weakly equivalent grammar.

(3.23) $\mathrm{A}(x, yz)$ **:-** $\mathrm{B}(x),\ \mathrm{C}(y),\ \mathrm{Eq}(y, z).$

(3.24) $\mathrm{Eq}(ax, ay)$ **:-** $\mathrm{Eq}(x, y).$ for every $a \in T$
$\mathrm{Eq}(\varepsilon, \varepsilon).$

Repeat this construction until the grammar contains no more non-simple rules. So **PMCFL** is contained in $\mathcal{S}$-**LML**.

(**ii**) **PMCFL** is closed under homomorphism (**3-14**) and membership for **PMCFL** is decidable ([SMFK91] or chapter **5**). Suppose that **PMCFL** = $\mathcal{S}$-**LML**. Since $\mathcal{S}$-**LML** is closed under intersection, this would yield a class containing the context-free languages that is decidable and closed under intersection and homomorphism. This is a contradiction.[9] □

3-30 example: counting and copy ability. Sharing can be used to mimick much of the capacity of arbitrary linear **MCFG** and the reduplication of **PMCFG** to describe simple formal languages. Let $\mathcal{F}$ be a class of **LMG** clauses including $\mathcal{S}$; then 2-$\mathcal{F}$-**LML** is closed under taking n copies for any n, because there is the clause

(3.25) $S_1(x_1x_2\cdots x_n)$ **:-** $S_2(x_1),\ \mathrm{Eq}(x_1, x_2), \mathrm{Eq}(x_2, x_3), \ldots, \mathrm{Eq}(x_{n-1}, x_n).$

Similarly, 2-$\mathcal{S}$-**LMG** can generate the **PMCFL** a^{2^n}. All n-counting languages are in 1-$\mathcal{S}$-**LML**, because they are finite intersections of context-free languages.

3-31 proposition. For each $k \geq 0$, the class of k-$\mathcal{S}$-**LML** is a pre-**AFL** [GGH69], that is, closed under marked product L_1aL_2, marked iteration $(Lc)^+$, inverse homomorphism and intersection with regular sets.

Proof. All except closure under inverse homomorphism are obvious. Unlike in the **AFL** case, closure under h^{-1} does not follow immediately from other closure results (in the case of multiple context-free grammars, this can be taken to follow from substitution by regular sets, *cf.* [Gin75], corollary on p. 76). It must be proved directly by finitely encoding indices into the homomorphic images of leftmost and rightmost terminals of the argument terms. This is laborious, but straightforward. □

3-32 example: simple LMG and partial verb clause conjunctions.
The grammar in *figure 3.13* generates, directly, a fragment of German slightly larger than L in proposition **3-15**; larger because this grammar also generates conjunctions in embedded verbal clauses.

[9] The notion of decidability and the results used here will be discussed in part **II**.

Conclusions to chapter 3

While head grammars were originally introduced to serve an explanatory-linguistic purpose, **MCFG/LCFRS** are not used in the literature for concrete structural descriptions; rather, once a head grammar had been constructed that generated Dutch subordinate clauses in head grammar, the job was considered done. One reason for this lack of interest is that there is no linguistic foundation for the ‘clustering’ analysis of surface structure except the one based on the notion of the position of the head introduced in [Pol84]. Another important reason is that an **MCFG** is not a nice format for writing grammars.

This chapter has tried to fill in this gap, first by showing that linear and parallel **MCFG** can give accounts of important phenomena that cannot be treated in **MHG**; then by introducing the new formalism **LMG** that gives further refined descriptions of the running example of Manaster-Ramer’s partial verb clause conjunctions, and has a more readable form. The notation has not been seen before in the literature, but others have mentioned that they developed it independently (Ristad, p.c.).

The step from multiple context-free grammars to literal movement grammars is straightforward, although the only similar step made in the literature is the introduction of **iLFP** in [Rou88]. The **iLFP** formalism however, discussed at length in chapter **5**, is meant as an intermediate logical language connecting grammar formalisms, finite arithmetic and complexity theory.

LMG can be viewed as the ultimately simplified, grammar-like representation of **iLFP**, although when I introduced **LMG** in [Gro95b] in the obsolete form not discussed in this thesis (but used in the **Prologue**, page 8), I was entirely unaware of

[1] $\mathrm{C}^{\mathrm{sub}}(\texttt{daß}\ o\ v)$:- $\mathrm{V}^{0}(o,v)$.

[2] $\mathrm{V}^{0}(o,\ v_1\ \texttt{und}\ v_2)$:- $\mathrm{V}^{0}(o,v_1),\ \mathrm{V}^{0}(o,v_2)$.

[3] $\mathrm{V}^{0}(s\ o,\ v)$:- $\mathrm{N}^{0}(s), \mathrm{V}^{\mathrm{I}}(o,v)$.
[4] $\mathrm{V}^{\mathrm{I}}(o,\ v)$:- $\mathrm{V}^{\mathrm{T}}(v), \mathrm{N}^{0}(o)$.
[5] $\mathrm{V}^{\mathrm{I}}(o,\ v\ h)$:- $\mathrm{V}^{\mathrm{R}}(h), \mathrm{V}^{0}(o,v)$.

[7] $\mathrm{N}^{0}(\texttt{Frank})$.
[8] $\mathrm{N}^{0}(\texttt{Julia})$.
[9] $\mathrm{N}^{0}(\texttt{Ute})$.
[10] $\mathrm{N}^{0}(\texttt{Antje})$.
[11] $\mathrm{N}^{0}(\texttt{Dieter})$.
[12] $\mathrm{N}^{0}(\texttt{Fred})$.

[13] $\mathrm{V}^{\mathrm{T}}(\texttt{küssen})$.
[14] $\mathrm{V}^{\mathrm{T}}(\texttt{umarmen})$.
[15] $\mathrm{V}^{\mathrm{T}}(\texttt{einladen})$.

[16] $\mathrm{V}^{\mathrm{R}}(\texttt{sah})$.
[17] $\mathrm{V}^{\mathrm{R}}(\texttt{hörte})$.
[18] $\mathrm{V}^{\mathrm{R}}(\texttt{sehen})$.
[19] $\mathrm{V}^{\mathrm{R}}(\texttt{hören})$.
[20] $\mathrm{V}^{\mathrm{R}}(\texttt{lassen})$.

Figure 3.13: Simple **LMG** for partial verb clause conjunctions.

connections to either **MCFG** or **LFP**. The current notation, which is half way between that of **MCFG** and **iLFP**, was introduced in a series of subsequent papers in 1995 and 1996. All material in these papers, except descriptions of Dutch in [Gro95a], and the correspondence between production notation and the current definite clause notation (called **CPG**) in [Gro96] is covered in equivalent or higher depth in this book.

Most of the results presented in this chapter are covered in the literature, but have not been summarized in a single text. New are the idea that **MHG** are not sufficient to treat the Dutch crossed dependencies embedded in full sentences (which is further worked out in chapter **8**), the **LMG** formalism and the resulting syntactical classification of the **PMCFG** hierarchy, and the remarkable observation that the partial verb clause conjunctions, unexpectedly, turn out to be weakly generable by a **PMCFG**.

Remarkable, because it seems intuitively clear that the partial verb clause conjunctions are *not* truly a case of reduplication; the underlying non-terminal structure may be, but the words themselves are certainly not reduplicated to form the conjuncts. It seems, then, that a strong generative capacity argument is needed to prove the inadequacy of **PMCFG** w.r.t. this construction. This is discussed further in chapter **8**.

A note that I put here hesitatingly is that the proof in [SMFK91] that **PMCFG** is a substitution-closed **AFL** is not very explicit, which leads me to have some doubt as to the closure under substitution; the proof claims that closure under substitution simply *follows from the definitions*. This is clearly the case for *linear* **MCFG**, but a parallel production

$$A \rightarrow f(a) \text{ where } f(\langle x \rangle) = \langle xx \rangle$$

will produce a recursive step $w \rightarrow ww$. The standard proof of closure under substitution would produce $\{vv \mid v \in \sigma(w)\}$ which is not the desired set $\sigma(w)\sigma(w) = \{uv \mid u, v \in \sigma(w)\}$. I have not found a way to fix this. This may be due to my own limited imagination, but if the proof cannot be substantiated, this would break the result that **PMCFG** can generate the German partial verb clause conjunctions. I have not found the authors and other experts in the field able or willing to comment on this issue, so it remains open, but I found it nevertheless worthwhile to include in the discussion.

The scope of the examples in this chapter is highly limited. It is often thought that an argument based on *co-ordination* as the running example in this chapter does not give much insight in the nature of more basic properties of syntax, because it has a number of 'mathematical' or 'logical' aspects that may not be part of the primary linguistic interface. A larger overview of phenomena and generative capacity arguments is given in chapter **8**. Chapter **5** will give a detailed treatment of the computational properties of simple **LMG** and another restricted version called *bounded* **LMG**.

An interesting open question is whether there is a number n such that n-$\mathcal{S}$-**LML** include, for example, all **MCFL**, all **PMCFL**. It seems reasonable to assume that this is indeed a hard question; see the last paragraph on page 118.

Part II

Computational Tractability

Chapter 4

Complexity and sentence indices

COMPUTATIONAL TRACTABILITY is an important motivation for several issues in the design of the formalisms of the previous chapters, and for the principle-based formulations still to come in part **III**.

Tractability can be divided into **(i)** a formal requirement of *polynomial complexity* for algorithms that recognize a language or construct a structural representation of a sentence and **(ii)** efficient and straightforward implementation on practical real-world computer systems. The first three chapters of part **II** will go into the different aspects of formal tractability in connection with the formalisms from part **I**. Chapter **7** will look at aspects of practical implementation.

For most of this thesis, a very simple approach to complexity is sufficient, and I will concentrate on developing good intuitions for making a "quick complexity diagnosis" rather than going through the formal material, which is of less immediate benefit to the every day (computational or generative) linguist. Nonetheless, to support these quick diagnoses, an amount of mathematical labour is unavoidable.

This chapter is a basic introduction to the notions in complexity theory I will be using; the fast reader can proceed straight to example **4-11**. Readers less experienced in complexity theory should read this chapter entirely, and may decide to skip large parts of sections **4.4** and **5.1**.

4.1 Definitions

I will first introduce a number of concepts that will need some form of definition but will be used rather loosely in what is to follow. As is usual in formal theory of computation, when talking about complexity, *interaction* is disregarded.

4-1 definition: device. In the following, a DECISION DEVICE M is an abstract machine that can perform, at any time, the following series of actions:

1. Receive an INPUT STRING w in an INPUT LANGUAGE $L^{\mathbf{in}}$.
2. Perform a computation that takes an integer number t, or infinitely many units of TIME; in the latter case it is said that the device *does not terminate* on input w. If the machine terminates, it has used an integer number s of units of SPACE.
3. On termination of the computation, output **accept** or **reject**.

Furthermore, the steps taken in the computation, and consequently the time and space requirements of the computation, and the output string, are uniquely determined by the input string.

The input language is usually taken to be T^* for an INPUT ALPHABET T. The set of input strings upon which computation terminates and outputs **accept** is the language $\mathcal{L}(M)$ recognized by the device.

A decision device can be extended to a PROCEDURAL DEVICE when the contents of the storage at the end of the computation are taken to represent an OUTPUT STRING encoded in some OUTPUT ALPHABET $L^{\mathbf{out}}$. In the case of a pure decision device, the output language is $\{\mathbf{accept}, \mathbf{reject}\}$.

*

The notion of a device is intended to capture that of a COMPUTER (a class) equipped with a PROGRAM (an algorithm). The notion of a *problem* is introduced below because in general one wants to talk about the difficulty of a task regardless of what type of computer system is performing it. The two models further specifying the notion of a computer I will use are BOUNDED ACTIVITY MACHINES, or TURING MACHINES, and RANDOM ACCESS MACHINES. Turing machines are popular in the study of large abstract classes of problems, whereas random access machines are closer to physical computers and as such give a simple model for more precise analysis of the difficulty of practical problems.

The first concern in the design of computer algorithms is to know whether a problem can be solved on a certain class of devices. This is formalized in the following definitions.

4-2 definitions: algorithm, problem. Devices can be divided into CLASSES in two typical ways.

(**i**) by further specifying the nature of the computation (step 2); an ALGORITHM for machines of a class C is a specification of a device in that class.

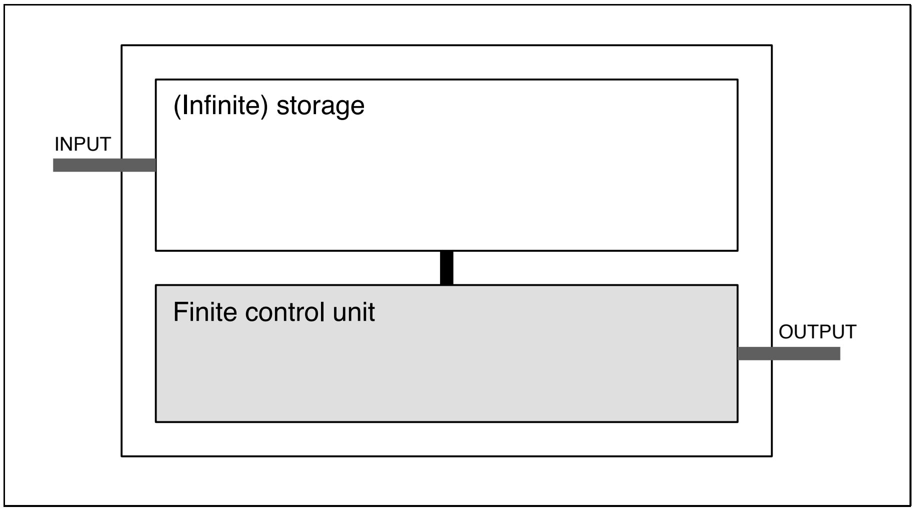


Figure 4.1: The architecture of a decision device.

(**ii**) A PROBLEM is a function $f: L^{\mathbf{in}} \rightarrow L^{\mathbf{out}}$. A problem defines a class C_f of devices such that every device in C_f terminates, and produces output $f(w)$ on input w. C_f is the class of devices that SOLVES f. A problem for $L^{\mathbf{out}} = \{\mathbf{accept}, \mathbf{reject}\}$ is called a DECISION PROBLEM and specifies a sublanguage of $L^{\mathbf{in}}$.

4-3 definition: decidability and enumerability. A problem is DECIDABLE for a class $\mathcal{D}$ of devices if $\mathcal{D} \cap \mathcal{C}_f$ is not empty. A decision problem is ENUMERABLE for a class $\mathcal{D}$ if there is a device in $\mathcal{D}$ that terminates with output **accept** on input w such that $f(w) = \mathbf{accept}$, and does not terminate or outputs **reject** otherwise.

*

The problems I will look at in this thesis are the following.

4-4 definition: recognition, parsing.

A device solving the UNIVERSAL RECOGNITION PROBLEM for a grammatical formalism $\mathcal{F}$ is a decision device which given a grammar $G \in \mathcal{F}$ and a string w, outputs **yes** if G recognizes w, **no** otherwise.

A device solving the FIXED RECOGNITION PROBLEM for a language L is a decision device which given a string w, outputs **yes** if $w \in L$, **no** otherwise.

A UNIVERSAL PARSER is a device that given a grammar G and a string w, outputs all structural analyses G assigns to w.

A FIXED PARSER for a grammar G is a device that given a string w, outputs all structural analyses G assigns to w.

*

If a problem is computable, the *intrinsic difficulty* of a problem, *i.e.* the maximum possible efficiency in solving a problem, will be assessed in terms of the time and space it takes to compute a solution. Time and space are *functions* of the input string. It is customary to try to express these only in terms of the *length* of the input string; furthermore, one is only interested in the rough growth behaviour of time and space consumption. This is usually done in LANDAU’S ORDER NOTATION.

4-5 definition: order notation. Let f and g be functions from nonnegative integers to nonnegative integers. Then f is said to be $\mathcal{O}(g)$ if there are constants c_1 and c_2 such that for every $k > c_1, f(k) \leq c_2 g(k)$.

*

The complexity statement (1.35) repeated here as (4.1) hence says that given a context-free language L there are a device A and constants $c_1,\ c_2$ such that given a string w of sufficient length n, device A decides whether $w \in L$ in time $t < c_1 n^3$ and using no more storage than $c_2 n^2$ (measured in some elementary unit, say bytes).[1]

(4.1) Fixed recognition for context-free languages can be performed in $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space

The statement

(4.2) Fixed recognition for context-free languages has $\mathcal{O}(n^3)$ time complexity and $\mathcal{O}(n^2)$ space complexity

adds to the previous that for any device A, there are strings w_1, w_2 and constants $c_1,\ c_2$ such that decision by A whether $w_1 \in L$ takes at least $c_1 n^3$ time, and decision by A whether $w_2 \in L$ requires at least $c_2 n^2$ units of computer storage.

In these statements (4.1) and (4.2) one further refinement is left out, that is relevant only in some cases: the statements do not limit the class of devices they are talking about. Of course one is generally talking about a class of devices that has similar characteristics to everyday computers, but the required characteristics vary upon what one is looking for; this will be explained in the next few sections.

An important class of decision problems is **PTIME** or simply **P** that contains those problems that can be solved in POLYNOMIAL TIME on a class of machines representative for real-world computers (BOUNDED ACTIVITY MACHINES, defined in section **4.3**). That is, given a problem $p \in$ **PTIME**, there are a device A and integer numbers c, d such that A solves p on any input string of length n in no more than cn^d time.

4-6 definition. Let $\mathcal{C}$ be a class of devices (say that of random access machines or Turing machines, defined later).

[1] The constant c_1 in the definition of order notation is of minor importance, and can be set to the highest number for which g is zero; this is usually 1.

PTIME-$\mathcal{C}$ is the class of problems solvable by a device in $\mathcal{C}$ in $\mathcal{O}(n^d)$ time for an integer degree d.

EXPTIME-$\mathcal{C}$ is the class of problems solvable by a device in $\mathcal{C}$ in time $\mathcal{O}(2^{cn})$ for some c.

LOGSPACE-$\mathcal{C}$ is the class of problems solvable by a device in $\mathcal{C}$ in $\mathcal{O}(\log\ n)$ space.

r.e. is the class of problems enumerable on a Turing machine.

*

When a problem is called (IN-)TRACTABLE, it is usually meant that it is (not) solvable in polynomial time; but use of the term tractable often implies the wish that the degree d of the bounding polynomial is low—under 10, say.

Polynomial time complexity and tractability are an important motivation in the design decisions underlying the **LMG** formalism and the empirical study in part **III**. Exponential time will enter the discussion in section **5.4** on BOUNDED **LMG** and the **cLFP** calculus and chapter **6** on the complexity of extraposition grammar. Nondeterminism and alternation will be defined in section **4.4**.[2]

[2] The reason **NPTIME** is not defined here, is that I will view nondeterministic and alternating Turing machines as separate classes of machines: **NPTIME**-**TM** = **PTIME**-**NTM**.

4.2 Index LMG and the predicate descent algorithm

Before I introduce informal definitions of the machine types, it is good to have a closer look at the sort of problems and solutions that are to be computed. I will now show how one moves from clauses about strings to clauses about integer positions in strings, then how to translate those clauses into simple procedures in imperative programming style, which I will call PREDICATE DESCENT ALGORITHMS; with a note on formalization of these intuitive algorithms.

The predicate descent approach is somewhat different than what is common in Computer Science, where much emphasis is laid on going through the input string from left to right. From a formal language theoretic point of view, there is usually no difference between left and right, and predicate descent algorithms are thus closer to the original formulation of the grammar than conventional parsing or recognition algorithms, and easier to grasp. They can be more easily extended to procedures for extensions of context-free grammar, while preserving the general results on the complexity of context-free recognition and parsing.

Index LMG

For the purpose of designing these concrete recognition algorithms, it is useful to introduce a form of **LMG** that does not have concatenation, but manipulates INDICES in the input string.

4-7 definition: index LMG. An INDEX LITERAL MOVEMENT GRAMMAR (**iLMG**) is a tuple $G = (N, T, V, S, P)$ where N and V are disjoint finite sets of nonterminal symbols and variable symbols respectively, and $T \subseteq N$ is a set of terminal symbols. $S \in N$ is the start symbol, and P is a finite set of clauses

$$\phi \;\; \texttt{:-} \;\; \psi_1, \psi_2, \ldots, \psi_m.$$

where $m \geq 0$ and each of $\phi, \psi_1 \ldots \psi_m$ is a simple predicate

$$A(x_1, \ldots, x_p)$$

where $p \geq 1, A \in N$ and $x_i \in V$. The $\texttt{:-}$ symbol is left out symbol when $m = 0$. The nonterminal on the LHS of the clause (in ϕ) must not be in T.

An ITEM is a predicate with integer arguments instead of variables:

$$A(i_1, \ldots, i_p)$$

A clause $R \in P$ is INSTANTIATED into an item by substituting for each variable an integer between 0 and n.

An **iLMG** is interpreted as follows. Let an input string $w = a_1 \cdots a_n \in T^*$ be given. Then for each $1 \leq i \leq n$ the formula $G, w \vdash a_i(i-1, i)$ is true.

For each instantiated clause

$$\phi \;\; \texttt{:-} \;\; \psi_1, \psi_2, \ldots, \psi_m.$$

if all of $G, w \vdash \psi_1,\; G, w \vdash \psi_2, \ldots, G, w \vdash \psi_m$ are true, then it can be concluded that $G, w \vdash \phi$ is true.

The grammar G now recognizes a string w when $G, w \vdash S(0, n)$.

*

In chapter **5**, proposition **5-6** it will be proved that every simple **LMG** has a weakly equivalent **iLMG**. For now, it will suffice to illustrate how a **CFG** can be translated to an **iLMG**, taking the weak equivalence for granted.

4-8 example. The **CFG** of example **1-2** is equivalent to the **iLMG** in *figure 4.2*.

[1]	$\mathrm{C}^{\mathrm{sub}}(i,k)$	`:-`	$\mathrm{C}(i,j),\ \mathrm{V}^0(j,k).$
[2]	$\mathrm{V}^0(i,k)$	`:-`	$\mathrm{N}^0(i,j),\ \mathrm{V}^{\mathrm{I}}(j,k).$
[3]	$\mathrm{V}^{\mathrm{I}}(i,k)$	`:-`	$\mathrm{V}^{\mathrm{T}}(i,j),\ \mathrm{N}^0(j,k).$
[4]	$\mathrm{V}^{\mathrm{I}}(i,k)$	`:-`	$\mathrm{V}^{\mathrm{R}}(i,j),\ \mathrm{V}^0(j,k).$
[5]	$\mathrm{V}^{\mathrm{I}}(i,j)$	`:-`	$\mathtt{swim}(i,j).$
[6]	$\mathrm{V}^{\mathrm{T}}(i,j)$	`:-`	$\mathtt{drank}(i,j).$
[7]	$\mathrm{V}^{\mathrm{T}}(i,j)$	`:-`	$\mathtt{drink}(i,j).$
[8]	$\mathrm{V}^{\mathrm{R}}(i,j)$	`:-`	$\mathtt{saw}(i,j).$
[9]	$\mathrm{V}^{\mathrm{R}}(i,j)$	`:-`	$\mathtt{see}(i,j).$
[10]	$\mathrm{V}^{\mathrm{R}}(i,j)$	`:-`	$\mathtt{hear}(i,j).$
[11]	$\mathrm{V}^{\mathrm{R}}(i,j)$	`:-`	$\mathtt{help}(i,j).$
[12]	$\mathrm{N}^0(i,j)$	`:-`	$\mathtt{Frank}(i,j).$
[13]	$\mathrm{N}^0(i,j)$	`:-`	$\mathtt{Julia}(i,j).$
[14]	$\mathrm{N}^0(i,j)$	`:-`	$\mathtt{Fred}(i,j).$
[15]	$\mathrm{N}^0(i,j)$	`:-`	$\mathtt{coffee}(i,j).$
[16]	$\mathrm{C}(i,j)$	`:-`	$\mathtt{that}(i,j).$

Figure 4.2: **iLMG** for the English **CFG** on page 15.

Predicate descent algorithms

The **iLMG** representation of a grammar is a good starting point for writing a straightforward computer algorithm performing string recognition. The style of the MEMOING PREDICATE-DESCENT ALGORITHMS used in this thesis is largely due to LEERMAKERS [Lee93], but I replaced the left-to-right analysis by an undirectional approach that captures, in a general fashion, the usual informal complexity reasoning method based on counting the number of sentence indices that are involved in checking a clause of the grammar.

Given an **iLMG** $G = (N, T, V, S, P)$ (to stay with the case of the previous example, let's say corresponding to a context-free grammar in Chomsky normal form), a recognition algorithm can be constructed as follows.

Construct so-called MEMO TABLES. These are arrays in computer memory that contain for each possible ITEM, that is an instantiated **iLMG** predicate ϕ, a cell that can contain the values **true**, **false** or **unknown**. All entries in the memo table are initialized to the value **unknown**.

There is a function for each nonterminal symbol, whose results, after being computed, are stored in the memo table. To prove an item $A(i, k)$ the function for nonterminal symbol A is called with i and k as arguments, which will try each clause for A, and then try all instantiations of a possible additional variable j appearing on the RHS of a clause. While a nonterminal is being checked, it is marked **false** in the memo table, so that indefinite recursion is avoided—after all, if there is a cyclic derivation of an item, then there is also an acyclic derivation.

The function for the nonterminal V^0 of the English **CFG** is displayed in *figure 4.3*. Given a string w of length n, the algorithm is invoked as $\mathrm{C}^{\mathrm{sub}}(0, n)$. Because every combination of a function and two integer arguments (every item) is memoized after it is computed, each item needs to be computed only once.

Complexity analysis. In this context-free case, there are in total $\mathcal{O}(|N|n^2)$ items, so this is the space complexity of the algorithm. Because the grammar has at most two nonterminals on the RHS of its rules, each function has at most one loop ranging at most 0 to n. So each function call will consume $\mathcal{O}(n)$ time. The number of calls made is proportional to the number of items; therefore under the assumption that memoing is an elementary step, the time complexity of the algorithm is $\mathcal{O}(|N|n^3)$.

The general case. From an arbitrary **iLMG**, a recognition algorithm can be constructed analogously. If the arity of the predicate on the LHS of a clause R in the **iLMG** is p_R, and the total number of variables used in the clause is r_R, then there are n^{p_R} items that trigger the clause. For each of these items, there are $q_R = r_R - p_R$ loops ranging at most $0 \cdots n$, leading to a time complexity of $\mathcal{O}(|N|n^{r_R})$, where R is the clause in the grammar with the highest value for r_R. The total space used is $\mathcal{O}(|N|n^{p_R})$ where R is the clause with the highest arity predicate on its LHS.

In section **5.2**, the values of p, q and r are calculated for a number of concrete formalisms and grammars in the **LMG** hierarchy mentioned in chapter **3**.

```
$V^0(i,k)$:
  if memo table entry for $V^0(i,k) \neq$ **unknown**
  then
    return memoed value
  else
    memo $V^0(i,k)$ as **False**

    loop $j = i \cdots k$
      if $N^0(i,j) =$ **True** and $V^I(j,k) =$ **True**
      then
        write $V^0(i,k) =$ **True** into memo table
        return **True**

    return **False**
```

Figure 4.3: Predicate descent procedure for a context-free nonterminal.

Random access machines

Although the illustration of the predicate descent algorithms is of an informal nature, it is not extremely laborious to formalize it; the bottleneck in the reasoning is the phrase "under the assumption that memoing is an elementary step", which is to say: a value at an arbitrary place in computer memory can be looked up in $\mathcal{O}(1)$ time, that is, in a fixed amount of time independent of how much storage is used. Provided that the total amount of computer memory used is within a computer's RANDOM ACCESS MEMORY, this is indeed the case.

This is formalized in the following definition; see [HS74] for a more detailed treatment.

4-9 deÆnition. A RANDOM ACCESS MACHINE (**RAM**) is a device whose computational unit has the following structure: its storage consists of an unbounded array of REGISTERS R_i indexed with an integer $i \geq 0$, and each register contains, at any step in the computation, any nonnegative integer.

The input language of a **RAM** is $\mathbf{N}^*$ where $\mathbf{N}$ is the set of nonnegative integers. In linguistic practice, one takes a terminal alphabet T and maps each terminal $a \in T$ to an integer number.

On receiving its input, the **RAM** writes the input string into its first n registers; the other registers are set to zero.

Each step in the computation is the performance of an INSTRUCTION. The **RAM** has a PROGRAM P consisting of a finite number n of instructions, labelled $I_1, \ldots, I_n$;

the machine starts at instruction I_1. The following instructions are allowed:

copy 0 **to** R_j
add 1 **to** R_j
copy R_{R_i} **to** R_{R_j}
if-zero R_i **then goto instruction** j
accept
reject

In all except a successful **if-zero**. . . **goto** case, or an **accept** or **reject** instruction, after performing the instruction, the computation will continue at the next instruction.

The output language of a **RAM** is {**accept**, **reject**}. The computation ends at an **accept** or **reject** instruction.

An **M-RAM** has the additional instructions

add R_i **to** R_j
multiply R_j **by** R_i

A BOUNDED (**M**-)**RAM** is a (**M**-)**RAM** with a fixed number **memtop**; the contents of all registers, memory cells, and addresses during the computation are required to be less than **memtop**. Arithmetic operations are performed modulo **memtop**.

A bounded **M**-random access machine is sufficiently close to everyday computers[3] to provide a good basis for reasoning about the practical value of an algorithm. The context-free recognition algorithm sketched here can be feasibly implemented on such a bounded machine, if **memtop** is set to an acceptable value (a few millions). Nevertheless, even then the algorithm will be able to recognize (not even to parse) strings up to a length of at most a few thousand terminals on a end-20th century state-of-the-art computer. For computer languages, algorithms exist that perform much better than that, but their complexity analysis is more informal and often depends on an average-case situation.

Another model of computation used frequently is a TURING MACHINE, which does *not* have elementary random access; these will be discussed now.

[3] An everyday computer being one with a multiplication operation that takes $\mathcal{O}(1)$ time. Of course, when looking at a bounded **RAM** model, order functions become formally ill-defined and can only be used for informal reasoning.

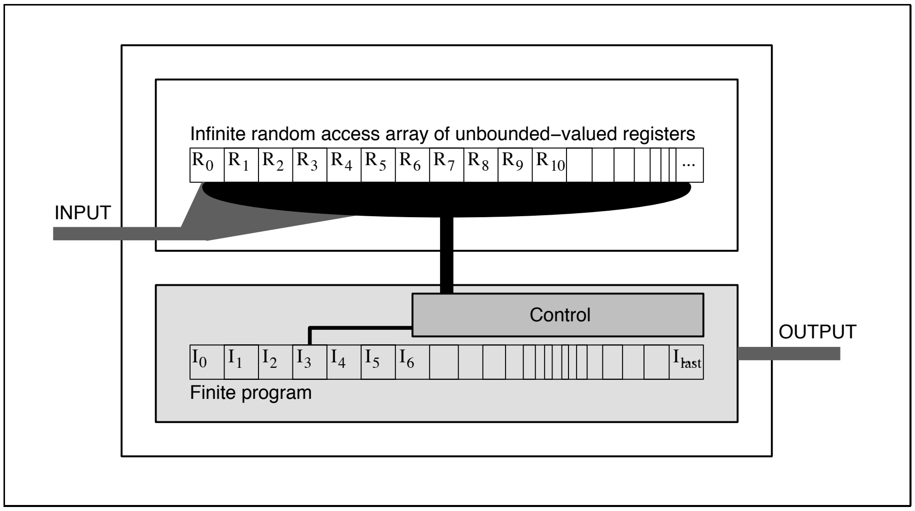

Infinite random access array of unbounded−valued registers
R_0 R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8 R_9 R_{10} ...
INPUT
Control
OUTPUT
I_0 I_1 I_2 I_3 I_4 I_5 I_6 I_{last}
Finite program

Figure 4.4: The architecture of a random access machine.

4.3 Deterministic bounded activity machines

Where random access machines are a formal model close to computers in the real world, formal language theory prefers to talk about TURING MACHINES or BOUNDED ACTIVITY MACHINES as a model of general *symbolic* computation. The reader who is interested only in informal complexity reasoning may skip the discussions of Turing machine implementations, and concentrate on the predicate descent variants.

The storage of a bounded activity machine consists of cells each containing one of finitely many symbols; the cells are ordered and cannot be accessed immediately; instead, the machine, like a tape recorder, has a HEAD which moves over to the cells it wants to read from or write to. The following definition is sketchy; a formal definition of Turing machines will be given in section **4.4**.

4-10 deÆnition. A (deterministic) BOUNDED ACTIVITY MACHINE (**BAM**) is a decision device with the following refinements: its storage consists of a finite number of TAPES t_i, $1 \leq i \leq k$ of dimension d_i, each consisting of CELLS indexed by a d_i-tuple of integers. The device has a HEAD on each tape, which is, at each step in the computation, above one of the cells of the tape. The control unit is, at each step of the computation, in a STATE q. The finite set of possible states Q includes special states q_0, $q_{\mathbf{accept}}$ and $q_{\mathbf{reject}}$

At each step of the computation, the control unit decides, based uniquely on its state q and the contents of the cells under its heads, to take one or more actions of the following types:

▷ write a symbol to one of the tapes

▷ move one of the heads over 1 cell position in any direction

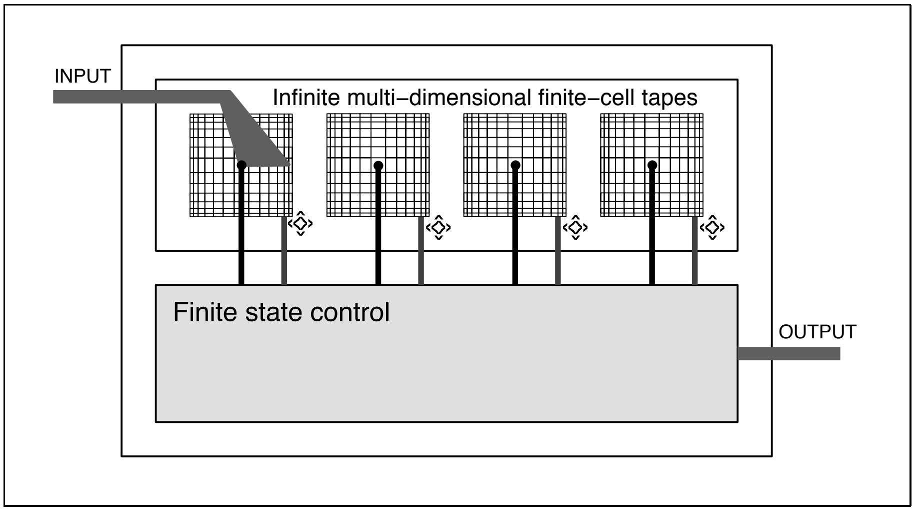


Figure 4.5: The architecture of a bounded activity machine.

▹ go to a different state

The computation begins in the INITIAL STATE q_0, and ends when the machine enters one of the states $q_{\mathbf{accept}}$ or $q_{\mathbf{reject}}$.

A TURING MACHINE is a bounded activity machine whose tapes are all of dimension $d_i = 1$.

*

It takes a **BAM** time to move through its storage; however, this time is no more than a polynomial function of the size of its input. Therefore **PTIME**-**BAM** is the same class of problems as **PTIME**-**RAM** [HS74]. For concrete problems with an exact polynomial time bound such as $\mathcal{O}(n^6)$, this is not a trivial matter.

An example illustrating the nontriviality is that EARLEY's well known algorithm [Ear70] for context-free recognition in $\mathcal{O}(n^3)$ time on a **RAM** can be constructed on a **BAM** with a two-dimensional tape, but on a standard Turing machine one has not been able to achieve a better time complexity than $\mathcal{O}(n^4)$.

Henceforth I shall be interested, as most of the literature, in time complexity on machines with any number k of 1-dimensional tapes—*i.e.* on regular Turing machines. It is a known result that a Turing machine with an arbitrary number k of tapes can be simulated on one with only 2 tapes, where the time consumption grows logarithmically; for a polynomial-time algorithm, this means that the time will increase only by a constant factor.

To illustrate deterministic Turing machines, I now present a sketch of a **CKY** recognizer for **MHG** (see section **3.2**). The informal predicate descent algorithm will turn out to be constructible on a Turing machine preserving the time complexity of its **RAM** equivalent; this works because in this case one can find a suitable *ordering* of the items in the storage, as is done by YOUNGER for the COCKE-KASAMI-YOUNGER (**CKY**) algorithm [You67]. The algorithm works directly on a deterministic 1-dimensional, k-tape Turing machine, where k is at least 4, and has the $\mathcal{O}(n^6)$ time complexity that is currently the best known upper time bound to **MHG** recognition.

The **CKY** algorithm for context-free grammars obtains an $\mathcal{O}(n^3)$ time complexity by looking at grammars in Chomsky normal form, and ordering the recognized "items" (the same ones as in the predicate descent algorithm) by their length as a substring of the input. This ordering prevents the Turing machine from excessively moving around, which would lead to a higher time complexity than a corresponding **RAM** algorithm.

4-11 example: Turing recognizer for MHG. An **MHG** derives pairs of strings; in a derivation of an input string w, arguments to instantiated predicates are always substrings of w, and a predicate $A(u, v) = A(l_1, r_1,\ l_2, r_2)$ consistently satisfies $l_1 \leq r_1 \leq l_2 \leq r_2$.

This means that items can alternatively be represented as (t, A, f, l, r) where t is the total length and f the length of the first component:

$$\begin{aligned} l &= l_1 \\ r &= r_2 \\ t &= (r_1 - l_1) + (r_2 - l_2) \\ f &= (r_1 - l_1) \end{aligned} \tag{4.3}$$

The head grammars are assumed to have terminal rules of the form $A(a, \varepsilon)$ or $A(\varepsilon, a)$ only, *i.e.* ones that do not produce pairs of strings with a total length of zero. Any **MHG** can be normalized to such a grammar plus a possible rule $S \rightarrow \langle \varepsilon, \varepsilon \rangle$.

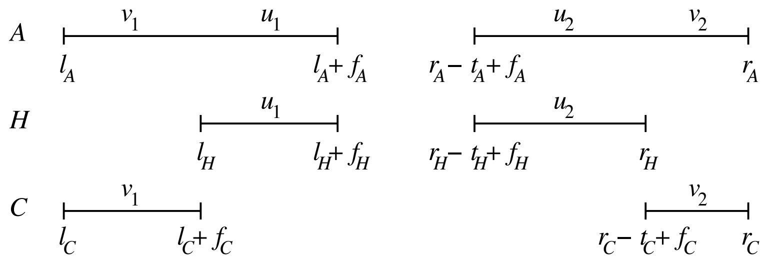
Order the nonterminal alphabet $N = \{A_1, A_2, \ldots, A_{|N|}\}$. The Turing recognizer has three tapes, on each of which the items are layed out from left to right, $(t, \ldots)$ before $(t+1, \ldots)$; $(t, A_k, \ldots)$ before $(t, A_{k+1}, \ldots)$; $(t, A_k, f, \ldots)$ before $(t, A_k, f+1, \ldots)$, *&c.* For each item, there is a single cell containing the value 0 (not recognized) or 1 (recognized). The total amount of tape space used is proportional to the number of items, *i.e.* n^4; a number of additional tapes may be used for storing counters ranging from 0 to n.

The algorithm proceeds in n PHASES; it checks the items of total length 1 first (these are terminal productions), then those of length 2, using the results of the items of length 1, the ones of length k using the results for the items of length shorter than k, and so forth until length n. The head of the first tape moves only to the right in single steps, and writes the results of combining items of shorter length read from the second and third tape. After each phase, the newly written contents of the first tape are copied to the second and the third tape (alternatively, one can think of the machine as having three heads on a single tape).

In each phase, the machine runs through an $\mathcal{O}(n^5)$ loop for each nonterminal A and each clause for A. To describe what is done in this loop, the three types of rules must be translated to the new representation. The wrapping rule $A \rightarrow \textit{head-wrap}(H, C)$ is the most fun example; the corresponding **TM** loop is shown in *figure 4.6.*

(4.4) $A(v_1 u_1,\ u_2 v_2)$ **:-** $H(u_1, u_2),\ C(v_1, v_2).$

(4.5)



(4.6) $(t_H + t_C,\ A,\ f_H + f_C,\ l_C,\ r_C)$ **:-** $(t_H, H, f_H, l_H, r_H), (t_C, C, f_C, l_C, r_C).$

```
{ Tape 1 (write) head is at $(t_A, A, 0, 0, 0)$. }
move head 2 to $t_H = 1$; head 3 to $t_C = t_A - 1$                                  $[1 \cdot n^4]$
while $t_H < t_A$, i.e. $t_C > 0$
  move head 1 to $f_A = 0$                                                            $[n \cdot n^3]$
  while $f_A \leq t_A$
    move head 2 to $f_H = 0$; head 3 to $f_C = f_A$                                   $[n^2 \cdot n^3]$
    while $f_H \leq f_A$, i.e. $f_C \geq 0$
      move heads 1, 3 to $l_A = l_C = 0$                                              $[n^3 \cdot n^2]$
           head 2 to $l_H = n - t_A + f_C$                                            $[n^3 \cdot n^2]$
      while $l_A = l_C \leq n - t_A$, i.e. $l_H \leq f_C$
        move heads 1, 3 to $r_A = r_C = n$                                            $[n^4 \cdot n]$
             head 2 to $r_H = n - t_C + f_C$                                          $[n^4 \cdot n]$
        while $r_A = r_C \geq l_A + t_A$, i.e. $r_C \geq l_A + t_A - t_C + f_C$
          copy boolean and of values on tapes 2 and 3 to tape 1                       $[n^5 \cdot 1]$
          move head 1 to $--r_A$; head 2 to $--r_H$; head 3 to $--r_C$
        move head 1 to $++l_A$; head 2 to $++l_H$; head 3 to $++l_C$
      move head 2 to $++f_H$; head 3 to $--f_C$
    move head 1 to $++f_A$;
  move head 2 to $++t_H$; head 3 to $--t_C$
copy values written on tape 1 to tapes 2 and 3                                        $[1 \cdot (n^4 + n^3)]$
```

Figure 4.6: Procedural description of actions for $A \rightarrow$ *head-wrap*(H, C).

Such induction on the total length of the items can be carried out for any of the tuple-based formalisms that are bottom-up nonerasing and top-down linear (*i.e.* **PMCFG**, see definition **3-26**). The length of a tuple of strings is then simply the sum of the lengths of the substrings; the equivalent of the Chomsky Normal Form reduction used in the construction for modified head grammar can be replaced with the one in [SMFK91].

4.4 Alternation and LFP calculi

This section shows that the auxiliary grammar system **iLMG** generates precisely the languages recognizable in deterministic polynomial time, by establishing a correspondence between **iLMG** and ROUNDS' logical system **iLFP**, which itself is related to logarithmic space-bounded ALTERNATING TURING MACHINES in [Rou88].

This construction serves to shed light on the historical background of what is done in the next chapter for simple **LMG**. I will first define **ATM**; then **iLFP**, and then sketch the link between **iLMG** and **iLFP**.

$$\begin{tikzcd}
\textbf{iLMG} \arrow[r, shift left, "\textbf{4-19}"] & \textbf{iLFP} \arrow[l, shift left, "\textbf{4-20}"] \arrow[r, shift left, "\textbf{4-14}"] & \textbf{LOGSPACE-ATM} \arrow[l, shift left, "\textbf{4-14}"]
\end{tikzcd}$$

Figure 4.7: Results in this section.

Alternation

Bounded activity machines were defined only informally in the previous section. The following definition of alternating Turing machines also serves as a formal definition of the 1-dimensional variant of a **BAM**, *i.e.*, a deterministic Turing machine. It is a slight generalization of the definition in [CKS81]: I make no distinction between read-write tapes and input tapes.

Informally, a NONDETERMINISTIC Turing machine is a machine that can be used to *try* unboundedly many computations at the same time. An ALTERNATING Turing machine can, in addition to nondeterministically checking whether *one of* many continuations of a computation succeeds, also be asked to check whether *all of* a number of continuations succeed; another word for this is PARALLELISM.

4-12 deÆnition. An ALTERNATING TURING MACHINE (**ATM**) is a tuple $M = (k, Q, \Sigma, \delta, q_0, g)$, where

k is the number of tapes;

Q is a finite set of STATES including **accept** and **reject**;

Σ is a finite TAPE ALPHABET consisting of an input alphabet T plus the symbols \$ (end marker) and # (blank symbol);

$\delta \subseteq (Q \times \Sigma^k) \times (Q \times (\Sigma - \{\#\})^k \times \{-1, 0, +1\}^k)$ is the NEXT MOVE RELATION;

$g : Q \to \{\wedge, \vee\}$ designates each state UNIVERSAL or EXISTENTIAL.

The next move relation δ should be understood as follows: when the machine's state is q_1 the symbols under the tape heads are $(a_1, \ldots, a_k)$, and delta contains an element

$$d = ((q_1, (a_1, \ldots, a_k)), (q_2, (b_1, \ldots, b_k), (m_1, \ldots, m_2))),$$

then the machine *can*, when $g(q) = \vee$, or *must*, when $g(q) = \wedge$, make the following step: the symbols $b_1, \ldots, b_k$ are written to the tapes, the machine moves the heads left, when $m_i = -1$, right, when $m_i = +1$, or not at all, when $m_i = 0$, and finally enters state q_2. The machine may be thought of as executing any number of threads in parallel.

A CONFIGURATION or INSTANTANEOUS DESCRIPTION of the machine is an element of the set

$$C_M = Q \times ((\Sigma - \{\#\})^*)^k \times \mathbf{N}^k$$

representing the machine's state, the contents of the k tapes, and the positions of the tape heads.

The machine has k tapes which each contain the input string $w \in T^*$ in the initial configuration (heads at position 0)

$$\sigma_M(w) = (q_0, \underbrace{w, \ldots, w}_{k}, \underbrace{0, \ldots, 0}_{k})$$

The SUCCESSOR RELATION $\vdash$, derived immediately from δ, defines how configurations change when a move is made; given a configuration α, there may be several SUCCESSOR CONFIGURATIONS β such that $\alpha \vdash \beta$. Formally, $\alpha \vdash \beta$ when

$$\begin{array}{llll} (4.7) & \alpha & = & (q_1, a_1^1 \cdots a_{h_1}^1 \cdots a_{n_1}^1,\ \ldots,\ a_1^k \cdots a_{h_k}^k \cdots a_{n_k}^k,\ h_1, \ldots, h_k) \\ (4.8) & \delta & \ni & ((q_1, (a_{h_1}^1, \ldots, a_{h_k}^k)), (q_2, (b_1, \ldots, b_k), (m_1, \ldots, m_k))) \\ (4.9) & \beta & = & (q_2, a_1^1 \cdots a_{h_1-1}^1 b_1 a_{h_1+1}^1 \cdots a_{n_1}^1,\ \ldots,\ a^{k_1} \cdots a_{h_k-1}^k b_k a_{h_k+1}^k \cdots a_{n_k}^k, \\ & & & \quad h_1 + m_1, \ldots, h_k + m_k). \end{array}$$

ACCEPTANCE by an **ATM** is defined as follows: Let α be a configuration, whose state is q.

- ▷ If q is **accept**, M accepts α;
- ▷ if q is **reject**, M rejects α.
- ▷ If q is a universal state, M accepts α if it accepts *all* successor configurations of α.
- ▷ If q is an existential state, M accepts α if it accepts in *one* of the successor configurations of α.

Now M accepts a string w if it accepts the initial configuration $\sigma_M(w)$.

4-13 classification.
(**i**) A NONDETERMINISTIC TURING MACHINE (**NTM**) is an **ATM** without universal states.

(**ii**) A DETERMINISTIC TURING MACHINE (**DTM**) is a **NTM** whose next move relation δ is a *function* (this definition supersedes the informal **4-10**).

(**iii**) Tape number i of an **ATM** is READ-ONLY if in each $d \in \delta$ as above, $a_i = b_i$.

*

The following proposition is an important result. It would go too far to even repeat sketches of the proofs here; but I will prove a version of a subproblem in the next chapter: the case of **iLMG** and read-only **ATM**.

4-14 proposition. [CKS81]
LOGSPACE-ATM = PTIME-DTM
SPACE(n)**-ATM = EXPTIME-DTM**

Least fixed point calculi

Grammars in **iLMG** can be regarded as "economic" versions of formulae in ROUNDS' [Rou88] calculus **iLFP** which defines languages in terms of *integer arithmetic*, and describes exactly the languages that can be recognized in polynomial time. This is proved in [Rou88] using a correspondence to logspace-bounded **ATM** and applying proposition **4-14**.

4-15 definition: iLFP. Let **Ivar** $= \{ x_1, x_2, x_3, \ldots \}$ be an indexed set of INDIVIDUAL VARIABLES, and **Pvar** a set of PREDICATE VARIABLES $(A, B, C, \ldots)$, and let each predicate variable A be assigned an ARITY $a(A)$.

Formulae in the calculus **iLFP** are constructed inductively as follows:

1. If $A \in$ **Pvar**, $n = a(A)$ and $x_1, \ldots, x_n \in$ **Ivar**, then $A(x_1, \ldots, x_n)$ is a formula.
2. If ϕ and ψ are formulae and $x \in$ **Ivar**, then $\phi \wedge \psi$, $\neg\phi$, $\exists x.\ \phi$ (*&c.*) are formulae.
3. A TERM in **iLFP** is an individual variable, the symbol 0, the symbol \$, or $t + 1$ where t is a term.

   If t_1 and t_2 are terms, then $t_1 = t_2$ is and $t_1 < t_2$ are formulae.
4. A RECURSIVE SCHEME Φ is a tuple (Φ_1, Φ_2) where Φ_1 is a finite subset of **Pvar** and Φ_2 is a function that assigns to each $A \in \Phi_1$ a formula ϕ_A. The DEFINING CLAUSE for A is then $A(x_1,\ \ldots,\ x_{a(A)}) \ \Leftrightarrow\ \phi_A$. A recursive scheme can be written down as a list of defining clauses.

   If ϕ is a formula and Φ is a recursive scheme, then $\mu\Phi.\phi$ is a formula.

4-16 example: translation of tuple grammars to iLFP. The context-free grammar of *figure 1.1* is translated into **iLFP** as (4.10), entirely along the lines of its translation to **iLMG**. It is now immediately obvious that a similar translation exists for any grammar that can be represented in **iLMG**. Moreover, such an **iLFP** translation does not contain terms or (in)equality subformulae other than $x = 0$ and $y = \$$.

$$
(4.10)\quad \mu\ [\ \begin{array}{lcll}
\mathrm{C}^{\mathrm{sub}}(i,\ k) & \Leftrightarrow & \exists j.\ \mathrm{C}(i,\ j) \wedge \mathrm{V}^0(j,\ k) & \\
\mathrm{V}^0(i,\ k) & \Leftrightarrow & \exists j.\ \mathrm{N}^0(i,\ j) \wedge \mathrm{V}^{\mathrm{I}}(j,\ k) & \\
\mathrm{V}^{\mathrm{I}}(i,\ k) & \Leftrightarrow & (\exists j.\ \mathrm{V}^{\mathrm{T}}(i,\ j) \wedge \mathrm{N}^0(j,\ k)) \vee & \\
 & & (\exists j.\ \mathrm{V}^{\mathrm{R}}(i,\ j) \wedge \mathrm{V}^0(j,\ k)) \vee & \\
 & & \mathtt{swim}(i,\ k) & \\
\mathrm{V}^{\mathrm{T}}(i,\ k) & \Leftrightarrow & \mathtt{drank}(i,\ k) \vee \mathtt{drink}(i,\ k) & \\
\mathrm{V}^{\mathrm{R}}(i,\ k) & \Leftrightarrow & \mathtt{saw}(i,\ k) \vee \mathtt{see}(i,\ k) \vee & \\
 & & \mathtt{hear}(i,\ k) \vee \mathtt{help}(i,\ k) & \\
\mathrm{N}^0(i,\ k) & \Leftrightarrow & \mathtt{Frank}(i,\ k) \vee \mathtt{Julia}(i,\ k) \vee & \\
 & & \mathtt{Fred}(i,\ k) \vee \mathtt{coffee}(i,\ k) & \\
\mathrm{C}(i,\ k) & \Leftrightarrow & \mathtt{that}(i,\ k) &].\mathrm{C}^{\mathrm{sub}}(0,\ \$)
\end{array}
$$

The formula $\mathrm{C}^{\mathrm{sub}}(0,\ \$)$ is an abbreviation for $\exists x.\exists y.\ x = 0 \wedge y = \$ \wedge \mathrm{C}^{\mathrm{sub}}(x,\ y)$.

*

The recursive schemes in **iLFP** are as the clauses of an **iLMG**, or the productions of a **CFG**. The interpretation of an **iLFP** formula closely resembles that of a **CFG** using the least fixed point interpretation of **1-9**. An **iLFP** formula is interpreted w.r.t. to an input string $w = a_1 \cdots a_\$$ that is represented as a set of *axioms*: for each $1 \leq i, \leq \$$, the predicate $a_i(i - 1,\ i)$ is taken to be true.

4-17 deÆnition: interpretation of iLFP. A PREDICATE ASSIGNMENT ρ is a set of predicates $A(i_1,\ \ldots,\ i_{a(A)})$ where $i_1,\ \ldots,\ i_{a(A)}$ are integer numbers between 0 and n.

Let $w = a_1 \cdots a_\$$ be a sentence, where $a_1,\ \ldots,\ a_\$$ are in **Pvar**. The INITIAL ASSIGNMENT $\rho_0 = \rho_0(w)$ contains all and only the predicates $a_i(i - 1,\ i)$.

A VARIABLE ASSIGNMENT α is a function from **Ivar** to integer numbers between 0 and n. The INTERPRETATION $[\![\phi]\!]$ of an **iLFP** formula ϕ is a function from nonterminal assignments to variable assignments. A formula ϕ is TRUE for w if its interpretation $[\![\phi]\!]\rho_0$ applied to the initial assignment is not empty.

The interpretation is defined inductively as follows:

1. $[\![A(x_1, \ldots, x_n)]\!]\rho = \{\ \alpha \mid A(\alpha(x_1), \ldots, \alpha(x_n)) \in \rho\ \}$
2. $[\![\phi \vee \psi]\!]\rho = [\![\phi]\!] \cup [\![\psi]\!]$ and so forth for the other logical connectives
3. $[\![\exists x.\phi]\!]\rho = \{\ \alpha \mid \exists i \leq n :\ \alpha(x \to i) \in [\![\phi]\!]\rho\ \}$, where $\alpha(x \to i)$ is the variable assignment that agrees with α except for the variable x, to which it assigns i.

4. $[\![t_1 = t_2]\!]\rho = \{\ \alpha \mid \alpha(t_1) = \alpha(t_2)\ \}$ where $\alpha(t)$ is the integer number obtained by substituting $\alpha(x)$ for every variable x occurring in t and substituting n for \$. Analogously for $<$.

5. $[\![\mu\Phi.\phi]\!]\rho = [\![\phi]\!] \left(\bigcup_{k=0}^{\infty}[\![\Phi]\!]^k\rho\right)$ where $[\![\Phi]\!]\rho = \rho \cup \{\ A(i_1,\ \ldots,\ i_{a(A)}) \mid A \in \Phi_1$ and $\exists\alpha \in [\![\Phi_2(A)]\!]\rho$ such that $\alpha(x_k) = i_k\ \}$

*

The interpretation of a recursive scheme here needs, in contrast to the previous example of the fixed-point construction in **1-9**, to be made continuous by explicitly joining ρ in with $[\![\Phi]\!]\rho$.[4]

It is now straightforward to check how formula (4.10) says that $S(0, \$)$ should hold, where S is the smallest relation over two integer numbers satisfying the clauses between the square brackets.

Let's look at the sentence *that Julia drank coffee*, and let's denote the recursive scheme in (4.10) by Φ. Then $n = \$ = 4$, and

$$\begin{aligned}
&(4.11) & \rho_0 &= \{\ \texttt{that}(0,1),\ \texttt{Julia}(1,2),\\
&&&\quad\ \texttt{drank}(2,3),\ \texttt{coffee}(3,4)\ \}\\
&(4.12) & \rho_1 = [\![\Phi]\!]\rho_0 &= \rho_0 \cup \{\ \mathrm{V^T}(2,3),\ \mathrm{N^0}(1,2),\ \mathrm{N^0}(3,4),\ \mathrm{C}(0,1)\ \}\\
&(4.13) & \rho_2 = [\![\Phi]\!]\rho_1 &= \rho_1 \cup \{\ \mathrm{V^I}(2,4)\ \}\\
&(4.14) & \rho_3 = [\![\Phi]\!]\rho_2 &= \rho_2 \cup \{\ \mathrm{V^0}(1,4)\ \}\\
&(4.15) & \rho_4 = [\![\Phi]\!]\rho_3 &= \rho_3 \cup \{\ \mathrm{C^{sub}}(0,4)\ \}\\
&&&= \{\ \texttt{that}(0,1),\ \texttt{Julia}(1,2),\\
&&&\quad\ \texttt{drank}(2,3),\ \texttt{coffee}(3,4),\\
&&&\quad\ \mathrm{V^T}(2,3),\ \mathrm{N^0}(1,2),\ \mathrm{N^0}(3,4),\ \mathrm{C}(0,1),\\
&&&\quad\ \mathrm{V^I}(1,3),\ \mathrm{V^0}(0,3),\ \mathrm{C^{sub}}(0,4)\ \}\\
&&&= \bigcup_{k=0}^{4}[\![\Phi]\!]^k\rho_0
\end{aligned}$$

So $[\![\mu\Phi.\mathrm{C^{sub}}(x,y)]\!]\rho_4$ contains the assignment $x \to 0,\ y \to 4$, and formula (4.10) is true.

The following proposition links **iLFP** and deterministic polynomial time; I will not give the proof here, but the constructions in section **5.1** will closely resemble those of Rounds.

4-18 proposition. [Rou88] The languages recognized by **iLFP** formulae are the same as those recognized by an **ATM** in log space.

*

[4] This is not the case in Rounds' original formulation, which limits the use of the negation operator in such a way that atomic predicates can occur only positively.

While **iLFP** is useful for showing rigorously that for a given linguistic formalism there must be a fixed recognition procedure with a polynomial time complexity, it doesn't give a precise upper bound to this complexity. For example, Rounds speaks of an $\mathcal{O}(n^{18})$ upper bound to recognition time of head grammars, which are known to be recognizable by more informally defined device in time $\mathcal{O}(n^6)$, and $\mathcal{O}(n^{12})$ for context-free grammars as opposed to the generally accepted value $\mathcal{O}(n^3)$. Rounds doesn't go into great detail, and closer inspection of the proof of **4-14** shows that the results will probably get even a bit worse: the number of variables in the **iLFP** formula is not squared, but raised to the power 3 in the 1-tape simulation (see proposition **5-8**).

Relating iLMG to iLFP

Although as said, the next chapter will carry out a version of Rounds' proofs directly for **iLMG** and $\mathcal{S}$-**LMG**, and hence it will follow that **iLMG** and **iLFP** both describe PTIME, it is worth looking in a bit more detail at how **iLMG** and **iLFP** relate. Clearly, **iLMG** can be regarded as a sort of 'undressed' version of **iLFP**; the equivalence of the formalisms suggests that universal quantification, negation, successor, equality and inequality in **iLFP** are redundant. Proposition **4-20** shows that not only this is the case, but also the number of variables in a clause of a corresponding **iLMG** is not larger than that in the original **iLFP** formula.

4-19 proposition. For every **iLMG**, there is a weakly equivalent **iLFP** formula.

Proof. Straightforward. The clauses of the **iLMG** directly represent a recursive scheme. It is a standard exercise (done for **CFG** in chapter **1**, proposition **1-10**) to show that the fixed point semantics of the **iLFP** scheme is equivalent to the derivational semantics of the **iLMG**. □

4-20 proposition. For every **iLFP** formula, there is a weakly equivalent **iLMG**.

Proof. (outline) (**i**) collapse all embedded recursive schemes into one recursive scheme; this can be done by standard variable renaming techniques. Assume moreover that the formula is of the form $\mu\Phi.S(0,\$)$, where the schema Φ does not contain the constants 0 and \$. (This can be done by increasing the number of arguments to all predicates defined in Φ by 2).
(**ii**) note that $\forall$ can be represented as $\neg\exists\neg$; and introduce new predicate variables for subformulae of defining clauses until each defining clause is either in $\vee$-$\exists$-$\wedge$ normal form or of the form $A(x_1,\ldots,x_n) \Leftrightarrow \neg B(x_1,\ldots,x_n)$.
(**iii**) translate the negation free, normal form clauses into an **iLMG**, where $=$, $<$ and $+1$ are replaced with predicates, the clauses for which are straightforward.
(**iv**) for every $A(\cdots) \Leftrightarrow \neg B(\cdots)$ clause left untranslated, add a nonterminal $B^{\neg}$, and replace all occurrences of A in the **iLMG** with $B^{\neg}$. Construct clauses for these negated nonterminals as follows:

for $B \in T$: there will be a number n of clauses for B:

$$\begin{array}{lll}
B(\vec{x}_1) & :- & C_{11}(\vec{y}_{11}), \ldots, C_{1m_1}(\vec{y}_{1m_1}). \\
 & \vdots & \\
B(\vec{x}_n) & :- & C_{n1}(\vec{y}_{n1}), \ldots, C_{nm_n}(\vec{y}_{nm_n}).
\end{array}$$

Introduce new nonterminals $B_1^{\neg}, \ldots, B_n^{\neg}$, and the following clauses:

$$\begin{array}{lll}
B^{\neg}(\vec{x}_1) & :- & B_1^{\neg}(\vec{x}_1), \ldots, B_n^{\neg}(\vec{x}_1). \\
B_1^{\neg}(\vec{x}_1) & :- & C_{11}^{\neg}(\vec{y}_{11}). \\
 & \vdots & \\
B_1^{\neg}(\vec{x}_1) & :- & C_{1m_1}^{\neg}(\vec{y}_{1m_1}). \\
 & \vdots & \\
B_n^{\neg}(\vec{x}_n) & :- & C_{n1}^{\neg}(\vec{y}_{n1}). \\
 & \vdots & \\
B_n^{\neg}(\vec{x}_n) & :- & C_{nm_n}^{\neg}(\vec{y}_{nm_n}).
\end{array}$$

for $B \notin T$: Let $B = b$ be a terminal symbol. Then add the clauses

$$\begin{array}{llll}
b^{\neg}(x,z) & :- & c(x,y), \mathrm{ITS}(y,z). & [\text{for all } b \neq c \in T] \\
b^{\neg}(x,z) & :- & b(x,y), \mathrm{ITP}(y_2,z). & \\
b^{\neg}(x,y) & :- & \mathrm{ITS}(y,x). & \\
 & & & \\
\mathrm{ITS}(x,x). & & & \\
\mathrm{ITS}(x,y) & :- & \mathrm{ITP}(x,y). & \\
\mathrm{ITP}(x,z) & :- & c(x,y), \mathrm{ITS}(y,z). & [\text{for all } c \in T]
\end{array}$$

(ITS and ITP are nonterminals that recognize arbitrary strings in T^* and T^+, respectively). □

4-21 corollaries. The construction in the proof of proposition **4-20** does not increase the total number of variables appearing in the **iLFP** clause during the translation process to **iLMG**, provided that an **iLFP** formula is in the normal form described under (**i**) in the proof of proposition **4-20**). This has the following consequences:

1. For **iLFP**, as for **iLMG**, the highest number of variables appearing in a clause of a formula determines the polynomial bound for the time complexity;
2. From item (**iv**) in the proof it follows that n-**iLMG** (and therefore $n - \mathcal{S} -$ **LMG**, see the following chapter) are closed under complement; in fact negation can be added at clause level, yielding a formal[5] proof that grammars in the n-**MCFG** or simple n-**LMG** spectrum enriched with REJECT PRODUCTIONS ([Vis97] p52*ff*) remain in n-$\mathcal{S}$-**LMG**; and hence preserve their polynomial complexity bound.

[5] Visser has a "proof by algorithm" of the tractability of reject productions—this algorithm however covers parsing whereas my construction concerns recognition only.

Conclusions to chapter 4

This chapter introduced a number of well-known concepts, partly in a notational system and from a point of view that clears the way for the approach of the next few chapters.

It has also shed light on the rôle of *sentence indices* in informal complexity reasoning, and how this leads to straightforward top-down parsing algorithms. The predicate-descent method proposed in section **4.2** is the most straightforward realization of such algorithms, and I haven't seen this idea being formulated directly in the literature, which seems to have a focus on going through a string from its 'beginning' to its 'end'—a model that becomes problematic when nonterminals are allowed to derive tuples of strings that may end up in different places in a derived sentence. An essential observation is that the undirectional approach does not break the standard time complexity results of $\mathcal{O}(n^3)$ for **CFG** and $\mathcal{O}(n^6)$ for **TAG**, provided that the algorithms are formalized within the random access model of computation.

Also new is the direct deterministic Turing machine implementation of a group of variants of YOUNGER's algorithm. It *seems* to be adaptable to **PMCFG** without much effort, and the remarks by YOUNGER about universal recognition seem to hold too; certainly for **MHG**. I have not been able to find the time to investigate rigorously whether this indeed extends to a universal recognition algorithm for **PMCFG** that is polynomial in the length of the input; this seems to be in contradiction with claims made on the literature, but in section **5.3** in the next chapter, I will argue that this is not the case.

I have convinced myself that an extension of Younger's algorithm to **iLMG**/$\mathcal{S}$-**LMG** is considerably harder, because sharing on the RHS of **LMG** clauses destroys the possibility of induction on the size of the items. Moreover, the reduction result to an analogue of CNF does not seem to work for simple **LMG** in general.

Although the **iLMG** formalism is fairly generally applicable for formalization parsing strategies, sometimes enriched versions of **iLFP** are more convenient, such as when I devise strategies for **XG** parsing in chapter **6**. The reduction of **iLFP** to **iLMG** proved in the last section of the current chapter ensures that such rich descriptions preserve the possibility to use the predicate-descent complexity heuristics based on counting the number of variables in clauses.

Chapter 5

Literal movement grammar

I promised in chapter **3** to show later that the general class of **LMG** generate all recursively enumerable languages, and that a restriction, SIMPLE **LMG**, has a recognition problem solvable in polynomial time. This is the subject of this chapter.

5-1 proposition. The class **LML** of languages recognized by generic **LMG** is precisely the class of recursively enumerable languages.

Proof. It is a well-known result (see *e.g.* [Gin75] p. 125) that any r.e. language L can be described as $h(L_1 \cap L_2)$ where h is a homomorphism and the languages $L_1 = \mathcal{L}(G_1)$ and $L_2 = \mathcal{L}(G_2)$ are context-free. Proposition **3-27** implies that each context-free grammar is an **LMG**; proposition **3-22** proved that **LMG** are closed under homomorphism and **3-28** proves closure under intersection. □

In the previous chapter I showed how integer sentence indices are the key to finding efficient—polynomial time—recognition algorithms. There, attention was limited to context-free grammar and head grammar, although I already defined **iLMG** as an intermediate language.

In this chapter, I will establish an additional proof cycle between $\mathcal{S}$-**LMG**, **iLMG** and read-only alternating Turing machines, touching briefly on the correspondence between this unorthodox class of *read-only* alternating Turing machines and deterministic polynomial time.

I will also investigate how one may obtain tight bounds to the exponent c such that classes of simple **LMG** are recognizable in $\mathcal{O}(n^c)$ time, look at the problem of universal recognition and the discrepancy between Turing machine and random access models, and discuss another version of **LMG** that corresponds to deterministic exponential time.

5.1 Complexity of simple LML

Chapter **3** introduced simple **LMG**, without arguing for the particular format of clauses that this restriction required. The constraint of noncombinatoriality naturally arises when one is thinking about a naive top-down implementation of a recognition algorithm based on **LMG**. But the why precisely simplicity as stated in definition **3-26**?

The examples have already shown how **LMG** subsumes the chain **CFG** $\subseteq$ **HG** $\subseteq$ **LCFRS** $\subseteq$ **PMCFG** of formalisms of increasing generative capacities. The fixed recognition problems for these formalisms are known to be decidable in polynomial time. I will now show that simple **LMG** is interesting from a formal point of view, because it describes exactly the class **PTIME** of languages recognizable in deterministic polynomial time.

First, I will show that $\mathcal{S}$-**LMG** can be mapped to **iLMG**; then, that **iLMG** can be mapped to *read-only* **ATM** and finally, that read-only **ATM** can be modelled in $\mathcal{S}$-**LMG**. The resulting chain of results is summarized schematically in *figure 5.1*. One link that is not made very explicit here or in the previous chapter— it was only briefly sketched in proposition **4-20**—is that $\mathcal{S}$-**LML** indeed contains **PTIME**. However, it is well-known that read-only **ATM** amounts to the same thing as a logspace-bounded **ATM**; alternatively, the equivalent of **5-9** for logspace-bounded **ATM** can be found in [Gro98]. Section **5.2** will discuss obtaining tight bounds to the degree of polynomial-time algorithms for simple **LMG**.

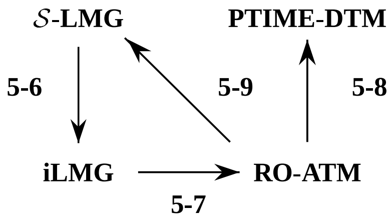


Figure 5.1: Results in this section.

Motivation for the simplicity constraint

Least fixed point semantics of rule based grammar are attractive in many ways; in the case of **LMG**, they will give a link to the integer sentence index model, as well as some insight into why the property of simplicity is a sensible restriction to the general form of **LMG** rules in order to get tractable recognition. **iLMG** as defined in the previous chapter will serve as an intermediate representation.

5-2 deÆnition: lfp semantics for LMG. Let $G = (N, T, V, S, P)$ be an **LMG**. Let $\mathcal{NA}$ be the set of nonterminal assignments: functions ρ mapping a nonterminal to a set of arbitrary tuples of strings over T. The set of productions P is then associated with an operator $[\![G]\!]$ taking an assignment as an argument and producing a new assignment, defined as follows: if ρ is an assignment, and

$$A(w_1, \ldots, w_p) \;\; \textbf{:-} \;\; B_1(v_1^1, \ldots, v_{p_1}^1), \ldots, B_m(v_1^m, \ldots, v_{p_m}^m).$$

is an instantiation of a clause in P, and for each $1 \leq k \leq m$, $(v_1^k, \ldots, v_{p_k}^k) \in \rho(B_k)$, then $(w_1, \ldots, w_p) \in ([\![G]\!]\rho)(A)$.

Define the complete partial order $(\mathcal{NA}, \sqsubseteq)$, the bottom element ρ_0 and the join $\sqcup$ as done before for context-free grammars; then $[\![G]\!]$ is a continuous and monotonic operator on $(\mathcal{NA}, \sqsubseteq)$.

(5.1) $\rho_1 \sqsubseteq \rho_2 \;\Leftrightarrow\; \forall A \in N.\; \rho_1(A) \subseteq \rho_2(A)$:

(5.2) $\rho_0(A) = \emptyset$ for all $A \in N$

(5.3) $$(\bigsqcup X)(A) = \bigcup_{\rho \in X} (\rho(A))$$

The interpretation of a grammar is then the least fixed point of $[\![G]\!]$:

(5.4) $$\mathcal{I}_G \;=\; \bigsqcup_{k=0}^{\infty} [\![G]\!]^k \rho_0$$

i.e. a function which takes a nonterminal and yields a set of tuples of strings. If the arity of start symbol S throughout the grammar is 1, then $\mathcal{I}_G(S)$ will be the language recognized by the **LMG** in the traditional sense; otherwise obtain the language by concatenating the tuples in $\mathcal{I}_G(S)$.

*

As before, it is easy to check that the rewriting semantics and the fixed point semantics are equivalent. The fixed point semantics is a useful tool for several purposes. It is an interpretation of grammars that is both mathematically elegant and more detailed than a rewrite semantics. More detailed, because it does not merely characterize the language generated by a single designated start symbol, but characterizes each of the nonterminals separately (*i.e.* the way the grammar classifies phrase types), and reflects the derivational behaviour of the grammar, without looking at single derivations in particular.

Now remember that the aim is to find out how the **LMG** grammars can be restricted in such a way that recognition can be performed as an alternating search in logspace. For a given string of length n, in space $\log n$ one can encode a bounded set of numbers ranging from 0 to n (in binary encoding). This means that the arguments of an **LMG** predicate in a derivation have to be encoded each with a bounded set of numbers. This is precisely what **iLMG**, defined in the previous chapter, does.

To make the step from **LMG** to **iLMG** smaller, introduce the following interpretation relative to a given input string.

5-3 definition: substring filter. Let $\rho \in \mathcal{NA}$ and define the operator $\sqcap$ restricting an assignment to a string as follows: $(w_1, \ldots, w_n) \in (\rho \sqcap w)(A)$ iff $(w_1, \ldots, w_n) \in \rho(A)$, and $w_1, \ldots, w_n$ are substrings of w; furthermore put $([\![G]\!] \sqcap w)(\rho) = ([\![G]\!]\rho) \sqcap w$.

$*$

The question is now: how can I make sure that

$$\left(\bigsqcup_{k=0}^{\infty} [\![G]\!]^k \rho_0 \right) \sqcap w \;=\; \bigsqcup_{k=0}^{\infty} ([\![G]\!] \sqcap w)^k \rho_0 ?$$

Redefine the fixed point semantics as follows.

5-4 definition: integer lfp semantics for LMG. Let $w = a_0 a_1 \cdots a_{n-1}$ be a terminal string of length n; then $\mathcal{NA}_w$ is the set of INTEGER NONTERMINAL ASSIGNMENTS ρ FOR THE INPUT w, mapping a nonterminal to a set of tuples of pairs of integers between 0 and n. Then define $[\![G]\!]_w$ as follows: if ρ is an integer assignment and

$$\begin{array}{lll} A(a_{l_1} \cdots a_{r_1},\ \ldots,\ a_{l_p} \cdots a_{r_p}) & \textbf{:-} & B_1(a_{l_1^1} \cdots a_{r_1^1},\ \ldots,\ a_{l_{p_1}^1} \cdots a_{r_{p_1}^1}), \\ & & \ldots, \\ & & B_m(a_{l_1^m} \cdots a_{r_1^m},\ \ldots,\ a_{l_{p_m}^m} \cdots a_{r_{p_m}^m}). \end{array}$$

is an instantiation of a clause in P, and for each $1 \le k \le m$,

$$(\langle l_1^k, r_1^k \rangle,\ \ldots,\ \left\langle l_{p_k}^k, r_{p_k}^k \right\rangle) \in \rho(B_k),$$

then

$$(\langle l_1, r_1 \rangle \ldots, \langle l_p, r_p \rangle) \in ([\![G]\!]_w \rho)(A).$$

$*$

It is important to see that what is done here, is in the general case *not* the same as taking the string-based least fixed point semantics **5-2**, and intersecting the sets of tuples with the domain of substrings of a given w, as in definition **5-3**. An instantiated clause

$$A(w_1, \ldots, w_p) \;\textbf{:-}\; B_1(v_1^1, \ldots, v_{p_1}^1),\ \ldots,\ B_m(v_1^m, \ldots, v_{p_m}^m).$$

such that $w_1, \ldots, w_p$ are substrings of w, but the v_j^i are not, will be ignored in the integer least fixed point semantics: even though all w_i are substrings and $A(w_1, \ldots, w_p)$ is derived by the grammar, it cannot be 'reached' with a derivation in which only substrings of the input appear.

So this type of clause needs to be ruled out; that is if $w_1, \ldots, w_p$ are substrings of the input, then so must the v_j^i. This now is why simple **LMG** is defined by disallowing terms other than single variables on the *right hand side* of the clauses. In this way, each rule can be uniquely replaced by a clause that is talking about integer positions instead of strings, while allowing the sharing constructions that cannot be handled by multiple context-free grammars (*cf.* example **3-32**). Note that variables on the RHS that do not appear on the LHS must also be disallowed, because these could be instantiated with any string.[1]

5-5 definition: simple LMG, repeated. An **LMG** is called SIMPLE if its clauses $R \in P$ are all of the form

$$A(t_1, \ldots, t_p) \;\textbf{:-}\; B_1(x^1_1, \ldots, x^1_{p_1}),\ \ldots,\ B_m(x^m_1, \ldots, x^m_{p_m}).$$

where the variables x^i_j on the RHS need not be disjoint, but each these variables appears precisely once in $t_1, \ldots, t_p$.

*

Finally then, proposition **5-6** is the translation from simple **LMG** to **iLMG**, from which it follows that $\mathcal{S}$-**LML** is recognizable in polynomial time.

5-6 proposition. For every $\mathcal{S}$-**LMG** there is a weakly equivalent **iLMG**.

Proof. Let G be a simple **LMG**. First, construct a weakly equivalent simple **LMG** $G' = (N, T, V', S, P')$ such that for each predicate $A(t_1, \ldots, t_n)$ on the LHS of a clause, t_i, either

type 1. $n = 1$ and t_1 is a single terminal symbol, or

type 2. all t_i are sequences of variables only.

Then $G'' = (N \cup T, V'', S, P'')$ is constructed as follows.

Let V'' contain two unique new variables l and r. A clause $A(a)$ in P' of type 1 is translated as $A(l, r) \;\textbf{:-}\; a(l, r)$ in P''.

For each clause R' of type 2 in G', P'' contains a clause R''. Let the LHS of R' be $A(t_1, \ldots, t_n)$, and $t_i = y^i_1 \cdots y^i_{p_i}$ (all y^i_j are disjoint by definition). Then V'' contains unique new variables y^i_0, $1 \leq i \leq n$, and R'' is obtained by replacing the LHS of R' by $A(y^1_0, y^1_{p_i}, \ldots, y^n_0, y^n_{p_n})$ and replacing each variable y^i_j in the RHS of R' by the two variables y^i_{j-1}, y^i_j.

It is now straightforward, given w, to prove inductively that G' recognizes a term over substrings $v = a_l \cdots a_r$, if and only if G'' recognizes the same term with each substring v encoded as a pair $l - 1, r$. Conversely, G' will recognize only predicates over substrings of w because G' is simple. □

The converse, going from **iLMG** to **LMG** is not trivial, because we cannot simply take the integer arguments up as pairs i, j such that $i \leq j$. Instead, integers must be represented as substrings of the input string w. Instead, I will repeat Rounds' **ATM** construction for the simpler case of *read-only* **ATM**, which is no harder than reducing **iLMG** to simple **LMG**.

[1] The decision not to allow variables to appear on the LHS more than once is more arbitrary—it does not influence the weak generative capacity.

Read-only ATM

This is a simple reconstruction of Rounds' implementation of **iLFP** on a regular alternating Turing machine. Because **iLMG** does not have the arithmetic operations of **iLFP**, a much simplified model of an **ATM** that has k read-only input tapes, and no read-write tapes, is sufficient, where k is the largest number of variables appearing in a clause of the **iLMG**. This then yields a slightly tighter polynomial time bound, when I work out the read-only alternative to CHANDRA et al.'s THEOREM 3.3 in [CKS81].

5-7 proposition. Every **iLML** can be recognized by a read-only **ATM**.

Proof. Let an **iLMG** $G = (N, T, V, S, P)$ be given. Assume that each nonterminal appears with only one arity; in particular, the start symbol appears uniquely with arity 2. For each clause R, p_R is the number of variables on its LHS and r_R the total number of variables in the clause.

Let r be the largest number of variables used in a clause in P; assume that $V = \{x_1, \dots, x_r\}$; for each clause, assume that the variables used are $x_1, \dots, x_{r_R}$, and the variables on the LHS are $x_1, \dots, x_{p_R}$.

The **ATM** M now has r tapes, all of which are read-only and contain the input string when the machine is started. M has states q_A for each nonterminal, q_R for each clause, q^R_l for each clause R and each tape/variable number l, and a number of auxiliary substates of these.

Assume the input string is $w = a_1 \cdots a_n$. In the initial state, the machine moves the head of tape 2 to its end (position n), and enters state q_S.

The states license the following actions:

State q_A : $\bigvee_{R \in P(A)}$ (enter state q_R)
where $P(A)$ is the set of rules whose LHS is an A-predicate.

For nonterminal clauses R:

State q_R : move tapes $p_R + 1 \cdots r_R$ to initial position, and enter state $q^R_{p_R+1}$

State q^R_l, $l \le r_R$: (enter state q^R_{l+1}) $\vee$ (if not at end, move head l right).

State q^R_{r+1} : $\bigwedge_{\phi = B(\dots) \text{ on RHS of } R}$ (copy tape head positions[2]and enter state q_B).

For terminal clauses $A(i,j)$ **:-** $a(i,j)$.:

State q_R: check that the terminal under head 1 is a, then check that if head 2 is moved leftward, it points at the same cell as head 1; if success enter $q_{\textbf{accept}}$ else $q_{\textbf{reject}}$.

[2] In the worst case, copying the positions of the heads to copy the variables may require one additional tape. This can be circumvented by multiplying out the machine's states over all permutations of the tapes.

Now M follows precisely the derivational interpretation of the **iLMG** w.r.t. the input string w. □

5-8 proposition. A k-tape read-only **ATM** can be simulated on a 1-tape **DTM** in time cn^{3k}, where n is the size of the input string.

Proof. Analogous to [CKS81], THEOREM 3.3. Lay out the instantaneous descriptions of the **ATM** M on the tape of **DTM** N. There are $t = |Q_M|n^k$ IDs. The execution paths of the **ATM** are guaranteed to terminate in t steps or to run forever. Initially, mark all accepting IDs **accept**, rejecting IDs **reject**, all other IDs **unknown**.

Then do t passes over the tape; for each cell, look up the values at the successor IDs (this requires $\mathcal{O}(t)$ time as these configurations may be anywhere on the tape), and compute their 3-valued boolean product.

So the machine makes t passes of each $\mathcal{O}(t^2)$ steps.[3] □

The circle is made complete by showing that a read-only alternating Turing machine can be simulated by a simple **LMG**. A similar, more involved proof translating a logspace-bounded **ATM** in $\mathcal{S}$-**LMG** is in [Gro98].

5-9 proposition. For any read-only **ATM** M, there is an $\mathcal{S}$-**LMG** G such that $L(G) = L(M)$.

Proof. Let the input string be $w = a_1 \cdots a_n$. Let w.l.o.g. universal states not move any heads. The **LMG** has one nonterminal for each state of the **ATM**, plus a start symbol S and two auxiliary nonterminals PlusOne and PlusTwo. An instantaneous description $(q, h_1, \ldots, h_k)$ is represented by the instantiated **LMG** predicate

$$q(w,\ a_1 \cdots a_{h_1}, w,\ a_1 \cdots a_{h_2}, w,\ \ldots,\ a_1 \cdots a_{h_k}, w).$$

The first copy of w is the 'fuel' for the operations PlusOne and PlusTwo, that will be responsible for simulating movement of the tape heads. The start rule is

$$\mathrm{S}(x)\ \ \mathbf{:-}\ \ q_0(x,\ \varepsilon, x,\ \varepsilon, x,\ \ldots,\ \varepsilon, x).$$

Now reading a tape, moving left and moving right can be straightforwardly simulated. Every m-fold existential state will correspond to a nonterminal with m clauses which each have on their RHS k auxiliary predicates and one successor predicate. Each m-fold universal state corresponds to a nonterminal with 1 clause whose RHS contains m successor predicates.

A simple example, assuming just one tape and a single successor state is

$$\begin{array}{llll} (5.5) & q_1(m, xc, yz) & \mathbf{:-} & \mathrm{PlusTwo}(m, x, y),\ q_2(m, y, m). \\ & \mathrm{PlusTwo}(am, ax, ay) & \mathbf{:-} & \mathrm{PlusTwo}(m, x, y). \\ & \mathrm{PlusTwo}(abm, \varepsilon, ab). & & [\text{ for each } a, b \in T\ ] \end{array}$$

[3] Note that this is a factor t more than Rounds suggests. I don't see how this can be improved in this 1-tape construction, although it can be improved using a reduction from multi-dimensional **BAM** to 2-tape 1-dimensional **TM**.

State q_1 checks that the symbol under the head is c, and if so, moves the head one symbol to the right and checks state q_2. The auxiliary nonterminal PlusTwo generates the right move.; PlusOne would generate no move, and using x directly generates a left move.

Finally, there is one clause $q_{\mathbf{accept}}(x_1, \dots, x_{2k+1})$., and no clause for $q_{\mathbf{reject}}$. □

We can now conclude that **iLMG**, $\mathcal{S}$-**LMG** and **RO-ATM** all generate **PTIME**.

5-10 corollary. $\mathcal{S}$-**LML** $=$ **iLML** $=$ $\mathcal{L}$(**RO-ATM**) $=$ **PTIME**-**DTM**.

Proof. The equivalence of $\mathcal{S}$-**LML**, **iLML** and $\mathcal{L}$(**RO-ATM**) follows by the chain **5-6**, **5-7** and **5-9**. Equivalence between **iLMG** and **PTIME** was already proved in the previous chapter. Alternatively, **RO-ATM** in **PTIME** by **5-8** and **PTIME** in $\mathcal{S}$-**LML** by the proof in [Gro98]. □

5.2 Finding the degree of the polynomial

Under the **RAM** model of computation, recognition can be performed by the **iLMG** predicate descent algorithm outlined in the previous chapter (example **4-8**). The algorithm will store **iLMG** items in its memo table, which results in a space complexity of $\mathcal{O}(n^p)$ where p is the largest arity of an **iLMG** nonterminal. Each item needs to be computed only once, and the function for a clause R contains q_R nested loops over $0 \cdots n$ where q_R is the number of variables on the RHS of the clause that do not appear on its LHS. So the total time complexity is $\mathcal{O}(n^r)$ where $r = \max_{R \in P} p_R + q_R$ is the largest number of variables in a clause in the **iLMG**.

5-11 classiÆcation: analysis of simple LMG examples. In terms of $\mathcal{S}$-**LMG**, p_R is 2 times the number, say a_R, of arguments in a predicate occurring in the grammar, and q_R corresponds to the maximal difference between the total number of variables in a clause and the number of argument positions of the predicate on its LHS—each additional variable introduces an extra index that needs to be looped over. Because the time complexity can be calculated as the sum of the times spent for each single clause, this means that the polynomial complexity bound for the grammar is $r = \max_{R \in P} a_R + b_R$ where b_R is the total number of variables in the clause R. The space complexity bound p is $2a$ where $a = \max_{R \in P} a_R$.

To see that this analysis also works for grammars like (3.16) which have both terminal symbols and variables on the LHS (which means they don't satisfy the w.l.o.g. condition on the format of clauses in proposition **5-6**), it should be noted that the terminals do not introduce any independent index values, because their length as a string is always 1. The bounds thus obtained for formalisms such as **MHG** whose recognition problems have been studied in the literature are tight; no better results are known. This is summarized for formalisms in chapter **3** and grammars in section **3.5** in *figure 5.2*.

The **PMCFG** grammar for $\mathtt{a}^{2^n}$ in *figure 3.7* is left out of this analysis because it has reduplication which is not allowed in simple **LMG**. When it is translated to a simple **LMG** using an Eq predicate, the resulting time complexity would be n^4; however, a smarter predicate descent approach which would check that the two occurrences are

Grammar	a	max. $a_R + b_R$	$p = 2a$	r	*space*	*time*
bilinear **CFG**	1	$1+2$	2	3	n^2	n^3
MHG	2	$2+4$	4	6	n^4	n^6
(3.16) ($\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$), p72	2	$2+2$	4	4	n^4	n^4
figure 3.10 (Dutch), p74	4	$3+5$	8	8	n^8	n^8
figure 3.13 (German), p79	2	$2+3$	4	5	n^4	n^5

Figure 5.2: Tuple grammars and their complexity bounds.

equal strings immediately gets a time bound of n^3, because there are n^2 items each taking $\mathcal{O}(n)$ time for splitting up the argument string and checking the equality. So use of reduplication accounts for a single extra factor n in all cases. This is precisely what is done in the prototype implementation described in section **7.2**.

*

The bounds obtained here can be formalized in terms of random access machines. In the Turing machine constructions of section **5.1** however, the polynomial bounds I just calculated are tripled. This can be reduced to only $\mathcal{O}(n^{2r})$ by the following argument: the predicate descent algorithm does *not* need random memory access if it is executed on a Turing machine with an r-dimensional tape. It is a known result that an n-dimensional **BAM** can be reduced to a 2-tape 1-dimensional **DTM**, where the execution time gets squared.

This is the best general result that I can see, but for **MHG**, this construction yields an $\mathcal{O}(n^{12})$ algorithm as opposed to the $\mathcal{O}(n^6)$ implementation illustrated in section **4.3**. It remains remarkable, and unsolved, that while the Turing recognizer for **MHG** of section **4.3** can be extended to one for **PMCFG** that has the correct polynomial bound of $\mathcal{O}(n^6)$, this seems to be impossible for the general case of simple **LMG**, because due to the possibility of sharing variables on the RHS, the induction on the total length of the string arguments fails.[4]

One the one hand a conclusion could be that a Turing model is highly idealized and very suitable for reasoning about large classes such as **PTIME**, but less suited for talking about bounded polynomial classes. On the other hand, the **RAM** model is an idealization too, because it assumes the presence of limitless amounts of memory: a space complexity of n^4, assumed layed out as a 4-dimensional array in computer memory, cannot realistically be called 'tractable'. A sentence of 10 words would already require 20.000 units of storage; a **Pascal** program of numerous kilobytes would not be machine parsable.

In practice therefore, one often reverts to representing the storage through linked data structures to avoid reserving storage that will not be used. These multi-dimensional linked lists need to be traversed, and although this does not double or triple the polynomial bound as in the generic Turing implementation, it does increase time complexity by a factor n (linear storage) or **log** n (storage as binary trees).

[4] It would moreover be interesting to see if this descrepancy between formalisms like n-**PMCFG** whose underlying tree sets are local, and the intersection-closed simple n-**LMG**, also yield different results for other known methods of predicting tractability and polynomial bounds, such as [McA93]. It seems unclear to me what model of computation McAllester relies on, and whether this has consequences for his analysis.

5.3 Parsing and universal recognition

Putting the discrepancy between the **RAM** and **TM** models aside, another thing to note is that so far the complexity study was limited to FIXED RECOGNITION. Without leaving the formal complexity perspective of this chapter, important questions to be raised are (**i**) what about UNIVERSAL RECOGNITION, that is, writing an algorithm that performs its task for any given grammar, rather than recognizing a single language and (**ii**) how about algorithms that return a representation of the possible derivations of a sentence rather than a crude **yes** or **no**?

I will pay most attention here to the first question, leaving the second to chapter **7**, which is of a more practical nature; formally, it would suffice to say that the *memo table* after execution of a predicate descent algorithm contains information that could be qualified as a parse forest. A parse forest, however, is an unformalized notion, and it is unclear whether it must contain explicit 'links' to daughter nodes, or whether these need to be inferred from the grammar.

As to the first question, it has been argued in the literature that when size parameters of the grammar are unknown in advance, that is the length of rules, and the number of variables in the case of **LMG** type grammars, then universal recognition will take at least exponential or nondeterministic polynomial time. For the latter class, it is unknown whether there are problems that in **PTIME-NTM** but not in **PTIME-DTM** (the P = NP problem). Examples of such claims are BARTON, BERWICK and RISTAD [BBR87] (NP-hardness for **GPSG** and **LFG**), and KAJI et al. [KNSK92] (EXP-POLY time hardness for **PMCFG**).

The content of these universal recognition results must be assessed very carefully—what the authors suggest is often not what they are proving. When the NP-hardness of **LFG** and **GPSG** is discussed in [BBR87], great effort is made to stress that it is not a good argument to give an a priori bound to the number of features needed in any possible natural language grammar. For any known such bound, a language may be found that must be classified as a natural language, but needs more features to be strongly-adequately described. However, this does *not* imply that the individual size parameters of the grammar, that is the number of nonterminals, the dimensions of a feature space, and in the case of **LMG**, the different numbers of variables in clauses should be considered irrelevant in determining the complexity. And this is exactly what both [BBR87] and [KNSK92] do: they express the execution time as a function of the sum of the sizes of the grammar and the input. Moreover, they do not look at an arbitrary grammar and an arbitrary input, but given a sentence, generate a grammar, putting input and grammar together in one pigeon-hole.

This leads to apparently contradictory results: both of the following two statements can be proven:

5-12 proposition. [KNSK92] The universal recognition problem for **PMCFG** is EXP-POLY time hard.

5-13 proposition. The universal recognition problem for **PMCFG** can be solved in time $\mathcal{O}(cn^{3r})$, where r is as in the previous section, and c depends only on size parameters of the grammar.

Proof. Solve the problem immediately for **iLMG**. Do the translation of an **iLMG** straight into **DTM** instead of using **RO-ATM** as an intermediate stage. The **DTM** takes an **iLMG** G and a string w as its input. It computes the maximum size of the RHS of clauses, and the parameters p and r. It then proceeds to simulate a **DTM** performing fixed recognition for G. It lays out a machine-readable representation of the finite state control device onto an extra tape; this is done in time dependent on the size of G only. It calculates an elapse counter, and enters a state *simulate* and further follows the instructions it has written on the extra tape. This final stage runs in time n^{3r} multiplied by a factor dependent only on the grammar. □

This is of course not to say that this 'factor dependent on the size of the grammar' is to be neglected—in fact it very much needs to be studied in detail, but if so, independent of the size and characteristics of the input. Under the **RAM** model, the exponent $3r$ is reduced to r.

A similar argument holds for the case of **GPSG** which is less interesting at this point, but is relevant to chapter **7**. A **GPSG** can be encoded in a **CFG**, and universal **CFG** recognition can be done in $\mathcal{O}(n^3)$ time. In this case it has been proven that the resulting 'constant' is of problematic proportions, but again, it does not depend on the size of the input. While Barton et al.'s result does not look at these size parameters individually, it is actually very worthwhile to know what quantitative properties of language are responsible for exponential growth of time complexity and which are not.

5.4 Complexity of bounded LML

Apart from **iLFP**, [Rou88] also introduces a calculus called **cLFP** which talks not about integer indices but about strings and concatenation. This is of course very similar to **LMG**; Rounds however decides to put a bound on the instantiations given to variables ranging over terminal strings. The result is that **cLFP** describes precisely the class **EXPTIME** of languages recognizable in deterministic exponential time.

5-14 deÆnition: cLFP. A **cLFP** formula is as an **iLFP** formula as defined in **4-15**, but replace the definition of a term by:

3. A TERM in **cLFP** is a terminal symbol, an individual variable, or t_1t_2 where t_1 and t_2 are terms.

   If t_1 and t_2 are terms, then $t_1 = t_2$ is a formula.

The semantics is modified as follows: a PREDICATE ASSIGNMENT is a set of predicates $A(w_1, \dots, w_{a(A)})$ over terminal strings. There are no axioms; the initial assignment ρ_0 is empty. The only interpretation rule that needs to be redefined is the one treating quantification:

3. $[\![\exists x.\phi]\!]\rho = \{\ \alpha \mid \exists w, |w| \leq n:\ \alpha(x \to w) \in [\![\phi]\!]\rho\ \}$.
4. $[\![t_1 = t_2]\!]\rho = \{\ \alpha \mid \alpha(t_1) = \alpha(t_2)\ \}$ where $\alpha(t)$ is the string obtained by substituting $\alpha(x)$ for every variable x occurring in t.

5-15 proposition. [Rou88] **cLFP** generates precisely the class of languages recognizable in time c^n on a **DTM**.

*

Without going into details, the following equivalent bounded semantics can be given for **LMG**.

5-16 deÆnition: bounded LMG. A generic **LMG** generates a string w under the BOUNDED INTERPRETATION if the notion of instantiation is modified as follows: an INSTANTIATED PREDICATE is of the form $A(w_1, \dots, w_p)$ where $w_1, \dots, w_p$ are of length at most n, where n is the size of the input string.

Now interpret the **LMG** as in definition **3-19**, but use the new notion of instantiation and perform interpretation relative to the input string w.

5-17 proposition. Bounded **LMG** and **cLFP** recognize the same class of languages.

*

This correspondence between bounded interpretations and **EXPTIME** will prove valuable when I look at the complexity of classes of **XG** in chapter **6**.

Conclusions to chapter 5

The results on alternation and the complexity of simple **LMG** presented here are simple modifications to analogous work done for the calculi **iLFP** and alternating Turing machines in [Rou88] and [CKS81]. The added value of this chapter is the format of **iLMG** and $\mathcal{S}$-**LMG**. The notation of **iLMG** is more elementary, that of **LMG** is more grammar-like, and the range of operations allowed in **iLMG** is minimal w.r.t. the rich logical language **iLFP**. So I have proved that there is an amount of redundancy in the **LFP** calculi.

On the other hand, the extra operations in **iLFP** make it a more versatile formalism. This will become clear when **iLFP** is used to characterize extraposition grammar in the next chapter.

By looking at the **RAM** model of computation, the concrete polynomial bounds can be improved drastically. Under the Turing machine model, the bounds were improved only slightly: in [Rou88], there was a small constant in addition to the exponent $2r$ that has been removed in the constructions in this chapter; for **MHG**, this leads to an $\mathcal{O}(n^{12})$ complexity instead of $\mathcal{O}(n^{20})$. The relative gain of this improvement by a constant polynomial factor is smaller for more complex formalisms.

It is regrettable that I didn't get Younger's deterministic Turing algorithm to work for **iLMG** in general; the only hint as to whether this should at all be expected solvable is the remark at the end of section **5.2** on the practical infeasibility of directly accessible memo tables. In chapter **7** and in the **Epilogue** I will get back on the issue of universal recognition.

Another unsolved question, that is more appropriate to mention here than at the end of chapter **3** is whether n-$\mathcal{S}$-**LMG** is strictly contained in $n+1$-$\mathcal{S}$-**LMG**. This seems to be nontrivial, because it would have remarkable consequences for bounded-degree subclasses of **PTIME**. A result using [CKS81] to prove the fact that $\mathcal{O}(n^r)$-time-**DTM** is included in r-**iLML** seems feasible, but is not a strict equivalence because for the converse, I have only been able to get the correspondence under the **RAM** model.

Chapter 6

Computational properties of extraposition grammar

This chapter will show first that **XG** in their original form from [Per81], *i.e.* under the semantics of definition **2-2**, describe any recursively enumerable language, which implies that even though the extraposition construction in **XG** may be "clear and concise", it cannot be considered minimal in expressive power.

In section **6.1**, I will show that when the number of brackets possibly introduced in a derivation is bounded, a recognition algorithm can be constructed that runs in exponential time. This is still not satisfactory if one insists that structural analysis should be tractable (hint **E** in the **Prologue**).

In **6.2**, the loose interpretation of definition **2-6**, in combination with the island rule **2-7**, is then shown to have fixed recognition problems in deterministic polynomial time. It has been shown in chapter **2** that **XG** under this interpretation are, from the perspective of linguistic structure, not less capable than with the original semantics.

The recognition algorithms in this chapter are built on the material developed in the previous chapter; to be precise, **iLFP** (or equivalently, simple **LMG**) allows one to count brackets, **cLFP** (or equivalently, bounded **LMG**) allows one to keep track of sequences of brackets with different labels.

6.1 Non-decidability of generic XG

Thus far, there are no results in the literature on the computational properties of extraposition grammar. This section shows that the original formulation of **XG** runs into problems because in general, it is not decidable whether a sentence is recognized by an a priori given **XG**. Before giving this result, it is useful to make a series of simplifications to the semantics of **XG**.

Recall the rule types of definition **2-1**:

1. $A \rightarrow X_1 X_2 \cdots X_n$ (a CONTEXT-FREE RULE)
2. $A \ldots B \rightarrow X_1 X_2 \cdots X_n$ (an ELLIPSIS RULE)

and their rewriting interpretation:

1. $\overline{\alpha} A \overline{\beta} \Rightarrow \overline{\alpha} X_1 X_2 \cdots X_n \overline{\beta}$
2. $\overline{\alpha} A \overline{\gamma} B \overline{\beta} \Rightarrow \overline{\alpha} X_1 X_2 \cdots X_n {<}\overline{\gamma}{>} \overline{\beta}$

6-1 proposition. The additional requirement in the interpretation of the ellipsis rule that $\overline{\gamma}$ contains *no nonterminal symbols*, does not change the semantics of **XG**.

Proof. Let d be the derivation of a terminal string from the start symbol S. Look at a step s in d where an ellipsis rule $A \ldots B \rightarrow C$ is used; this step is $\overline{\alpha} A \overline{\gamma} B \overline{\beta} \Rightarrow \overline{\alpha} C {<}\overline{\gamma}{>} \overline{\beta}$. Suppose $\overline{\gamma}$ contains nonterminal symbols. Then for each of these nonterminals, there will be a rule, applied *after* step s, to eliminate the nonterminal. Whatever such a rule is, it cannot depend on $\overline{\alpha}$ and $\overline{\beta}$ because $\overline{\gamma}$ is enclosed in brackets. Hence these rules could also have been applied *before* step s. Repeat the observation to obtain a derivation in which $\overline{\gamma}$ contains no nonterminal symbols. □

The modification in the semantics imposes a stronger restriction on the *order* of the application of rules in an **XG** rewriting sequence. Quite remarkably, this extra restriction seems to capture precisely what the brackets were used for in the original interpretation: requiring $\overline{\beta}$ in the interpretation of the ellipsis rule to be terminal seems to be the same as requiring the derivation graphs to have a planar representation as explained in section **2.1**. Hence the following *bracket-free* **XG** semantics.

6-2 deÆnition. BRACKET-FREE REWRITING in an **XG** is defined over unbracketed sequences $\alpha \in (N \cup T)^*$ of nonterminal and terminal symbols as in a context-free grammar. Let α and β be such sequences, let w be a terminal string and let a rule of type 1, 2 be in P, then we have, respectively:

1. $\alpha A \beta \Rightarrow \alpha X_1 X_2 \cdots X_n \beta$
2. $\alpha A w B \beta \Rightarrow \alpha X_1 X_2 \cdots X_n w \beta$

6-3 proposition. Bracket-free derivation is equivalent to the original definition in **2-1**.

Proof. One implication is obvious given proposition **6-1**. As to the other: suppose there is a bracket-free derivation, then there is a derivation according to the original definition (**2-2**). This is also simple: let d be a bracket-free derivation, then let the bracketed derivation d' apply the same sequence of rules. Then d' is a correct derivation, because whenever it applies an ellipsis rule in a step $\overline{\alpha}A\overline{\beta}B\overline{\gamma} \to \overline{\alpha}C{<}\overline{\beta}{>}\overline{\gamma}$, $\overline{\beta}$ must be a bracketing of a terminal string. So it only puts brackets *around terminal strings*; that is no nonterminals appear inside matching brackets. Since $\overline{\beta}$ is immediately next to the two nonterminals A and B, there cannot be any unmatched brackets in $\overline{\beta}$. □

In three steps, I will now show that the class **XL** of languages recognized by an **XG** includes the r.e. languages, and hence that recognition is not decidable. First, I show that **XL** is closed under homomorphism. I then show how the bracketing mechanism of **XG** derivations can be used to mimic a context-free derivation. Finally, I construct a bilinear **XG** that will recognize the intersection of two context-free languages. I will use the bracket-free **XG** derivation of definition **6-2**.

6-4 proposition. The languages recognized by **XG** are closed under homomorphism.

Proof. Analogous to the one for context-free grammars [HU79]; replace terminals a in the **XG** rules by their images $h(a)$. □

I now begin the construction that mimics the nonterminal rules of a context-free grammar in Chomsky normal form (see theorem **1-14**) using ellipsis rules.

6-5 definition. Let $G = (N, T, S, P)$ be a context-free grammar in CNF, that is, its productions are of the form $S \to \varepsilon$, $A \to a$ or $A \to BC$. Then define the set of **XG** productions P_X as follows: for every rule $A \to BC$ in P let P_X contain the rule $B \ldots C \to A$.

6-6 definition: sound labeling. Let G be a context-free grammar. A G-SOUND LABELING is a sequence $A_1 w_1 A_2 w_2 \cdots A_n w_n$ such that for each $1 \leq i \leq n$, $A_i \Rightarrow w_i$.

6-7 lemma. Let α be a G-sound labeling, $R \in P_X$ and $\alpha \overset{R}{\Rightarrow} \beta$. Then β is a G-sound labeling.

Proof. Let R be $B \ldots C \to A$; the general case is $\alpha = \gamma BuCv\delta$, and $\beta = \gamma Auv\delta$, where γ and δ are G-sound labelings, so δ is either empty or begins in a nonterminal. Because α is a G-sound labeling, $B \Rightarrow u$ and $C \Rightarrow v$. So $A \Rightarrow uv$, and β is a G-sound labeling. □

6-8 definition. Let G be a context-free grammar. A G-SOUND TERMINAL LABELING of a string $w = a_1 a_2 \cdots a_n$ is a labeling $A_1 a_1 A_2 a_2 \cdots A_n a_n$ such that for each $1 \leq i \leq n$, P contains the production $A_i \to a_i$.

6-9 proposition. Write $\alpha \overset{P}{\Longrightarrow} \beta$ to denote that α can be rewritten to β’ in zero or more steps using productions in P only. Let G be a context-free grammar and $w = a_1 a_2 \cdots a_n$ a terminal string. Then $A \Rightarrow w$ if and only if there is a G-sound terminal labeling α of w such that $\alpha \overset{P_X}{\Longrightarrow} Aw$.

Proof. The *if* part follows from the lemma. *Only if* is proved by induction on the depth of the context free derivation tree; let R be the rule applied in its top node;

Terminal case $R = A \to a$. Obvious: $Aa \overset{P_X}{\Longrightarrow} Aa$.

Nonterminal case $R = A \to BC$. If $A \Rightarrow w$ then there are u and v such that $w = uv$, $B \Rightarrow u$ and $C \Rightarrow v$. By i.h. there are terminal G-sound labelings β and γ such that $\beta \overset{P_X}{\Longrightarrow} Bu$ and $\gamma \overset{P_X}{\Longrightarrow} Cv$. Since P_X contains the production $B \ldots C \to A$, this means that $\beta\gamma \overset{P_X}{\Longrightarrow} BuCv \overset{P_X}{\Longrightarrow} Auv$. □

6-10 proposition. Let L_1 and L_2 be context-free languages. Then there is an **XG** X such that $\mathcal{L}(X) = L_1 \cap L_2$.

Proof. Let L_1 and L_2 be recognized by the **CNF** context-free grammars $G_1 = (N_1, T, S_1, P_1)$ and $G_2 = (N_2, T, S_2, P_2)$, respectively, and assume w.l.o.g. that $N_1 \cap N_2 = \emptyset$. Then construct an extraposition grammar $X = (N_1 \cup N_2 \cup \{\Sigma, \Phi\}, T, \Sigma, P)$ as follows:

1. P contains $\Sigma \to \varepsilon$ if the empty string is in both L_1 and L_2.
2. P contains the rule $\Sigma \to \Phi S_1$.
3. P includes the nonterminal rules of G_1.
4. If P_1 contains $A \to a$ and P_2 contains $B \to a$, then P contains the rule $A \to Ba$.
5. P includes $P_X(G_2)$.
6. P contains the rule $\Phi \ldots S_2 \to \varepsilon$.

Let d be any X-derivation of w. Then it is easy to see that there is a derivation d' of w that applies the rules of type 1 first, then rules of type 2, and so forth.

Let $w = a_1 a_2 \cdots a_n$ be nonempty. If d is a derivation of w, then d' starts in the derivation $S \to \Phi S_1 \overset{*}{\Longrightarrow} \Phi B_1 a_1 B_2 a_2 \cdots B_n a_n$ of a terminal G_2-sound labeling using only rules of type 2, 3 and 4. From this initial segment of d' one immediately has a G_1-derivation of w. Because d' will end in an application of rule 6, one must have $B_1 a_1 B_2 a_2 \cdots B_n a_n \overset{P_X}{\Longrightarrow} S_2 w$, which by proposition **6-9** is equivalent to saying that G_2 derives w.

The reverse implication is now obvious. □

6-11 corollary. The class **XL** of languages recognized by an extraposition grammar includes all recursively enumerable languages.

Proof. It has now been proved that **XL** is closed under homomorphism and contains the class $\{L_1 \cap L_2 \mid L_1 \text{ and } L_2 \text{ context-free}\}$; it is a standard result (see *e.g.* [Gin75] p. 125) that any such class includes all r.e. languages. □

6.2 Bounded and loose XG recognition

To obtain the promised tractability results for classes of extraposition grammars, I will again develop an alternative, equivalent way of looking at **XG** derivation; now in terms of a context-free grammar that generates bracketed strings.

6-12 deÆnition: corresponding CFG. Let $G = (N, T, S, P_1 \cup P_2)$ be an extraposition grammar, where P_1 contains context-free rules only, and P_2 contains only ellipsis rules. Then the CORRESPONDING CONTEXT-FREE GRAMMAR $\mathcal{CF}(G)$ is the context free grammar $(N, T \cup \{\ <_B,\ {}_B{>} \mid B \in N\ \}, S, P_1 \cup P_3)$ where for every rule $A \ldots B \to \alpha$ in P_1, include in P_3 the two rules

(i) $A \quad \to \quad \alpha <_B$
(ii) $B \quad \to \quad {}_B{>}.$

6-13 proposition. Let G be an extraposition grammar. Define the rewriting operation $\rightsquigarrow$ over bracketed terminal strings as follows: for every $B \in N$,

$$\overline{u}<_B\ v\ {}_B{>}\overline{w} \rightsquigarrow \overline{u}v\overline{w}$$

(where v contains no brackets).

Then G derives a string w iff $\mathcal{CF}(G)$ derives a $\overline{w}$ such that $\overline{w} \overset{*}{\rightsquigarrow} w$.

Proof. The definition of $\rightsquigarrow$ is merely a formal characterization of the idea that G derives w if and only if $\mathcal{CF}(G)$ derives a *balanced* bracketing of w.

Clearly, if there is a G-derivation of w, then there is also a $\mathcal{CF}(G)$-derivation of a balanced bracketing $\overline{w}$ (the bracketing is identical to that in the **XG** derivation according to definition **2-1**).

As to the converse, if there is a derivation of a balanced bracketing $\overline{w}$ in $\mathcal{CF}(G)$, then it must be shown that the context-free derivation can be transformed such that for each pair of matching brackets, the rules $R_1 : \quad A \to \alpha <_B$ and $R_2 : \quad B \to {}_B{>}$ applied to produce these brackets, follow each other immediately. This then corresponds to applying the single **XG** rule $A \ldots B \to \alpha$.

It is straightforward to see that this can indeed be done. Order the bracket pairs rightmost–innermost, which is to say the *opening* brackets are ordered from right to left. Repeatedly take the next pair from this list, and write out the least number of rules needed to introduce the brackets. It must be shown that to do this, no other new brackets need to be introduced. This cannot be because of a rule of type **(ii)**, because it has no nonterminals on its RHS, so its application can be postponed as long as we want. Rules of type **(i)** cannot be needed either, because applying one would introduce another opening bracket right of the opening bracket we want to introduce. But in a rightmost–innermost rewriting strategy, that bracket must have already been treated. □

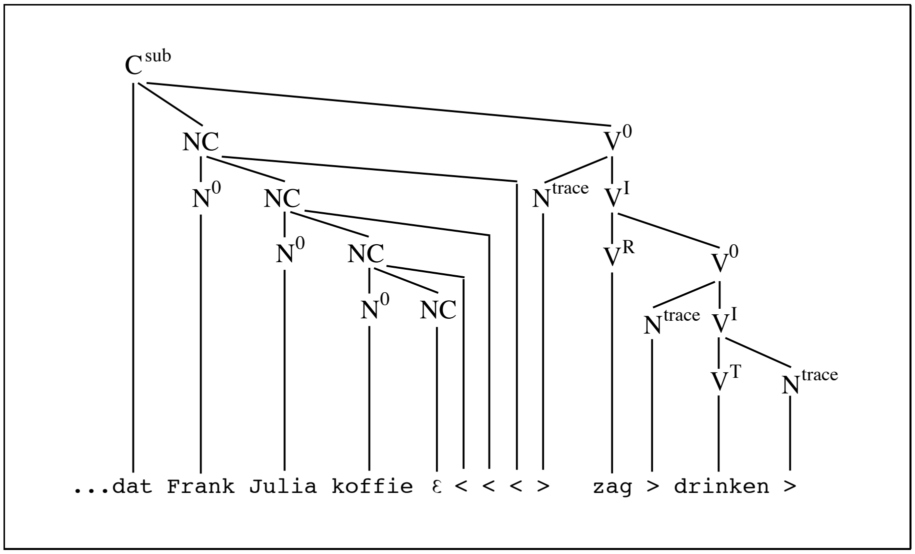


Figure 6.1: Derivation in the corresponding context-free grammar.

6-14 example. In the example grammar for Dutch from section **2.1**, there is only one ellipsis rule; therefore only one bracket symbol will be required. Rule [5] is replaced with

[5] $\text{NC} \rightarrow \text{N}^0\ \text{NC}\ <$

and the following rule is added:

[5'] $\text{N}^{\text{trace}} \rightarrow\ >$

The context-free derivation corresponding to the derivation in *figure 2.4* is shown in *figure 6.1*.

*

Given this context-free model, it is considerably easier to design a simple recognition algorithm that is similar to known algorithms for context-free grammars and tuple-based extensions as illustrated in the previous chapters. The idea is straightforward: augment a left-recursion-proof memoing recursive descent algorithm for context-free grammars with a mechanism that ensures that the derived strings have a balanced bracketing. Note that it should be able to perform this task taking as its input the *original string* w, and not its bracketed version $\overline{w}$.

For the simple case that there is only one ellipsis rule, this can be done by counting the number of unmatched brackets for each recognized constituent. This case includes the grammar for Dutch subordinate clauses.

Because this section is interested primarily in computational complexity, I will give translations to **LFP** calculi rather than concrete algorithms. The step to such concrete algorithms is analogous to the step made for the examples in the previous chapters. To obtain grammars with favourable recognition properties, the following restriction must be made:

6-15 deÆnition. Let $G = (N, T, S, P)$ be an arbitrary extraposition grammar; G is *linear bounded* (under the loose interpretation) if for any bracketing $\overline{w}$ of a string w, when $S \Rightarrow \overline{w}$ (under the loose interpretation) then the number of bracket pairs in $\overline{w}$ is not greater than the length of w.

$*$

Instead of checking the linear boundedness property of a grammar, one will usually check some sort of *overt extraposition* property which says that applying an ellipsis rule $A \ldots B \rightarrow C$ necessarily immediately introduces one or more terminal symbols; *i.e.* fillers cannot be empty. Equivalent properties in the literature often also follow from *lexicalization*.

6-16 proposition. If an extraposition grammar G has only one ellipsis rule and is linear-bounded, then there is an algorithm that decides G-membership for a given string of length n in $\mathcal{O}(n^7)$ time.

Proof. Let the input string be $w = a_0 a_1 \cdots a_{n-1}$. Translate the grammar G to a recursive **iLFP** scheme Φ_G using the rules in *figure 6.2*.[1]

Intuitively, it is straightforward to check that a formula $A(i, j, l, r)$ is true whenever A recognizes a bracketing $\overline{v}$ of $v = a_i \cdots a_{j-1}$, where l is the number of unmatched closing brackets in $\overline{v}$, and r is the number of unmatched opening brackets. Now the context-free grammar derives a balanced bracketing of a string $w = a_0 a_1 \cdots a_{n-1}$ if and only if the formula $\phi = \mu\Phi_G.\ S(0, n, 0, 0)$ is true under the assignment $\rho_0(w)$.

The only perhaps non-straightforward clause is that for $A \rightarrow BC$; the two disjuncts correspond to the two situations (6.1) and (6.2).

$$\begin{array}{ccc} \boxed{\quad B \quad}\ \boxed{\quad\quad C \quad\quad} & \longrightarrow & \boxed{\quad\quad A \quad\quad} \\ \underbrace{>>}_{l_1}\ \underbrace{<<}_{r_1}\ \underbrace{>>>}_{l_2}\ \underbrace{<<}_{r_2} & & \underbrace{>>>}_{l_1+l_2-r_1}\ \underbrace{<<}_{r_2} \end{array} \tag{6.1}$$

$$\begin{array}{ccc} \boxed{\quad\quad B \quad\quad}\ \boxed{\quad C \quad} & \longrightarrow & \boxed{\quad\quad A \quad\quad} \\ \underbrace{>>}_{l_1}\ \underbrace{<<<}_{r_1}\ \underbrace{>>}_{l_2}\ \underbrace{<<}_{r_2} & & \underbrace{>>}_{l_1}\ \underbrace{<<<}_{r_2+r_1-l_2} \end{array} \tag{6.2}$$

In checking intuitively that the formula must be true, I overlooked that the **iLFP** semantics of the arithmetical operations is modulo n arithmetic. Therefore this only holds if it is known in advance that the number of unmatched brackets on either side will never exceed n. This prerequisite is satisfied by the linear-boundedness property.

Finally, the scheme's largest rule contains 9 variables, so a naïve implementation of the **iLFP** scheme would give an $\mathcal{O}(n^9)$ algorithm. However, careful study of the

[1] This scheme contains proper terms within atomic predicates, and the operators $+$ and $-$. These can be translated to bare **iLFP** by adding clauses with at most 3 variables.

CFG rule	enhanced **iLFP** clause		
$A \rightarrow a$	$A(i,j,0,0)$	$\Leftrightarrow$	$a(i,j)$
$A \rightarrow \varepsilon$	$A(i,i,0,0)$	$\Leftrightarrow$	**true**
$B \rightarrow >$	$B(i,i,1,0)$	$\Leftrightarrow$	**true**
$A \rightarrow BC$	$A(i,k,l_3,r_3)$	$\Leftrightarrow$	$\exists j,\ l_1,l_2,r_1,r_2.$ $B(i,j,l_1,r_1) \wedge C(j,k,l_2,r_2) \wedge$ $((r_1 \leq l_2 \wedge l_3 = l_1 + l_2 - r_1 \wedge r_3 = r_2) \vee$ $(r_1 > l_2 \wedge l_3 = l_1 \wedge r_3 = r_2 + r_1 - l_2))$
$A \rightarrow C <$	$A(i,j,l,r+1)$	$\Leftrightarrow$	$C(i,j,l,r)$

Figure 6.2: Translation scheme for **XG** with one ellipsis rule.

rule shows that in each of the inner disjuncts, two variables are fixed given the other seven. □

It is now straightforward to make similar constructions for the general case and for the loose semantics.

6-17 proposition. If G is an arbitrary linear-bounded **XG**, then membership for G can be decided in deterministic exponential time in terms of the length of the input.

Proof. (sketch) Instead of **iLFP**, use **cLFP**, and instead of keeping track of the *number* of brackets, look at arbitrary *sequences* of different brackets. Because of linear-boundedness, the length of the sequences is bounded by the length of the input. □

The loose semantics of definition **2-6** can be given a corresponding **CFG** equivalent along the same lines as in proposition **6-13**. In this case, counting brackets is sufficient again, because brackets need to be matched only if they have the same label.

6-18 proposition. Let G be an extraposition grammar. Define the rewriting operation $\leadsto$ over bracketed terminal strings as follows: for every $B \in N$, if $\overline{v}$ does not contain any B-brackets $<_B$ and $_B>$, then

$$\overline{u} <_B \overline{v}\ _B> \overline{w} \leadsto \overline{u}\ \overline{v}\ \overline{w}$$

Then G loosely derives a string w iff $\mathcal{CF}(G)$ derives a $\overline{w}$ such that $\overline{w} \overset{*}{\leadsto} w$.

Proof. Entirely analogous to **6-13**. □

The calculus of *figure 6.2* can now be extended by introducing a pair of arguments l, r for each of the different annotated bracket pairs $<_B$ and $_B>$; it will ensure that the derived bracketed string is balanced only w.r.t. each individual pair of brackets. The complexity of the resulting algorithm becomes $\mathcal{O}(n^{3+4m})$ where m is the number of bracket pairs.

6-19 proposition. If G is an arbitrary linear-bounded **XG** under loose derivation, then membership for G under loose derivation can be decided in deterministic polynomial time in terms of the length of the input.

*

Finally, the **iLFP** translation can be extended by specifying the deduction rule for the island production in *figure 6.3*, which requires that the numbers of unmatched opening and closing brackets in C are both zero; the example assumes 1 ellipsis rule as in *figure 6.2*, but can be extended to take arguments for each bracket label.

Island rule	**iLFP** clause
$A(B) \rightarrow C$	$A(i,j,0,0) \ \Leftrightarrow\ C(i,j,0,0)$

Figure 6.3: Translation for the island rule.

Conclusions to chapter 6

This chapter concludes the presentation of **XG** started in chapter **2**. No results were previously known on the computational properties of **XG**, and all material in this chapter is original.

Not discussed is how a concrete predicate descent algorithm can be constructed for **XG** under loose derivation. Although this is rather straightforward, and the complexity heuristics is preserved, such an algorithm allows practical implementations less directly than an algorithm for **LMG**. The most important reason is that the 'bracket' arguments to items must be guessed. In chapter **7**, the cost of such guessing is cut down by trying only a limited selection of items based on properties of the grammar. It is unclear how this should be done for **XG** in a top-down parsing technique. It is not unlikely that a **GLR**-based algorithm could be constructed that takes account of numbers of unmatched brackets more efficiently.

Still left to investigate are the precise formal power of **XG** under the bounded and loose semantics, and its closure and pumping properties. Both loose and bounded **XL** are clearly closed under homomorphism, so they will not correspond to **PTIME** and **EXPTIME** like classes of **LMG**. Since brackets are eliminated pairwise, some form of linearity is present that resembles the linearity in tuple grammars, which points more in the direction of pumpability. The brackets also indicate a connection to DYCK LANGUAGES (see e.g. [VSWJ87]), which may well provide an answer to this question.

Some of the intuitions given by **XG**'s derivation graphs, and the computational consequences of using the different constructions investigated in chapter **2** will be used in chapter **10**. **XG** will also come back in chapter **8** when scrambling in German is discussed.

Chapter 7

Trees and attributes, terms and forests

Until this point, the discussion has been in a very formal setting, and has concentrated on abstract properties of very mathematically oriented formalisms. This chapter is a pivot, in the sense that all following chapters will be of a more linguistic, *descriptive* nature. Because *computational tractability* is one of the desiderata of the **Prologue** and chapter **1**, it first needs to be investigated if and how the formal tractability of the formalisms discussed so far survives when these formalisms are applied in real-world descriptive linguistic systems.

This chapter will be brief and casual, but knowledge of the subjects addressed here is necessary to understand the motivation of various choices made in the descriptive-linguistic part **III** of this thesis. I will split up the question of tractability in practice into two parts, one of which is twofold;

1. Can the recognition and parsing solutions proposed in the previous chapters be used in applications that can perform realistic tasks on medium-size modern computer systems?

2a. How can linguistic analysis be extended from a purely *structural* nature, to include morphological, and perhaps semi-semantic features?

2b. Do the phases **1** and **2a** yield structures based on which any remaining (semantic) analysis can be carried out efficiently? In particular, if tractability implies pushing non-local *ambiguity* to further stages of processing, will those further stages not suffer an equivalent amount of problems?

In the first section of this chapter, I briefly introduce various methods of describing trees, sets of trees (*forests*), attributes, and combinations of those, used in various approaches to descriptive and computational linguistics; some examples are included of what types of attributes are frequently used. In section **7.2** I briefly discuss the predicate-descent based prototype **LMG** parser that I built in 1996, and discuss some of its performance properties. The next section looks at the parsing problem from the broader perspective of an elaboration on the notion of ambiguity and OFF-LINE PARSABILITY, including an elaboration on possible improvements using efficient context-free parsing techniques such as **GLR** [Rek92] [Tom86] to reduce on the excessive memory consumption of a straightforwardly implemented predicate-descent algorithm. This multi-stage approach appears again in the next section, **7.4**, where methods of attribute evaluation over the ambiguous *parse forests* produced by a parser are discussed.

7.1 Annotated grammars

As there are different methods for the description of underlying structure, there is a number of different approaches to describing attributes over a structural backbone. Such attributes can be morphological properties of words and phrases, such as number, person and case; they can be selectional properties such as the property of being *animate* required by verbs such as *to walk*, or representations of the meaning of a word or phrase.

I will now briefly sketch two of the most common ways of describing linguistic structures with attributes; on a limited scale, I will start bringing in the influence of the desiderata of the **Prologue**, most notably computational tractability (**E**), full localization (**F**) and separation of structure and finite attributes (**C**).

Trees, terms and forests

Most of the work on computationally efficient treatment of attributes in the literature assumes either a realm of *feature structures* (discussed further below) or is based on a projective formalism, *i.e.*, defines attributes over the trees of a context-free grammar.

In this thesis, I have looked at two extensions of **CFG**. Extraposition grammar has a derivational structure that cannot be straightforwardly transformed to one that looks like the trees produced by a **CFG** under their derivational semantics; therefore I will not take **XG** into consideration in this chapter.

Context free grammar and the tuple-based formalisms in chapter **3** on the other hand have similar derivation structures. These structures have so far only informally been associated with the derivational interpretations **1-8**, **3-3** and **3-19**, and indeed, this association is fairly trivial. The important points to stress are

1. A single proof that a grammar derives a string can be uniquely represented by a tree whose nodes are bearing a LABEL that identifies a grammar rule, or partially identified by a label just referring to a nonterminal or terminal symbol.

2. A string is thus associated with a *set* of trees, that is empty if the string is not recognized by the grammar, and otherwise contains one tree for each derivation of the string.

An abstract way of looking at such labelled trees is to think of them as TERMS, and of the labels as *functions* that construct terms from smaller terms.

7-1 deÆnition: free terms. Let F be a set of FUNCTIONS. Then the set $\mathbf{T}(F)$ of FREE TERMS OVER F is a set of syntactic forms, satisfying:

1. Every element of F is in $\mathbf{T}(F)$;

2. If $f \in F$ and $t_1, \ldots, t_n \in \mathbf{T}(F)$ then $f(t_1, \ldots, t_n) \in \mathbf{T}(F)$.

*

The simple grammar in *figure 7.1* will be used frequently in this chapter to illustrate attributed trees. The derivation tree of *the man walks* can be represented uniquely in a term syntax as $\mathrm{V}^0(\mathrm{N}^0(\texttt{the man}), \mathrm{V}^\mathrm{I}(\texttt{walks}))$. Since there are two rules generating *men* as a noun phrase however, the sentence *the men walk* has two derivations, which both have the tree $\mathrm{V}^0(\mathrm{N}^0(\texttt{the men}), \mathrm{V}^\mathrm{I}(\texttt{walk}))$, but differ in the more precise term syntax with rule identifiers as labels: $r_1(r_8, r_4)$ and $r_1(r_8, r_5)$. When a word or string occurs twice in a sentence, it is typically fruitful to distinguish between the two occurrences. Therefore, more complex representations may also include sentence indices.

$[r_1]$	V^0	→	N^0 V^I	
$[r_2]$	V^0	→	V^0 `and` V^0	
$[r_3]$	V^I	→	`walks`	{third person singular}
$[r_4]$	V^I	→	`walk`	{first or second person singular}
$[r_5]$	V^I	→	`walk`	{plural}
$[r_6]$	N^0	→	`you`	{second person singular}
$[r_7]$	N^0	→	`the man`	{third person singular}
$[r_8]$	N^0	→	`the men`	{third person plural}

Figure 7.1: Labelled **CFG** with informal annotations

In chapter **4**, a *parser* was defined to be a machine that, given a string, outputs all structural analyses of that sentence according to a given grammar. For grammar formalisms that output trees, a representation of all derivation trees is called a *forest*. I will use the following definitions:

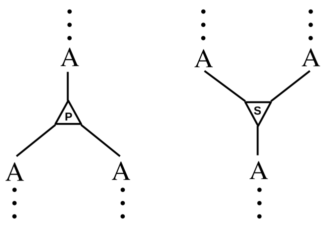


Figure 7.2: Packing and sharing.

7-2 definition: forests. A FOREST is a set of trees. A PACKED FOREST is an economical representation of such a set which makes use of PACKING NODES that can attach two or more subderivations to one parent node. A SHARED FOREST is a representation that can attach one subderivation to two or more parent nodes. This is shown schematically in *figure 7.2*.

A representation of a forests is called MAXIMALLY PACKED or MAXIMALLY SHARED if it has no equivalent representation with a larger number of packing and sharing nodes, respectively.

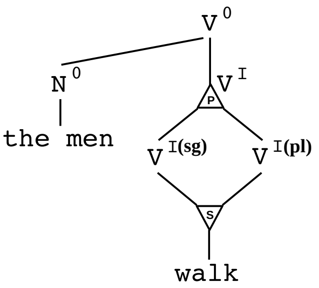


Figure 7.3: An example of a packed and shared forest representation.

*

Packed forests can be represented in a term syntax by adding a function **pack** to represent packing nodes; the two derivations of the sentence *men walk* can be represented economically as

(7.1) $r_1(r_8, \mathbf{pack}(r_4, r_5))$

Only one of the terms in this forest representation corresponds intuitively to a correct sentence—one of the analyses used rule r_4 to produce *walk*, which disagrees with the 3rd person plural of *men*. This will be formalized below.

A similar, but significantly more involved approach with abstraction over variables in a sort of **let** construction can be applied to model sharing in a term syntax—I will not explain it here because it will not be needed in this thesis. Some concrete systems manipulating terms, such as EPIC [WK96], have sharing built in explicitly, *i.e.* at a meta level.

One typically looks at forests that are maximally shared w.r.t. a representation *including sentence indices*, that is: identical substrings occurring at different places in the represented sentence are *not* to be shared. *Cf.* the following sentence, where two occurrences of the same word *walk* will get different annotations.

(7.2) You walk and the men walk

Annotated tuple grammars

Terms do not only serve well as a way of talking about derivation trees independently of the grammar formalism or parser that is used to produce them—they are also used to represent, possibly complex, *attributes* over the syntactic constructions of a language.

I will use the notion of a term in this section to capture two different styles of annotating grammar, *feature structures* and *grammars over finite lattices*, in a single framework. To do this smoothly, a method is required of distinguishing meaningful terms from terms that will not be used.

7-3 deÆnition. Let $\mathcal{S}$ be a set of basic types of SORTS. Then the set $\mathcal{T}(\mathcal{S})$ of FUNCTIONAL TYPES OVER $\mathcal{S}$ is the smallest set containing

1. Each sort $A \in \mathcal{S}$; that is $\mathcal{S} \subseteq \mathcal{T}(\mathcal{S})$;
2. The type $\tau = \tau_1 \times \cdots \times \tau_n \to \sigma$ where $n \geq 0$ is the ARITY of τ, and σ, $\tau_1, \ldots, \tau_n \in \mathcal{T}(\mathcal{S})$.

Let F be a set of functions and $\phi : F \to 2^{\mathcal{T}(\mathcal{S})}$ be a function assigning a set of types to each function in F, each of the same arity. Then the set $\mathbf{T}(F)$ of TYPED TERMS OVER $\mathcal{S}$ and the function $\tau : \mathcal{T}(\mathcal{S})$ are constructed inductively as follows:

1. If a function f has arity 0, then $\phi(f)$ contains only basic types $\tau \in \mathcal{S}$. Then $f \in \mathbf{T}(F)$ is called a CONSTANT.
2. If $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{S})$ and

   $$\tau(t_1) \times \cdots \times \tau(t_n) \to \sigma \in \phi(f),$$

   then the term

   $$t = f(t_1, \ldots, t_n)$$

   is in $\mathbf{T}(F)$, and $\sigma \in \tau(t)$.

Let $\tau \in \mathcal{T}(\mathcal{S})$ be a type; then $\mathbf{T}(\tau)$ is the set of terms t in $\mathbf{T}(F)$ that have the type τ, that is $\tau \in \tau(t)$.

A set of typed terms is called a FIRST ORDER SIGNATURE if ϕ is a function, that is $\phi(f)$ is a singleton for each $f \in F$.

*

Both ways of constructing terms corresponding to single derivation trees of a **CFG** or **LMG** discussed above form a first order signature if the nonterminals of the grammar are taken as the set of basic types. If sets of terms are represented using the functions **pack** and **share**, these functions are *polymorphic*, *i.e.* they must be assigned more than one type; **pack** for example has the type $(A, A) \to A$ for every basic type, or nonterminal, A.

7-4 definition. An ANNOTATED **LMG** is a tuple $(G, F, \mathcal{S}, \phi, \nu, C)$ whose first component $G = (N, T, V, S, P)$ is an **LMG**, $\mathcal{S}$ is a set of basic sorts, $\nu : N \to \mathcal{S}$ assigns a basic type to each nonterminal of the grammar, and the operator C assigns a CONSTRAINT to each of the clauses, or productions, of the grammar: if P contains a clause R of the form

$$A(\cdots) \;\mathbf{:-}\; B_1(\cdots), \ldots, B_m(\cdots).$$

then

$$C(R) \subseteq \mathbf{T}(\nu(A)) \times \mathbf{T}(\nu(B_1)) \times \cdots \times \mathbf{T}(\nu(B_m)).$$

7-5 definition. An annotated **LMG** $(G, F, \mathcal{S}, \phi, \nu, C)$ is said to RECOGNIZE a string w if w is in the language recognized by G, and there is at least one derivation d of w and a way of associating each node in d with a term t such that for each node, the tuple consisting of the term t and the terms $t_1, \ldots, t_m$ associated to the daughter nodes are in the constraint relation $C(R)$ where R is the rule applied at the node.

Grammars over a finite lattice

I will assume for now, and will substantiate in part **III**, that on the basis of a method like the **LMG** system it will indeed be possible to provide sufficiently general linguistic descriptions with a bounded dependency domain (definition **1-12** on page 23). While some attributes of syntactic structures such as anaphoric linking and semantic interpretations seem to need a recursive, hence infinite domain, it seems possible—without claiming that this would be trivial—for a first phase of syntactic processing[1] to maintain, at each node in the derivation, only a bounded set of attributes ranging over boundedly many values.

Such grammars are captured in the following definition of a *tuple grammar over a finite lattice* (**TGFL**), which is the **LMG** equivalent of the ATTRIBUTE GRAMMARS OVER FINITE LATTICES (**AGFL**) developed at the university of Nijmegen as part of the GRAMMAR WORKBENCH [Kos91] [NK92].[2] Each node in a **TGFL** derivation is associated with a finite number of AFFIX VALUES ranging over finitely many values; this is formalized in the following definition.

7-6 definition. A TUPLE GRAMMAR OVER A FINITE LATTICE is an annotated **LMG** $(G, F, \mathcal{S}, \phi, \nu, C)$ where $G = (N, T, V, S, P)$ and

[1] Glossing over languages with a complex morphological structure, the analysis of which will be left to a syntactic *pre*-processing stage—I assume a finite dictionary given at the beginning of a sentence processing stage, perhaps generated by such a pre-processing phase for that sentence only.

[2] A study on the complexity of a formalism similar to **AGFL**, called AGREEMENT GRAMMAR, is found in [BBR87]—but is subject to the same objections as I raised in chapter **3**. It is claimed that the universal recognition problem for agreement grammars is NP-complete, hence the best possible algorithm is probably exponential—*in the size of the attribute domains*. As in the other constructions in the book, the grammar and input in the NP-completeness proof are not taken to be independent.

$$\begin{array}{rcrcl}
 & & D & = & \{D_{\text{person}}, D_{\text{number}}\} \\
 & & D_{\text{person}} & = & \{1, 2, 3\} \\
 & & D_{\text{number}} & = & \{\mathtt{sg}, \mathtt{pl}\} \\
 & & \nu(\mathrm{V}^0) & = & \mathrm{EMPTY} \\
 & & f_{\mathrm{V}^0} & = & e : \mathrm{EMPTY} \\
\nu(\mathrm{V}^{\mathrm{I}}) & = & \nu(\mathrm{N}^0) & = & \mathrm{VAGR} \\
f_{\mathrm{V}^{\mathrm{I}}} & = & f_{\mathrm{N}^0} & = & \mathit{vagr} : (D_{\text{person}} \times D_{\text{number}}) \to \mathrm{VAGR}
\end{array}$$

$$\begin{array}{lllll}
[r_1] & \mathrm{V}^0 & & \to & \mathrm{N}^0 : \mathit{vagr}(p, n),\ \mathrm{V}^{\mathrm{I}} : \mathit{vagr}(p, n) \\
[r_2] & \mathrm{V}^0 & & \to & \mathrm{V}^0\ \mathtt{and}\ \mathrm{V}^0 \\
 & & & & \\
[r_3] & \mathrm{V}^{\mathrm{I}} : \mathit{vagr}(3, \mathtt{sg}) & & \to & \mathtt{walks} \\
[r_4] & \mathrm{V}^{\mathrm{I}} : \mathit{vagr}(1|2, \mathtt{sg}) & & \to & \mathtt{walk} \\
[r_5] & \mathrm{V}^{\mathrm{I}} : \mathit{vagr}(1|2|3, \mathtt{pl}) & & \to & \mathtt{walk} \\
 & & & & \\
[r_6] & \mathrm{N}^0 : \mathit{vagr}(2, \mathtt{sg}) & & \to & \mathtt{you} \\
[r_7] & \mathrm{N}^0 : \mathit{vagr}(3, \mathtt{sg}) & & \to & \mathtt{the\ man} \\
[r_8] & \mathrm{N}^0 : \mathit{vagr}(3, \mathtt{pl}) & & \to & \mathtt{the\ men}
\end{array}$$

Figure 7.4: **AGFL** for agreement in simple English sentences

1. $\mathcal{S}$ contains, in addition to a type $\nu(A)$ for each nonterminal $A \in N$ (these need not be different for two different nonterminals), a finite number of additional types called AFFIX DOMAINS. Let $\mathcal{D}$ be the set of affix domains.

2. For each nonterminal $A \in N$, the set of functions F contains a TUPLE CONSTRUCTOR f_A of type

   $$\phi(f_A) = D_1 \times \cdots \times D_n \to \nu(A)$$

   where $D_1, \ldots, D_n \in \mathcal{D}$ are affix domains and $n \geq 0$.

3. There are finitely many other functions in F; these are called AFFIX VALUES, are constant and have a type $d \in D \in \mathcal{D}$.

A constraint $C(R)$ is allowed, for each subterm of the terms associated with the daughter nodes and the parent node, to

1. equate it to another subterm; and/or

2. restrict its value to a set of affix values.

*

An example of a **TGFL** is the formal equivalent of the annotated **CFG** on page 131 in *figure 7.4*. Since its backbone grammar is a **CFG**, it is actually a notational variant of an **AGFL**. Grammars over finite lattices can be straightforwardly extended to allow more complex terms, but keeping the set of types free of *cycles*, so that a node in a derivation can be associated with only a finite number of possible terms. An example where this is useful is to describe *subcategorization*: every verb has morphological properties *verb form*, *number* and *person*, and in the lexical information associated with a verb like *to want* such information appears twice: once for the verb itself, and once for its complement, which necessarily appears with infinitive morphology. This is a bounded-domain version of the well-known *feature structures*, which will be discussed after the following remark.

7-7 remark: coding. A key property of finitely annotated grammars, in a formal setting, is that they can be expanded to an equivalent grammar without attributes. This is done by first encoding the nonterminals in the associated terms, resulting in a grammar with has $|N| = 1$. Then, a new grammar is constructed whose set of nonterminals is $N' = \mathbf{T}(F, \mathcal{S})$. The productions P of this grammar are those that satisfy the constraints of the original, annotated grammar. This is called GRAMMAR CODING.

From a more practical point of view, it has been argued that such a construction is not very valuable, because the resulting grammar has a size proportional at least to the third power of the size of the attribute domain $\mathbf{T}(F)$, which is prohibitive in practice. Moreover, extracting an annotated parse forest according to the original grammar is nontrivial.

In other words, for the purpose of this chapter, formal complexity reasoning does not provide enough detail.

Feature structures

A popular method of associating linguistic structure with attributes is the use of FEATURE STRUCTURES and UNIFICATION. Feature formalisms are so powerful, that it is theoretically possible to simulate any grammar by a feature grammar over a backbone that has only one nonterminal. This is essentially done in **HPSG**. I will instead concentrate on feature formalisms that are constructed on top of a tuple grammar, typically a **CFG**. This grammar is called the STRUCTURAL BACKBONE of the feature grammar.

Feature structures are obtained by relaxing the definitions for **TGFL** to allow more arbitrary types.

7-8 deÆnition. A FEATURE GRAMMAR is as a **TGFL**, but with the following modifications

1. The elements of D are called FEATURE DOMAINS.

2. For each basic type $\tau \in \mathcal{S}$, the set of functions F contains finitely many constants $c : \tau$, and at most one non-constant function f_τ of type

   $$\phi(f_\tau) = D_1 \times \cdots \times D_n \to \tau$$

   where $D_1, \ldots, D_n$ are feature domains and $n \geq 1$.

$*$

The essential difference between this more general definition and a **TGFL** is that it allows structures of arbitrary depth. A grammatical phenomenon that can be argued to need arbitrarily deep structures is *selection* or *subcategorization*. Although the number of complements selected by known verbs is limited to 4, including the subject, it can be argued that a language that has verbs selecting for more complements must not a priori be excluded.

Feature structures are traditionally displayed as matrices in which the "tuple constructors" are represented as square brackets, and the arguments of these constructors are labelled—these labels are indices to the argument positions of the constructor functions. An example of a verb with a long subcategorization list is *to trade something to someone for something*.

$$(7.3) \qquad \textit{trades} \to \left[\begin{array}{ll}
\text{cat:} & \mathtt{verb} \\
\text{head:} & \left[\begin{array}{ll}
\text{form:} & \mathtt{finite} \\
\text{tense:} & \mathtt{present} \\
\text{agr:} & \boxed{1}\left[\begin{array}{ll} \text{person:} & 3 \\ \text{number:} & \mathtt{sg} \end{array}\right] \\
\text{subject:} & \left[\begin{array}{ll}
\text{cat:} & \mathtt{np} \\
\text{animate:} & \mathtt{true} \\
\text{head:} & \left[\begin{array}{ll} \text{agr:} & \boxed{1} \\ \text{case:} & \mathtt{nom} \end{array}\right]
\end{array}\right]
\end{array}\right] \\
\text{subcat:} & \left[\begin{array}{ll}
\text{first:} & \left[\begin{array}{ll}
\text{cat:} & \mathtt{np} \\
\text{animate:} & \mathtt{false} \\
\text{head:} & [\text{case: } \mathtt{acc}]
\end{array}\right] \\
\text{rest:} & \left[\begin{array}{ll}
\text{first:} & \left[\begin{array}{ll} \text{cat:} & \mathtt{pp} \\ \text{prep:} & \mathtt{to} \end{array}\right] \\
\text{rest:} & \left[\begin{array}{ll}
\text{first:} & \left[\begin{array}{ll} \text{cat:} & \mathtt{pp} \\ \text{prep:} & \mathtt{for} \end{array}\right] \\
\text{rest:} & \mathtt{nil}
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

Although I will argue below that in the context of this thesis, it is a rather obvious choice to prefer finite attribute grammars over feature structures, feature structures are discussed briefly here for two important reasons: (**i**) the typical way of describing long-distance dependencies (w.r.t. the surface order) in feature-driven formalisms needs to be compared with the methods proposed in this thesis and (**ii**) feature grammars offer methods that enable a form of LEXICALIZATION that cannot be applied trivially without the full capacity of the structure sharing and unification present in feature feature-based systems.

7-9 example: slash features. Whereas the formalisms in part **I** of this thesis aim at modelling long-distance dependencies by relocating substrings of a generated sentence, feature grammars make use of their recursive nature to keep track of entire structural representations. Whenever a phrase appears in a 'dislocated' position, its entire feature structure is passed through the structures of its context through feature sharing, until it meets its deep-structural position. Like with subcategorization, it is furthermore often assumed that any number of phrases can be simultaneously extraposed from a given constituent.[3] So every constituent bears a SLASH feature, which is typically a *list* of complete feature structures associated with extraposed subconstituents. Such a list makes the feature domain of a grammar strictly unbounded, and this has its consequences for computational tractability.

For simplicity however, let's assume that each constituent has a SLASH feature that stores the feature structure corresponding to one single extraposed constituent. Consider the topicalized sentence

(7.4) Whom$_i$ did John see ε_i?

The accusative form *whom* is used to stress that there are morphological properties which must be carried over the long-distance link between the topic and the verb *see* that selects for it. Since these may in principle be any property, a feature grammar stores *all* information about the phrase *whom* in the SLASH feature.

The rules responsible for head-complement selection can be reduced to a low number, since the lexical entries contain so much information. Such rules will ensure that only one of the daughter phrases has a nonempty SLASH feature, and if one daughter phrase has such a feature, will structure-share it with the SLASH feature of the parent. An example is the rule that generates the standard left-right head-complement structure of English:

$$(7.5)\qquad \mathrm{XP}\begin{bmatrix}\text{head:} & \boxed{1}\\ \text{subcat:} & \boxed{2}\\ \text{slash:} & \boxed{3}\end{bmatrix} \rightarrow \mathrm{X}\begin{bmatrix}\text{head:} & \boxed{1}\\ \text{subcat:} & \begin{bmatrix}\text{first:} & \boxed{4}\\ \text{rest:} & \boxed{2}\end{bmatrix}\\ \text{slash:} & \texttt{nil}\end{bmatrix} \quad \mathrm{NP}\,\boxed{4}\begin{bmatrix}\text{slash:} \ \boxed{3}\end{bmatrix}$$

And an alternative version of the same rule 'consumes' the SLASH feature and produces an empty complement, *i.e.*, a trace:

$$(7.6)\qquad \mathrm{XP}\begin{bmatrix}\text{head:} & \boxed{1}\\ \text{subcat:} & \boxed{2}\\ \text{slash:} & \boxed{3}\end{bmatrix} \rightarrow \mathrm{X}\begin{bmatrix}\text{head:} & \boxed{1}\\ \text{subcat:} & \begin{bmatrix}\text{first:} & \boxed{3}\\ \text{rest:} & \boxed{2}\end{bmatrix}\\ \text{slash:} & \texttt{nil}\end{bmatrix}$$

Finally, there is a rule which produces an NP filler at S level:

$$(7.7)\qquad \mathrm{S}\begin{bmatrix}\text{slash:}\ \texttt{nil}\end{bmatrix} \rightarrow \mathrm{NP}\,\boxed{1}[\,]\quad \mathrm{S}\begin{bmatrix}\text{slash:}\ \boxed{1}\end{bmatrix}$$

[3] It is dubious whether this is really the case—see section **9.4**.

7-10 example: lexical sharing. In feature grammars, *structure sharing*, or identification of subterms, in lexical items can associate values of other grammatical entities. An example is the following lexical entry of the verb *to be*:

$$\textit{is} \rightarrow \left[\begin{array}{ll}
\text{cat:} & \mathtt{verb} \\
\text{head:} & \left[\begin{array}{ll}
\text{form:} & \mathtt{finite} \\
\text{tense:} & \mathtt{present} \\
\text{agr:} & \boxed{1} \left[\begin{array}{ll} \text{person:} & 3 \\ \text{number:} & \mathtt{sg} \end{array}\right] \\
\text{subject:} & \left[\begin{array}{ll}
\text{cat:} & \mathtt{np} \\
\text{head:} & \left[\begin{array}{ll} \text{gender:} & \boxed{2} \\ \text{agr:} & \boxed{1} \\ \text{case:} & \mathtt{nom} \end{array}\right]
\end{array}\right]
\end{array}\right] \\
\text{subcat:} & \left[\begin{array}{ll}
\text{first:} & \left[\begin{array}{ll}
\text{cat:} & \mathtt{np} \\
\text{head:} & \left[\begin{array}{ll} \text{gender:} & \boxed{2} \\ \text{agr:} & \boxed{1} \\ \text{case:} & \mathtt{acc} \end{array}\right]
\end{array}\right] \\
\text{rest:} & \mathtt{nil}
\end{array}\right]
\end{array}\right] \tag{7.8}$$

In French, a verb like *is* requires the subject and the complement to show agreement on both number and gender. Even though no GENDER feature is explicitly present in the lexical entry for *is*, the co-indexing of the AGR feature on the subject and first complement will enforce this agreement.

*

Practical introductions to feature grammars are [Shi86] and [Sel85]; an example of a full-fledged theory is [PS94], and a formalization is found in [Rou97].

Discussion

Since grammars over finite lattices can be extended straightforwardly to grammars over more complex, but still bounded, term structures, the essential difference between such finite grammars and feature formalisms is the interaction of structure sharing with the ability to attach to each node in a derivation an unboundedly deep, recursive structure. One of the conjectures of the **Prologue** is that when the structural backbone has a sufficient capacity, such an unbounded attribute domain is not necessary. This claim, and the conjecture that simple **LMG** is sufficient in this sense, are worked out in part **III** of this thesis.

Feature grammars describe, in their general form, any recursively enumerable language, even if all but one nonterminal of the underlying grammar generate the empty string. A special class of feature grammars, called OFF-LINE PARSABLE, is defined by PEREIRA and WARREN in [PW83]: these are feature grammars on a context-free backbone whose underlying **CFG** analyses are free of *cycles*, or in other words, is not infinitely ambiguous. This eliminates arbitrary calculations which produce no overt syntax, and recognition for these types of grammars indeed turns out to be decidable. However, they still require exponential time for parsing and recognition. Using a more complex backbone, such as an **MCFG** or simple **LMG**, yields a new, larger class of off-line parsable grammars and can hence extend the scope of decidable feature grammars. The step to eliminating unboundedly deep feature structures completely is then only an inch away.

It should be noted that **AGFL** as they occur in the literature are, as they are based on a **CFG** with only a finite domain of space for additional administration, intrinsically incapable of describing heavily non-projective phenomena such as cross-serial dependencies in Dutch. In some cases, this is arguably not a problem—see the discussion in section **9.4**. However, unless the structural backbone underlying a finite-attribute grammar is increased to the strength of at least linear **MCFG**, the scope of such grammars is highly limited—contrary to what some papers, on **AGFL**, such as [Kos91], are claiming. Nonetheless the work done in evaluating **AGFL** attributes seems to be very valuable in the context of the highly localized formalisms studied in this thesis.

7.2 A prototype of an $\mathcal{S}$-LMG parser

In the remaining sections of this chapter, I will discuss concrete approaches to parsing, translation and evaluation of attributes. This first section presents a prototype implementation of an **LMG** parser attached to the term rewriting system EPIC [WK96]. The next sections discuss possible improvements that may make such a system applicable to more realistic, large-scale applications.

Short description

In 1996–97, I implemented the predicate descent algorithm for **LMG** as discussed in chapters **4** and **5** in a prototype parser package written in **C** and **Yacc**/**Lex**, that has the following features

- ▷ The parser package takes a grammar file and a sentence as its input, *i.e.* it is not a parser generator but rather implements a *universal parser*.

  The grammar file is in **LMG** predicate notation or **CFG**-style productions, and is allowed to contain attribute and translation specifications in the format of equations in the equational programming language EPIC.

- ▷ The grammar is a simple **LMG** with the extension that multiple occurrence of a variable on the LHS is allowed (*i.e.*, a **PMCFG** with reduplication can be specified in its canonical form).

- ▷ There is no separate lexical phase—the terminal symbols of the grammar are the 256 **ASCII** codes. White space occurrence can be controlled on a rule-by-rule basis.

- ▷ Output is by default a term in EPIC syntax representing all analyses of the sentence; this is an elementary term format where functions begin in a lower case letter and variables (which do not occur in concrete terms) in capital letters, as in the **Prolog** conventions.

- ▷ Options allow

  - Reducing a production-style grammar (covering standard **CFG** notation) to **LMG** predicate notation.
  - Output in tree form rather than EPIC term.
  - Compiling the grammar into a signature for the generated EPIC terms plus equations (these are copied literally).
  - Unparsing or pretty-printing: input is a term representing a single tree and output is the full sentence according to the grammar.

Below, I will briefly discuss some of these features along the lines of a simple example: a toy translator for simple Dutch sentences into Spanish.

Syntax of LMG grammars and rewriting-based translation

Manipulation of terms, in the form of TERM REWRITING, is a technique that is often used in *software engineering* circles to model the semantics of programming language constructions. Whereas *translation* is a far away target in Linguistics, it is every day practice in Computer Science, and the formal study of such translation mechanisms or *compilers* often concentrates on source and target languages as sets of *terms* and the compiler as a program that transforms terms of one language into terms of another language using straightforwardly definable *rewriting* operations on these terms.

In intermediate stages, generic terms are useful to define auxiliary values that are not part of either the source or the target language—these correspond to the *attributes* one can attach to the structures produced by a context-free grammar or an **LMG**. Such use of term syntax is possible in the **LMG** parser prototype but not discussed here.

The grammar files fed to the **LMG** prototype consist of four SECTIONS: `start symbol`, `types`, `syntax` and `equations`—similar divisions can be found in equational specification systems such as **ASF+SDF** [Kli93] and EPIC [WK96].

start symbol defines the start nonterminal of the grammar

types defines functions in the term signature directly; most such functions are derived automatically from the grammar rules, but some extra-syntactic functions (such as `pack` and `none` which are built-in, but must to be included when referred to in the equations), translation predicates (`trans` in the example discussed here) or attribute functions (no example given) may need to be defined. The function declarations take the form

function-id: -> type-id

for constants or

function-id: $\{$ **type-id** # $\}^*$ **type-id -> type-id**

for proper functions. The type-identifiers are ignored but can be specified to indicate the *intended* use of the functions.

syntax this section consists of a sequence of grammar rules, each bearing a label (if a label is omitted, one is generated from the predicate on the LHS). This label is the function symbol used when the parser generates a forest in term representation (see pages 130*ff*). The underlying term signature is untyped, that is, only the *arity* of the function symbols plays a rôle. This is a feature of the EPIC system, and comes in handy when defining the `pack` operator used to generate ambiguous nodes.

The grammar rules can be either in the familiar **LMG** notation, *e.g.*,

```
sn_sub:  S("dat" s o v) :- NP(s), VP(o, v).
```

or equivalently, in a production format that eliminates the one component of the **LMG** predicates by using straightforward concatenation as in a **CFG** (an example of such a production-style grammar is given in the **Prologue**, grammar (2) on page 8):

```
sn_sub:  S               -> NP o VP(o).
```

The prototype translates rules of the second type to the first. Both these rules generate the following type declaration:

```
sn_sub:  NP # VP         -> S;
```

Optionally, each grammar rule may be immediately followed by one or more arbitrary *equations*.

equations this section contains a sequence of REWRITE RULES over the signature defined in the **types** and **syntax** sections. These are ignored by the parser, but an option of the prototype is to output an EPIC specification containing the signature and these equations, which can then be executed on the output of the parsing stage. In the example discussed below, this feature is used to define a simple translator. Another possible use is to define simple attributes such as singular/plural and a function `disambiguate` which removes incorrect trees in the forest output by the parser. The equations have the form

term = term

where a term is built up from the function symbols and *variables* which start with a capital letter—a good convention is to use the nonterminal symbols for variables if they correspond a phrase of that type. The rewrite rules are interpreted by EPIC as licensing a term matching the pattern on the left hand side of the equation to be rewritten to the term on the right hand side, appropriately substituting terms for the variables used in the LHS.[4]

In the space available in this chapter, I cannot go much deeper into terms and term rewriting, and do justice to the more general algebraic or equational view underlying term rewriting as a software engineering tool. A general textbook on algebraic specification is [BHK89]. At the level of terms and term rewriting, the explanation here overlooks the possibility of function *typing* and the *higher order* or *multi-level* nature of operators such as **pack** and **share**—these features are discussed at length in VISSER's Ph.D. thesis [Vis97], written in the context of the same project as this book.

Figures *7.5* and *7.6* show an example of a grammar specification that can be read by the **LMG** prototype parser. It defines simple syntactic constructions in Dutch

[4] EPIC does this is done in a rightmost-innermost, most specific equation first, strategy.

(nonterminals ending in n) and Spanish (nonterminals ending in e). Most of the rules in the grammar are followed by one or more equations. They construct a function trans that defines a strictly compositional translation of the terms corresponding to Dutch phrases to terms corresponding to Spanish phrases.

For example, on parsing the sentence

(7.9) Jan heeft een auto

the parser outputs the term

```
   sn_decl(jan, vpn_tr(heeft, npn(een, auto))).
```

if the function trans is applied to this sentence and fed to EPIC, the following rewrite sequence takes place:

```
       trans(sn_decl(jan, vpn_tr(heeft, npn(een, auto))))

    => se_decl(trans(jan), trans(vpn_tr(heeft,
                                        npn(een, auto))))

    => se_decl(trans(jan), vpe_tr(trans(heeft),
                                  trans(npn(een, auto))))

    => se_decl(trans(jan), vpe_tr(trans(heeft),
                           npe(trans(een), trans(auto))))

    => ...

    => se_decl(trans(jan), vpe_tr(tiene, npe(un, coche)))

    => se_decl(jan, vpe_tr(tiene, npe(un, coche)))
```

The last rewriting step follows the rule trans(X) = X in the **equations** section which is triggered because no translation is defined for jan.

There is one exception to this *don't translate* default rule—the word *geen* is not given a translation rule, but translation is defined in context; the second rule in the **rules** section in *figure 7.6*, covering this case, is called a NON-COMPOSITIONAL translation rule. It matches not on a single clause, but a combination of syntactic constructions. A transitive verb applied to a noun phrase whose determiner is *geen* results in a negated verb phrase. Hence the translation of sentence (7.10) is

(7.10) Jan heeft geen auto

(7.11) Jan no tiene un coche

Such a translation rule which depends on a construction stretching over more than a single parent–daughters group corresponds to the general case of *pattern matching*, and is called a *polynomial* by Huysen [Hui98].

The first equation in the **rules** section deals with the case that the input sentence is ambiguous, in a rather naive fashion, by lifting the `pack` operator over `trans`. The interaction of this rule with the noncompositional rule following it is problematic. If the second argument is headed by `pack`, the rule is not triggered, and the sentence is left untranslated. In this very simple case, this last equation for `trans` could be modified to also distribute the `pack` function over `trans`, but in a larger specification, this needs to happen for each argument and each rule, which results in a system that is (**i**) unmanageable for maintenance by hand and (**ii**) if generated automatically would result in a number of rules exponential in the number of nodes in the translated constructions. A solution to such problems is discussed in section **7.4**.

The following steps are needed to use the **lmg** and EPIC packages to translate with this specification:

Call the **lmg** package to compile the specification `trans.lmg` into an EPIC source `trans.ep` containing the signature corresponding to the grammar productions, and rewriting rules copied from the specification.

Compile the resulting EPIC file to an executable file `trans`

Call **lmg** on the Dutch sentence

Feed the resulting term to the EPIC executable `trans`

Call **lmg** in *unparse* mode to convert the resulting term back to the corresponding Spanish sentence

This procedure is automated using **UNIX** pipes and **make** files.

```
start symbol Sn

types

   pack:  _ # _   -> _;
   none:          -> _;

   trans: _       -> _;

syntax

sn_sub:  Sn("dat" s o v)  :- NPn(s), VPn(o, v).
sn_decl: Sn(s v o)        :- NPn(s), VPn(o, v).
sn_ques: Sn(v s o)        :- NPn(s), VPn(o, v).

se_sub:  Se("que" s v)    :- NPe(s), VPe(v).
se_decl: Se(s v)          :- NPe(s), VPe(v);

   trans(sn_sub(NP, VP))  = se_sub(trans(NP), trans(VP));
   trans(sn_decl(NP, VP)) = se_decl(trans(NP), trans(VP));
   trans(sn_ques(NP, VP)) = se_decl(trans(NP), trans(VP)).


vpn_tr:  VPn(o, v)        :- VTn(v), NPn(o).
vpe_tr:  VPe(v o)         :- VTe(v), NPe(o);

   trans(vpn_tr(VT, NP))  =  vpe_tr(trans(VT), trans(NP)).


%% negated VP in Spanish (target language) only
%%
vpe_neg: VPe("no" v)      :- VPe(v).

npn:     NPn(d n)         :- DPn(d), CNn(n).
npe:     NPe(d n)         :- DPe(d), CNe(n);

   trans(npn(DP, CN))     =  npe(trans(DP), trans(CN)).
```

Figure 7.5: Toy Dutch to Spanish translator specification (part 1)

```
   DPn("de").
   DPn("een").
   DPe("el").
   DPe("un");

      trans(de)      =  el;
      trans(een)     =  un.

   %% No immediate translation for ‘geen’ and names
   %%
   DPn("geen").

   NPn("Jan").
   NPn("Marie").
   NPn("Fred").

   CNn("auto").
   CNe("coche");

      trans(auto)    =  coche.

   VTn("heeft").
   VTe("tiene");

      trans(heeft)   =  tiene.

rules

   %% Lift ‘pack’ over ‘trans’
   %%
   trans(pack(X, Y)) = pack(trans(X), trans(Y));

   %% Non-compositional pattern rule for negated NPs
   %%
   trans(vpn_tr(VT, npn(geen, CN)))
      =  vpe_neg(vpe_tr(trans(VT), npe(un, trans(CN))));

   %% Default rule is: don’t translate
   %%
   trans(X) =  X;
```

Figure 7.6: Toy Dutch to Spanish translator specification (part 2)

7.3 Off-line parsability and multi-stage analysis

It was already mentioned in chapter **5** (page 114) that in practice, storing the items in arrays indexed by the integer sentence indices would too quickly consume amounts of memory that are beyond the capacity of any reasonably sized modern computer system. Therefore, a realistic parser must

- ▹ store only those items that it encounters in the search for a derivation
- ▹ include machinery that further narrows down such ‘predicted items’

In this section, I will discuss the limited form of prediction used in the current **LMG** prototype, and an improvement in terms of *two-phase* parsing with a context-free backbone derived from the **LMG**.

Details of the current implementation

A simple calculation points out that simply encoding the memo tables into arrays of boolean values is a hopeless enterprise—assume an input of length 1000, and one would already need 1 megabits for each nonterminal of a context-free grammar; this gets dramatically worse for tuple grammars. Therefore the prototype stores only those items it encounters in a derivation. Storing only a limited set of items implies that looking up an item in the memo storage is no longer an atomic action. However, as the **LMG** prototype does, storing them in binary tree structures reduces the look-up time to $\mathcal{O}(p \log n)$ where p is the number of indices—twice the number of arguments in a simple **LMG**.

I tested a straightforward implementation of a predicate descent algorithm, storing the memo tables as binary trees, on sample grammars of Dutch and the toy programming language **pico**, which is a highly simplified version of Pascal. The performance on Dutch was within reasonable bounds. Although the times for parsing increasingly large **pico** programs confirmed the expected $\mathcal{O}(n^3 \log n)$ complexity, the constants involved turned out to be still so large that a 25 line program was too large to be parsed within 32 megabytes of memo storage. I then made the following improvements:

- ▹ Not only are the complete items corresponding to nonterminals plus arguments stored, also each partial assignment to the variables occurring in a rule, where the variables are traversed in a fixed order, is stored, up to the point where such a partial assignment is known not to produce any complete variable instantiations for which a succeeding parse exists. This can be thought of as a form of on-the-fly *bilinearization* of the grammar.

  This optimization requires extra storage, but decreases the number of items for which a proof is attempted—on the average, the space consumption turns out not to increase. The time complexity is improved by an order of magnitude: for example, only after this optimization an $\mathcal{O}(n^3)$ complexity is obtained for all context-free grammars.

▷ Every component of each nonterminal, and every variable in each of the **LMG** rules is associated with three tables ranging over terminal symbols: *left-corner*, *right-corner*, *occur* and a flag *lambda*. These are instantiated when scanning the grammar (*i.e.* independent of the input) and contain a boolean value indicating whether given terminal symbols can occur at the left, right, at arbitrary places in a component of a nonterminal, or whether this string can be empty.

These tables are used in the parser to restrict the number of items evaluated and memoed to a limited subset: the idea is that a parse of a VP is not attempted if it starts with the word *Jan*, because it can be derived from the grammar that this situation does not occur.

A problem of this approach is that looking at the leftmost and rightmost terminal symbol generated in a component of an item is not always effective. Moreover, the terminal symbols in the **LMG** prototype are the 256 ASCII values—the prototype is *scannerless*. With a sufficiently large dictionary, think of

$$\mathrm{NP} \rightarrow \texttt{almond} \mid \texttt{bee} \mid \texttt{cheddar} \mid \texttt{dog} \mid \texttt{elk} \mid \ldots$$

this means that terminal corner prediction will not cut down the checked item sets at all, unless longer sequences are examined, but this quickly results in tables that are too big to store.[5]

The resulting system parsed a **pico** program of 84 lines in 6 seconds but took approximately 15MB of memory. With each doubling of the input program size, the memory consumption grew five-fold and time consumption 10-fold. Although this does correspond to the theoretical $\mathcal{O}(n^3)$ complexity, in practice much better algorithms exist.

The performance on a basic Dutch crossed dependency grammar however, a simple **LMG** with 4-ary nonterminals, exceeded all expectations and was processed extremely efficiently.

Multi-stage parsing

Instead of the static terminal corner predications described above, whose capacities for statically ruling out items is too limited, this section and the next will look at improvements of the performance of parsing (below) and attribute evaluation (section **7.4**) using multiple stages, where the result of each stage is used in the next stage to limit the number of investigated possibilities. This does not influence the theoretical, or worst-case, complexity of these problems, but it has in practice often proven to be an effective method, and is altogether not psychologically implausible—it is a form of breaking a hardly manageable task into a number of chunks that are each manageable in practice.

There are various solutions to **CFG** parsing that have tackled the space and time consumption problems that occur in the **LMG** prototype. One of the most successful of

[5] It is tempting to think that this is a problem with scannerless parsing, but the problem is bound to occur in some other form once grammars get sufficiently sophisticated.

these is **LALR** parsing, which is popular in Computer Science, but is limited to a form of context-free grammars that is deterministically parsable: one can proceed through the sentence from left to right, finding precisely one parse if and only if the input string is accepted by the grammar. A solution to this limitation is **GLR** parsing [Tom86] [Tom91] [Rek92], which is still highly space and time efficient but can handle all context-free grammars, and outputs a packed and shared forest. **GLR** parsers perform essentially better than a predicate descent parser as outlined in the previous section.

I will now discuss how the time and space used in predicate descent parsing of simple **LMG** can be essentially improved by adding a pre-processing stage of parsing with a context-free grammar derived from the **LMG**. The idea is to look at each of the components of the tuples derived by a nonterminal independently, so as to exclude very improbable analyses—the occur checks in the **LMG** prototype already illustrated such an independent analysis in terms of terminal corner and occur tables, but the one on a derived **CFG** is more sophisticated, and will cut down the number of items to be stored more substantially.

This is a simple construction, and two examples are sufficient to show how it will work. *Figure 7.7* on page 151 shows the context-free backbone for a grammar analogous to the grammars of chapter **3**.

In general, simple **LMG** or linear **MCFG** are not *off-line parsable* when split up into a context-free phase and a filter; that is, the analyses w.r.t. a context-free backbone contain cycles. This is already manifested in very simple practical grammars. While the grammar in *figure 7.7* is free of cycles, the more standard NP/VP grammar in *figure 7.8* has a **CF** backbone that does generate cycles ($\mathrm{VP}_o \Rightarrow \mathrm{VP}_d\mathrm{VP}_o \Rightarrow \mathrm{VP}_o$). At first sight it is a pure coincidence that the binary branching grammars in this thesis, with verb clauses rather than traditional VPs, are free from cycles in the context-free backbone—they were certainly not designed to have this property. On the other hand, they simply capture more linguistic knowledge and as such happen to provide the simple context-free pre-processing stage with more information—in the NP/VP grammar, a VP has a 'direct object' which is empty in the case of an intransitive VP. This is an inelegant feature that the grammar writer is punished for by cyclicity in the context-free backbone. When the grammar is extended with rules for modal verbs and (partial) extraposition verbs, further ambiguity is introduced—be it finite and acyclic. Whichever way the facts are turned, cyclicity in the context-free backbone is likely to appear in more sophisticated grammars that allow more arbitrary components to be empty.[6]

Such non-offline parsability is often considered a problem in implementations of feature grammars—indeed, the corresponding SLASH-feature based models of Dutch crossed dependencies are also not off-line parsable. However, if a parsing stage is strictly split into two phases, whose interface representation of forests has a construction to represent cycles, this need not at all present a real problem. Such a division is often *not* present in the **Prolog** based **DCG** implementations in which off-line parsabil-

[6] An example is an analysis in which lexical entries for verbs consist of two components, one of which contains prefixes that can be separated from the verb, as appears frequently in German and Dutch—see the footnote on page 193.

LMG:

[1] $\mathrm{C}^{\mathrm{sub}}(\texttt{dat}\ s\ o\ h\ v)$ `:-` $\mathrm{V}^0(s, o, h, v)$.
[2] $\mathrm{C}^{\mathrm{decl\text{-}wh}}(s\ h\ o\ v)$ `:-` $\mathrm{V}^0(s, o, h, v)$.
[3] $\mathrm{C}^{\mathrm{ques}}(h\ s\ o\ v)$ `:-` $\mathrm{V}^0(s, o, h, v)$.

[4] $\mathrm{V}^0(s,\ o,\ h,\ v)$ `:-` $\mathrm{N}^0(s), \mathrm{V}^{\mathrm{I}}(o, h, v)$.
[5] $\mathrm{V}^{\mathrm{I}}(o,\ v,\ \varepsilon)$ `:-` $\mathrm{V}^{\mathrm{T}}(v), \mathrm{N}^0(o)$.
[6] $\mathrm{V}^{\mathrm{I}}(s\ o,\ r,\ h\ v)$ `:-` $\mathrm{V}^{\mathrm{R}}(r), \mathrm{V}^0(s, o, h, v)$.

[8] $\mathrm{V}^{\mathrm{I}}(\varepsilon,\ \texttt{zwemmen},\ \varepsilon)$.

Backbone **CFG:**

[1] $\mathrm{C}^{\mathrm{sub}} \rightarrow \texttt{dat}\ \mathrm{V}^0_s\ \mathrm{V}^0_o\ \mathrm{V}^0_h\ \mathrm{V}^0_v$
[2] $\mathrm{C}^{\mathrm{decl}} \rightarrow \mathrm{V}^0_s\ \mathrm{V}^0_h\ \mathrm{V}^0_o\ \mathrm{V}^0_v$
[3] $\mathrm{C}^{\mathrm{ques}} \rightarrow \mathrm{V}^0_h\ \mathrm{V}^0_s\ \mathrm{V}^0_o\ \mathrm{V}^0_v$

[4] $\mathrm{V}^0_s \rightarrow \mathrm{N}^0$
[4′] $\mathrm{V}^0_o \rightarrow \mathrm{V}^{\mathrm{I}}_o$
[4″] $\mathrm{V}^0_h \rightarrow \mathrm{V}^{\mathrm{I}}_h$
[4‴] $\mathrm{V}^0_v \rightarrow \mathrm{V}^{\mathrm{I}}_v$

[5] $\mathrm{V}^{\mathrm{I}}_o \rightarrow \mathrm{N}^0$
[5′] $\mathrm{V}^{\mathrm{I}}_h \rightarrow \mathrm{V}^{\mathrm{T}}$
[5″] $\mathrm{V}^{\mathrm{I}}_v \rightarrow \varepsilon$

[6] $\mathrm{V}^{\mathrm{I}}_o \rightarrow \mathrm{V}^0_s\ \mathrm{V}^0_o$
[6′] $\mathrm{V}^{\mathrm{I}}_h \rightarrow \mathrm{V}^{\mathrm{R}}$
[6″] $\mathrm{V}^{\mathrm{I}}_v \rightarrow \mathrm{V}^0_h\ \mathrm{V}^0_v$

[8] $\mathrm{V}^{\mathrm{I}}_o \rightarrow \varepsilon$
[8′] $\mathrm{V}^{\mathrm{I}}_h \rightarrow \texttt{zwemmen}$
[8″] $\mathrm{V}^{\mathrm{I}}_v \rightarrow \varepsilon$

Figure 7.7: Verb clause analysis and its acyclic context-free backbone.

LMG:

[1]	$\mathrm{C}^{\mathrm{sub}}(\texttt{dat}\ s\ d\ o\ h\ v)$	:-	$\mathrm{NP}(s),\ \mathrm{VP}(d,o,h,v).$
[2]	$\mathrm{C}^{\mathrm{decl\text{-}wh}}(s\ h\ d\ o\ v)$	:-	$\mathrm{NP}(s),\ \mathrm{VP}(d,o,h,v).$
[3]	$\mathrm{C}^{\mathrm{ques}}(h\ s\ d\ o\ v)$	:-	$\mathrm{NP}(s),\ \mathrm{VP}(d,o,h,v).$
[4]	$\mathrm{VP}(\varepsilon,\ \varepsilon,\ h,\ \varepsilon)$	:-	$\mathrm{V}^{\mathrm{I}}(h).$
[5]	$\mathrm{VP}(d,\ \varepsilon,\ h,\ \varepsilon)$	:-	$\mathrm{V}^{\mathrm{T}}(h), \mathrm{NP}(d).$
[6]	$\mathrm{VP}(n,\ d\ o,\ r,\ h\ v)$	:-	$\mathrm{V}^{\mathrm{R}}(r), \mathrm{NP}(n),\ \mathrm{VP}(d,o,h,v).$

Backbone **CFG:**

[1]	$\mathrm{C}^{\mathrm{sub}}$	→	$\texttt{dat}\ \mathrm{NP}\ \mathrm{VP}_d\ \mathrm{VP}_o\ \mathrm{VP}_h\ \mathrm{VP}_v$
[2]	$\mathrm{C}^{\mathrm{decl}}$	→	$\mathrm{NP}\ \mathrm{VP}_h\ \mathrm{VP}_d\ \mathrm{VP}_o\ \mathrm{VP}_v$
[3]	$\mathrm{C}^{\mathrm{ques}}$	→	$\mathrm{VP}_h\ \mathrm{NP}\ \mathrm{VP}_d\ \mathrm{VP}_o\ \mathrm{VP}_v$
[4]	VP_d	→	ε
[4′]	VP_o	→	ε
[4″]	VP_h	→	V^{I}
[4‴]	VP_v	→	ε
[5]	VP_d	→	NP
[5′]	VP_o	→	ε
[5″]	VP_h	→	V^{T}
[5‴]	VP_v	→	ε
[6]	VP_d	→	NP
[6′]	VP_o	→	$\mathrm{VP}_d\ \mathrm{VP}_o$
[6″]	VP_h	→	V^{R}
[6‴]	VP_v	→	$\mathrm{VP}_h\ \mathrm{VP}_v$

Figure 7.8: Traditional NP-VP analysis and its cyclic context-free backbone.

ity is a strict requirement. This is partly due to the fact that in a **Prolog** setting, cyclicity (and even left-recursion $A \overset{*}{\Rightarrow} Aw$) often presents difficulties. The memoing algorithms proposed here, and also **GLR** parsing for context-free grammars, do not suffer from similar problems—this is partly because their formulation arose in the context of implementations in imperative programming languages whose pointer paradigm enables straightforward circumvention of problems with cyclicity without penalties in terms of algorithmical complexity.

Using the CF backbone for prediction

The problems, mentioned earlier, that arise when the ambiguous output of a parser is used for further processing, play an equally awkward rôle when a direct attempt is made at a form of *merging* the derivations for the different components of the same nonterminal in a context-free parse forest resulting of the backbone parsing phase into derivations according to the original **LMG**. The presence of cycles in the parse forests makes such an algorithm even harder to construct. Obviously, the decision to multiply out a parse forest into a list of parses, which is often taken at this point, is not sensible at such an early stage.

However, in this particular case, there is a much simpler solution, which merely uses the result of a context-free parsing stage for *prediction*, much like the corner and occur tables are used in the prototype **LMG** parser of the previous section. The only information that is used by the second stage in this case is, for each **CF** predicate, whether it leads to a context-free parse or not. Hence the **CF** parser is only required to produce a table of items that have a successful parse, and can *disregard cycles*. The idea that if there is a cyclic parse, then there is also a non-cyclic parse, was already used in the predicate descent algorithm (section **4.2**). This context-free item information is then used to predict useful items for the real **LMG** derivation. So the cycles need not even be represented in the interface between the **CF** component and the **LMG** parser.

So far, I have glossed over the formal details of deriving a context-free backbone from an **LMG**. I will refrain from working this out in detail, because it is a simple construction. However, there is a number of details that needs attention. If the grammar is an **MCFG**, it can be translated into a **CFG** by taking one nonterminal for each component of each nonterminal.

Since the output of the context-free parser is used for prediction in the **LMG** stage, it should contain at least all items that occur in a derivation of the sentence, but certainly not less; in other words, the information must be accurate, but is allowed to be *optimistic*. In the case of a **PMCFG** with a reduplicating rule, or an **LMG**, some further investigation is necessary.

Case 1 A reduplicating **PMCFG** production, type $A(xx)$ **:-** $B(x)$. In this case, the rule is replaced with $A_1 \rightarrow B_1\ B_1$. The latter rule produces all concatenations of two strings recognized by B_1, whereas the original **PMCFG** rule recognizes only concatenations of two *identical* copies of strings recognized by B. Since this is over-general, this is not a problematic case.

Case 2 A sharing **LMG** clause, type $A(x) \;\textbf{:-}\; B(x), C(x)$. An optimistic solution to this case is simply to drop the second use of the variable x on the RHS, producing the context-free rule $A_1 \to B_1$. While this seems an acceptable solution when looking at this single rule, this rule may be the only one in a given derivation that requires the item $C(w)$ to be proved for a certain substring w of the input, in which case the context-free parser will output an item set that is incomplete, and the second phase may incorrectly conclude that no parse is found. Therefore, such a clause must be translated to *two* context-free rules $A_1 \to B_1$ and $A_1 \to C_1$. In the general case, the **LMG** rules must be multiplied out into 2^s rules where s is the number of times a variable is shared between two predicates on the RHS of the clause.

A preliminary conclusion is that two-stage **LMG** parsing is essentially more involved than two-stage **PMCFG** parsing, and when the former is implemented, sharing is relatively expensive.

I have not tested an actual implementation. Therefore it is uncertain whether the approach sketched actually performs well. It should be noted that the performance of efficient context-free parsing techniques such as **GLR** is known to decrease when cyclicity is introduced; I am not sure what the implications are of the idea that these cycles need not be actually constructed, which is the case in the two-phase analysis proposed here that outputs a set of recognized items only, instead of a forest.

Finally, it should be noted that the worst-case complexity is not improved by the proposed optimizations. In the case of the **PMCFG** for a^{2^n} on page 70 for example, the result of the context-free pre-processing phase is useless. But in practical cases, the benefit of each excluded predicate implies a constant speed-up that is polynomial in the size of the input. In practical examples such as a slightly extended version of the simple grammar for Dutch, the number of checked items is even a *factor* $\mathcal{O}(n^2)$ smaller than in the case without context-free prediction.

7.4 Attribute evaluation

In section **7.2**, a few problems were already mentioned relating to automated translation over forest representations. The rewriting operations or translation functions must be allowed to distribute over packing nodes, but this results in a high number of matching rules, and possibly into unnecessary un-packing of forests if the translations obtained turn out afterwards to be identical. The latter problem is sometimes solved by re-packing, but none of the currently known solutions of this form avoid exponential complexity in terms of the size of the packed, shared forest.

For some stages of processing performed on the output of a phase of syntactic analysis, such as translation of compound expressions as illustrated in section **7.2**, un-packing forests seems to be a necessary step. In any case, it is obviously important to cut down the size of the forest as far as possible before doing any such form of structure-to-structure translation. This is where the finite attribute formalisms play an important rôle.

For those stages of analysis that require only finite attributes, relations between which are defined by equations local to the productions of the underlying grammar, such as **TGFL**, better algorithms are known—however, their tractability depends strongly on one's point of view.

This problem has been recognized in the literature, and there is a number of proposals for assigning finite attributes *after* a context-free parsing stage. DEKKERS, NEDERHOF and SARBO [NS93] discuss a range of such solutions, and propose one superior solution that takes a form of two-stage processing similar to the one for **LMG** discussed in the previous section.

They propose to evaluate the attributes in two phases, the first of which is very efficient, but *optimistic*, and substantially reduces the size of the forest. The second phase proposed is again optimistic, but it interleaved with user interaction with the final goal to yield a single remaining parse tree. The correctness of the algorithm depends on this user interaction and selection of one single tree.

I will now discuss this proposal in a little more detail, give an example, point out some problems, and propose solutions—but refer to [NS93] for the algorithms themselves. Nederhof and Sarbo's method proceeds as follows:

1. Parse the input sentence with the underlying context-free grammar.

2. A REGULAR NODE is a node that is not a packing or sharing node. Divide the resulting parse forest into CELLS which are maximal connected groups of regular nodes.[7]

3. (*attribute evaluation phase A.*) Associate with each regular node and each of the restricting affix equations a table defining a subset of the relevant affix domain. Initialize all these tables to contain all possible values.

[7] In [NS93] the packing and sharing nodes are inside the cells, but the choice to include them or not is a matter of presentation and may play a rôle in a concrete implementation.

Repeatedly perform the following actions, until there are no more changes to the stored affix sets:

(a) For each regular node in each cell, restrict the stored affix subsets by applying the restricting equations once.

(b) For each packing node, restrict the values of the affixes of the parent node to the union of the values of the daughter node.

(c) For each sharing node, restrict the values of the affixes of the daughter node to the union of the values of the parent nodes.

(d) Remove cells containing a node with an empty affix subset—these are called *dead wood* and lead to a derivation that fails one of the constraint. If a packing node has only one daughter, or a sharing node has only one parent, eliminate it; the parent and daughter cells become a single cell. If a packing node has no daughters, or a sharing node has no parents, eliminate the remaining cell connected to it.

4. (*attribute evaluation phase B*.) Associate with each regular node a table storing all possible *tuples* of affix values for the restricting equations, and repeat the same procedure as in *phase A*, but when after termination the resulting forest contains more than one parse, present one of the packing or sharing nodes to the user and ask the user to choose one of the daughters. Remove the other daughters and apply phase B again. Do this until only a single tree is left.

This algorithm has a time complexity linear in the number of nodes in the packed/shared forest representation,[8] exponential in the size of the largest domain of a single attribute (*phase A*) and in the size of the largest total affix domain (*phase B*). Nederhof and Sarbo propose various methods of storing the tuples of affix values in *phase B* so as to achieve a better performance in practical implementations, and their results are promising.

The approach is divided into two phases for practical reasons. *Phase B* is the one with the highest complexity in terms of the sizes of the attribute domains, and in fact *phase A* could be left out, resulting in a procedure with roughly the same behaviour, and the same formal complexity properties.

In practice however, many parses can be ruled out by looking at the affix values individually. The **AGFL** of *figure 7.4* provides a good example: the context-free analysis of the (incorrect) sentence *the man walk* yields the parse forest at the top of *figure 7.9*. The figure shows how this forest is reduced to a single tree already in *phase A*, so *phase B* need not even be applied.

In some cases however, the first phase does not spot impossible combinations, such as with the grammar in *figure 7.10*. On the noun phrase *der Hunde*, *phase A* will restrict the agreement features to `agr(nom|dat, sg|pl)`, but will not remove all

[8] Context-free parsing techniques which output packed and shared forest with the least possible order of nodes in terms of the length of the input sentence are known—an example is **GLR** parsing.

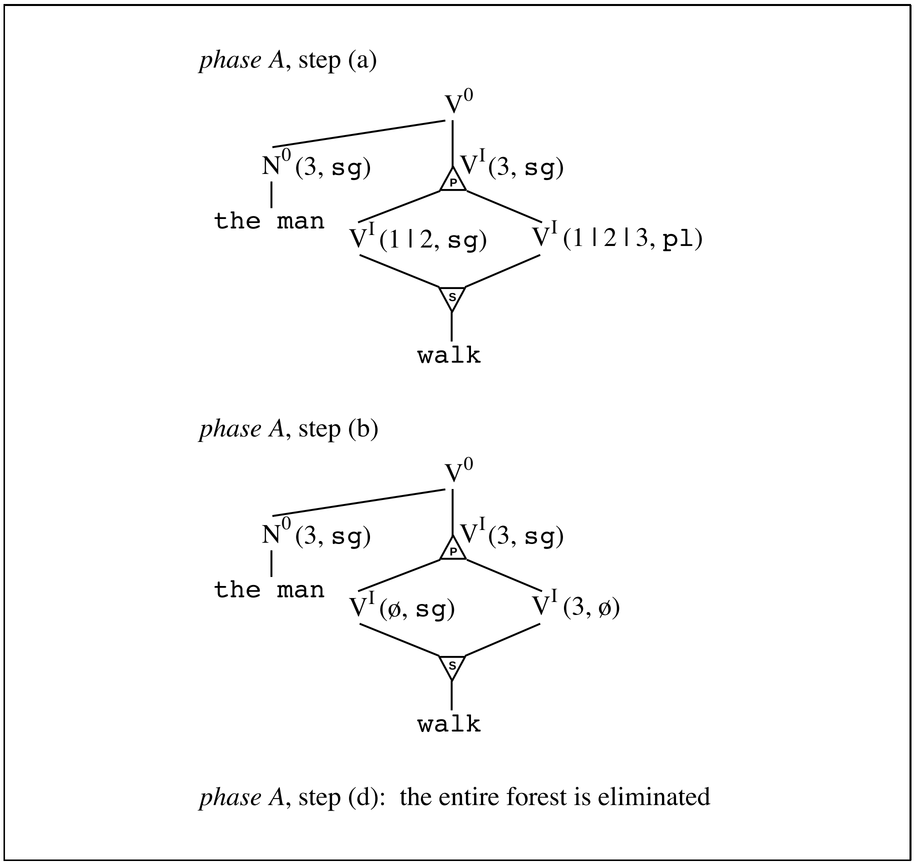
ambiguity. It is only in the second stage that the feature is reduced to $\mathtt{agr}(\mathtt{dat}, \mathtt{pl})$ and only a single tree is left.

In the context of **TGFL** over arbitrary simple **LMG**, there are two problems with Nederhof and Sarbo's approach,

1. When the algorithm is applied to simple **LMG** with sharing, or infinitely ambiguous **CFG** with many ε-productions,[9] there will be cases in which the algorithm will fail to recognize a forest that is empty, and will prompt the user to select one of the alternatives. This happens in the situation in *figure 7.11*. Only after the user has chosen one alternative, the algorithm will fail to produce a tree and discover that the forest is empty.

2. A syntactical analysis based on finite attributes may be followed by stages of

[9] Such cases are explicitly ruled out in [NS93], amounting essentially to a requirement of off-line parsability in the case of context-free grammars; for the **LMG** cases Nederhof and Sarbo's conditions effectively forbid the use of sharing, reducing the applicability of the algorithm to **PMCFG**.


phase A, step (a)

phase A, step (b)

phase A, step (d): the entire forest is eliminated

Figure 7.9: Phase A working on the incorrect sentence *the man walk*.

> further disambiguation, *e.g.* based on anaphoric linking and/or semantic domain restrictions, which arguably cannot be carried out over finite domains. Both the efficiency and the correctness of the algorithm depend on the property that the result is a *single* parse tree: if the algorithm is applied *without* user interaction, and at a stage in *phase B* where the algorithm cannot further cut down the forest, there are two mutually dependent binary packing nodes left, then the algorithm has at that stage a forest containing 4 parses only 2 of which are correct.

The solution to both these problems is to apply algorithms in the spirit of Nederhof and Sarbo's for as many possible stages of analysis, but as a final step, to eliminate packing nodes where necessary, and perform further stages of analysis on the remaining, hopefully very few, analyses.

An objection to such an approach by the formal-minded reader may be that this may be an average-case speed-up, but it does not improve the situation in the general

$$
\begin{array}{rcrcl}
 & & D & = & \{D_{\text{gender}}, D_{\text{number}}, D_{\text{case}}\} \\
 & & D_{\text{gender}} & = & \{\mathtt{m}, \mathtt{f}, \mathtt{n}\} \\
 & & D_{\text{number}} & = & \{\mathtt{sg}, \mathtt{pl}\} \\
 & & D_{\text{case}} & = & \{\mathtt{nom}, \mathtt{gen}, \mathtt{dat}, \mathtt{acc}\} \\
 & & \nu(\mathrm{N}^0) & = & \mathrm{EMPTY} \\
 & & f_{\mathrm{V}^0} & = & e : \mathrm{EMPTY} \\
\nu(\mathrm{V}^{\mathrm{I}}) & = & \nu(\mathrm{N}^0) & = & \mathrm{NAGR} \\
f_{\mathrm{V}^{\mathrm{I}}} & = & f_{\mathrm{N}^0} & = & \mathit{nagr} : (D_{\text{gender}} \times D_{\text{number}} \times D_{\text{case}}) \to \mathrm{NAGR}
\end{array}
$$

$$
[r_1] \quad \mathrm{N}^0 \to \mathrm{D}^0 : \mathit{nagr}(g,n,c),\ \mathrm{N}^{\mathrm{C}} : \mathit{nagr}(g,n,c)
$$

$$
\begin{array}{lllll}
[r_2] & \mathrm{D}^0 : \mathit{nagr}(\mathtt{m}, \mathtt{sg}, \mathtt{nom}) & \to & \mathtt{der} \\
[r_3] & \mathrm{D}^0 : \mathit{nagr}(\mathtt{m}|\mathtt{n}, \mathtt{sg}, \mathtt{gen}) & \to & \mathtt{des} \\
[r_4] & \mathrm{D}^0 : \mathit{nagr}(\mathtt{m}|\mathtt{n}, \mathtt{sg}, \mathtt{dat}) & \to & \mathtt{dem} \\
[r_5] & \mathrm{D}^0 : \mathit{nagr}(\mathtt{m}, \mathtt{sg}, \mathtt{gen}) & \to & \mathtt{den} \\
[r_6] & \mathrm{D}^0 : \mathit{nagr}(\mathtt{f}, \mathtt{sg}, \mathtt{nom}|\mathtt{acc}) & \to & \mathtt{die} \\
[r_7] & \mathrm{D}^0 : \mathit{nagr}(\mathtt{f}, \mathtt{sg}, \mathtt{gen}|\mathtt{dat}) & \to & \mathtt{der} \\
[r_8] & \mathrm{D}^0 : \mathit{nagr}(\mathtt{n}, \mathtt{sg}, \mathtt{nom}|\mathtt{acc}) & \to & \mathtt{das} \\
[r_9] & \mathrm{D}^0 : \mathit{nagr}(\mathtt{m}|\mathtt{f}|\mathtt{n}, \mathtt{pl}, \mathtt{nom}|\mathtt{acc}) & \to & \mathtt{die} \\
[r_{10}] & \mathrm{D}^0 : \mathit{nagr}(\mathtt{m}|\mathtt{f}|\mathtt{n}, \mathtt{pl}, \mathtt{gen}|\mathtt{dat}) & \to & \mathtt{der} \\
 & & & \\
[r_{11}] & \mathrm{N}^{\mathrm{C}} : \mathit{nagr}(\mathtt{m}, \mathtt{sg}, \mathtt{nom}|\mathtt{dat}|\mathtt{acc}) & \to & \mathtt{Hund} \\
[r_{12}] & \mathrm{N}^{\mathrm{C}} : \mathit{nagr}(\mathtt{m}, \mathtt{sg}, \mathtt{gen}) & \to & \mathtt{Hundes} \\
[r_{13}] & \mathrm{N}^{\mathrm{C}} : \mathit{nagr}(\mathtt{m}, \mathtt{sg}, \mathtt{dat}) & \to & \mathtt{Hunde} \\
[r_{14}] & \mathrm{N}^{\mathrm{C}} : \mathit{nagr}(\mathtt{m}, \mathtt{pl}, \mathtt{nom}|\mathtt{gen}|\mathtt{dat}|\mathtt{acc}) & \to & \mathtt{Hunde}
\end{array}
$$

Figure 7.10: **AGFL** for noun phrase agreement in German.

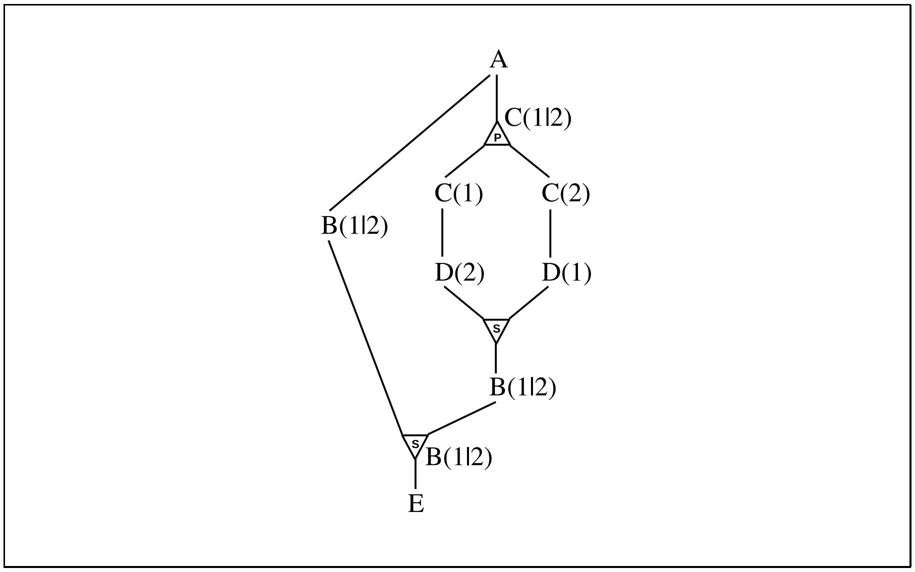


Figure 7.11: Annotated **LMG** forest for which Nederhof/Sarbo's algorithm fails.

case. This is true, but as in the case of context-free preprocessing for **LMG** parsing, every analysis eliminated from the forest in order *n* time reduces the input for the further, theoretically exponential stage, which means *an exponential benefit*, even though theoretically, the remaining problem is exponential.

In some practical cases, different sets of finite attributes have no interaction at all. An example is the illustrated morphological analysis in terms of person and number versus simple lattice-based semantic domain information. In such cases, *phase B* can be executed on these different, independent sets of affixes separately and consecutively, so as to achieve a better time and space complexity in each of the phases. The different phases may also be equipped with algorithms whose parameters are tuned for efficiency on the particular size of the involved attribute domains, which are likely to be essentially different for *e.g.* morphological processing and semantic domain information. This observation may also have implications for some of the calculations made for the computational hardness of finite attribute systems such as **GPSG** in practical cases.

7.5 Using simple LMG for extended representations

The proposed solutions in this chapter have glossed over the possibility of using the power of simple **LMG** directly to model attachment of various attributes. This possibility is interesting from a formal perspective, but in practice, is rather limiting, because the complexity of such approaches is prohibitive. For the sake of completeness however, I will illustrate this option briefly. Another interesting facet of **LMG** is its capacity of capturing constraints over integer numbers bounded by the length of the input sentence.

7-11 example: LMG over lattices. If a simple **LMG** is translated to an **iLMG** as defined in proposition 5-6 on page 109, the indices of this **iLMG** need not be interpreted as indices in a *string*; they can also be taken to refer to the nodes in a *string lattice*, or equivalently, to the states in a finite automaton. Such an approach is often taken in speech analysis, in which the input string consists of an ambiguous sequence of morphemes.

One problem to watch out for in this case is the use of *sharing* on the RHS of an **LMG** production. This will enable an **LMG** analysis over an input lattice to take different threads in the lattice for the shared part of the input. Further research will need to investigate how such possible problems can be circumvented, perhaps by requiring subderivations to show matching patterns at their leaves when sharing is used. Of course, such a solution may have consequences for the complexity of **LMG** parsing.

7-12 example: additional input and attributes. Without damaging the complexity results, an **LMG** may be taken to have a start symbol S that takes more than one argument, one of which is the input string, and where the other arguments are instantiated with constant strings or string lattices. Such a lattice can then model a domain of finite affixes as used in **AGFL**. Sharing can be used to model agreement on affix values.

7-13 example: integer components. A hybrid definition combining features of simple **LMG** and **iLMG** can allow a grammar to manipulate both strings and integer values bounded by the length of the input, while maintaining polynomial complexity of recognition.

This has various applications, most notably in examples such as co-ordination and number series, illustrated in chapter **8**. A problem with co-ordination not discussed yet is that a grammar using sharing will recognize the incorrect Dutch sentence (7.12).[10]

(7.12) $*$. . . dat Anne Frank aankwam en zag.
"That Anne Frank arrived and Anne saw Frank." vs.
* *"That Anne saw and arrived Frank"*

Such a situation can be avoided if verb clause conjuncts have an integer argument that counts the number of object noun phrases.

[10] Manaster-Ramer [MR87] used *Willem-Alexander* for similar examples, but pointed out that the dash between *Willem* and *Alexander* is problematic.

More complex examples, perhaps along the lines of the rules for counting occurrences of categorial operators in co-ordinated (non-)constituents proposed in [CH], may be able to make sophisticated sharing analyses of co-ordination correct that would otherwise be able to conjoin elliptic phrases whose structures are not compatible. In more general terms, the cases of sharing marked ‘dangerous’ elsewhere in this thesis can be fixed with, or replaced with, sharing over integer values.

This capacity to deal with integer values bounded by the size of the input string may also help treating those cases that cannot be dealt with using the finite attribute techniques proposed in this chapter.

Conclusions to chapter 7

This chapter is a plea for separation of the tasks of parsing and processing of attribute constraints into several dedicated phases. These phases need to be well argued for, before any attempts are made to develop an actual natural language system on these grounds. Part **III** of this thesis will provide additional argumentation for the divisions made here. This is not the current trend in linguistics, which is rather more to attempt to put the whole of linguistic structure, attributes, semantic representations, *&c.* into a single descriptive framework. There are, however, published arguments inspired by practically oriented research which argue for separation of tasks into separate dedicated phases of analysis.[11]

This is not to say that such an approach is void of problems—most notably, the interaction between various stages of analysis, which *all* feature highly similar disambiguation problems, is hard to implement. Once one stage of analysis is over, later stages can take profit from the attributes it has calculated, but not vice versa. Another problem of having several distinct, optimized stages of analysis is that they need to be argued for on a principled basis, because once such a stage is built in, it is there for its fixed purpose, and this is against the wisdom of *re-usability*. On the other hand, there clearly are distinct differences in the nature of ambiguity at different locations in the process of syntactic, binding and semantic analysis. The more complex features of semantic disambiguation seem, at the current level of knowledge, altogether impossible to treat without exponential expansion. Therefore, the more analyses can be eliminated using simple tools, the better one can cope in large-scale systems on the basis of contemporary methods of analysis—when it is born in mind that if a *final* stage of analysis is exponential in the size of the structure it operates on, then every previous stage that can reduce the size of this structure by as much as a constant factor, already implies a speed-up by an exponential factor.

[11] An example is [Sei93], which surprisingly is a discussion in the context of feature formalisms and unification, which usually aim at capturing as much as possible in a single representational system.

Part III

Principles

Chapter 8
Abstract generative capacity

This chapter is the first of part **III** of this thesis, in which instead of looking at *what*, *i.e.* what string sets a formalism can theoretically generate, I investigate what one *should*, linguistically speaking, want to be able to describe, and in a limited an abstract sense, *how* one should be able to describe those phenomena. That is, the focus is moved from a theoretical, platonistic view on linguistic capacity, to an *explanatory* view—I demand that the linguistic analyses given by our formalisms shed light on the precise nature of what is analyzed.

Section **8.1** picks up the thread of the chapters **1** and **3** by going deeper into *weak generative capacity* and introducing the notion of MILD CONTEXT-SENSITIVITY (**MCS**) as defined by Joshi in [Jos85], and puts the formalisms and complexity results of parts **I** and **II** in the context of the **MCS** discussion. In this section, I also give a brief list of other phenomena to argue about the structural capacities of grammatical formalisms, than the partial verb clause conjunctions of Manaster-Ramer that dominated the discussion in chapter **3**.

Section **8.2** does the same, but in a less wide scope, for strong generative capacity. It looks at the consequences of demanding a *bounded dependency domain* (defined in chapter **1**) for descriptions in **MHG** and **PMCFG**. The conclusion drawn for the case of **MHG** will be used as a motivation for some properties of the head grammar solutions proposed in chapter **10**.

8.1 Weak generative capacity and the mildly context-sensitive

In chapters **1** and **3**, I gave arguments that the underlying structure of natural languages is beyond the weak generative capacity of **CFG** and even beyond the capacity of the more general class of linear **MCFG**. In chapter **5** it was shown that simple **LMG** describe precisely the class of languages that can be recognized in polynomial time. In this section I make an attempt to put these facts together in a discussion concentrating on finding a class containing precisely those languages that are structurally no more complex than a "natural language". In [Jos85], ARAVIND JOSHI proposed an informal outline of such a class: the *mildly context-sensitive* languages. I will give his definition here, then look at a series of examples and finally discuss other specifications of similar classes.

8-1 deÆnition. [Jos85] A MILDLY CONTEXT-SENSITIVE LANGUAGE (**MCSL**) is a language that has the following three properties

1. limited crossed dependencies
2. constant growth (definition **8-2**)
3. polynomial parsing

$*$

The class of mildly context-sensitive languages is often confused with the class of languages described by the **TAG** formalism; this is because the concept of mild context-sensitivity appears almost uniquely in the **TAG** literature, which often fails to mention stronger formalisms; it is more appropriate to say that **MCS** is most adequately approached by linear **MCFG**.

I will here equate 'polynomial parsing' with the notion of having a polynomial recognition procedure. A better definition is not feasible as long as the definition of mild context-sensitivity is said to talk about *languages* rather than about grammatical formalisms, for a class of languages has no a priori given structure. 'Limited crossed dependencies' is also vague, and I will attempt to replace it here with formal equivalents.

The essential element in the definition proposed by Joshi is the constant growth property, which is defined as follows:

8-2 deÆnition. A language L has the CONSTANT GROWTH PROPERTY if there is a constant c_0 and a finite set of constants C such that for all $w \in L$ where $|w| > c_0$ there is a $w' \in L$ such that $|w| = |w'| + c$ for some $c \in C$.

$*$

Constant growth is intended to roughly capture two properties of natural language: (**i**) it features *recursion* or unbounded *embedding* and (**ii**) such recursions appear, in some sense, in finitely many different basic forms. Two stronger properties that emphasize approximately the same features are k-pumpability as defined in chapter **3**, repeated here as **8-4**, and *semilinearity*.

8-3 definition. A language is called SEMILINEAR if it is letter equivalent[1] to a context-free language, *i.e.*, for any semilinear language M there is a context-free language L such that a word w is in M if and only if there is a permutation v of w in L.

8-4 definition. A language L is UNIVERSALLY k-PUMPABLE if there are constants c_0 and k such that for any $w \in L$ with $|w| > c_0$, there are strings $u_0, \ldots, u_k$ and $v_1, \ldots, v_k$ such that $w = u_0 v_1 u_1 v_2 u_2 \cdots u_{k-1} v_k u_k$, for each i: $0 \leq |v_i| < c_0$, at least one of the v_i is not the empty string and for any $p \geq 0$, $u_0 v_1^p u_1 v_2^p u_2 \cdots u_{k-1} v_k^p u_k \in L$.

*

Both k-pumpability and semilinearity imply constant growth; there is no direct relationship between semilinearity and pumpability. Every linear **MCFL** is obviously semilinear, because a letter equivalent **CFG** can be constructed straightforwardly by eliminating the component boundaries. Not every semilinear language is a linear **MCFL**; Owen Rambow [Ram94] presents a formalism that has a greater weak generative capacity than linear **MCFG** (equivalently, Rambow refers to **MC-TAG**), but still generates semilinear languages.

I will now proceed by giving a series of examples of languages, both simple formal and from natural languages, from the literature, and the properties these languages satisfy.

8-5 example. The unbounded partial conjunctions of proposition **3-15** were shown not to be universally k-pumpable for any k. Along the lines of the same argument, it is easily seen that as an isolated fragment, they also do not have the constant growth property: let c be the largest number in C, then take c conjuncts each of length $c + 1$. As a consequence, they are also not semilinear.

*

The unbounded conjunctions are sometimes argued to be of a not immediately syntactic nature; as anaphoric binding, one can argue that co-ordination has a semantic, or even mathematical quality that is perhaps not wise to include when looking at the capacity of linguistic structure—for example, the human language faculty may be able to tackle these restrictions by a form of multi-stage processing as discussed in chapter **7**.

The following two examples show sets that are not pumpable, not semilinear, but do satisfy constant growth. The first example is clearly beyond the intended capacity of natural languages, opinions on the second example vary. A preliminary conclusion, drawn often in the literature, is that constant growth is a too weak restriction for what it is intended to capture.

[1] Note that *letter* here should be interpreted as *terminal symbol*, *i.e.* usually an entire word when thinking in terms of natural language grammars.

8-6 example. The languages $\mathtt{a}^{2^n}$ and $\mathtt{b}^*\mathtt{a}^{2^n}$ both clearly qualify as unnatural. The first language does not have the constant growth property, but the second language does. Neither of the languages is semilinear, and neither of them is universally *k*-pumpable. The second language is not *k*-pumpable, because the statement of *k*-pumpability includes the possibility to *pump down*, and the if string $\mathtt{ba}^{2^n}$ can be pumped up, then the pumped string must be $\mathtt{b}$; pumping down removes the $\mathtt{b}$ and results in a string outside the language.

*

Various attempts have been made to improve the definition of constant growth, by increasing its detail, but for each of these, further counter-examples were found (see [Rad91]).

8-7 example. [Rad91] In Mandarin Chinese, the highest number unit expressed in a single word is *zhao*, which is 10^{12}. Higher numbers are expressed by series of *zhao*, just like in English one can think of a *thousand thousand* as denoting one million. Let **NC** be the set of well-formed Chinese number names. The fragment of numbers expressed using *zhao* and the word *wu* (five) has the following form:

(8.1)
$$\begin{aligned}\mathbf{J} &= \mathbf{NC} \cap (\mathtt{wu}\ \mathtt{zhao}^+)^+ \\ &= \{\mathtt{wu}\ \mathtt{zhao}^{k_1}\ \mathtt{wu}\ \mathtt{zhao}^{k_2}\ \ldots\ \mathtt{wu}\ \mathtt{zhao}^{k_n} \\ &\qquad\qquad\qquad\qquad \mid k_1 > k_2 > \cdots > k_n > 0\ \}\end{aligned}$$

The set **J** is shown not to be *k*-pumpable for any *k* in [Rad91]. This set will, modulo a finite number of exceptions, be equal to the intersection of Chinese with the same regular set. Hence Chinese is not a linear **MCFL**.

However, the fragment **J** can be generated straightforwardly by a **PMCFG** or simple **LMG**.

*

In [MK96], KRACHT and MICHAELIS show that the fragment **J** is also non-semilinear. However, unlike the proof that **J** is not a linear **MCFL**, this does not imply that the full Chinese language is not semilinear, unless one claims that properties like semilinearity of natural languages are preserved under taking certain well-defined subclasses, as **J** is a well-defined subclass of **NC**. The same holds for the argument using Dutch or German unbounded partial verbal clause conjunctions.

Michaelis and Kracht do show the non-semilinearity of a full natural language, but using a rather complex construction which, as far as I can see, is not trivially applied to the Chinese number names and the verb clause conjunctions; this language is OLD GEORGIAN, and the example has the additional advantage that it is completely free of co-ordination and mathematical properties. Old Georgian features a form of *genitive suffix stacking* that is highly similar to the Chinese numbers: there are noun phrases of the form

(8.2) N_1-*nom* N_2-*gen-nom* N_3-*gen-gen-nom* $\ldots$ N_k-gen^{k-1}-*nom*
("N_1 of N_2 of $\cdots$ of N_k")

where *gen* and *nom* contain suffix elements and N_i are noun stems.

From the discussion so far, the following conclusions can be drawn: on the one hand, the constant growth property is too weak to exclude sufficiently a number of artificial languages that one does not want to permit (but which are recognizable in polynomial time), so one would rather want to replace it with a property like semilinearity or universal *k*-pumpability. These however suffer from two problems: (**i**) they characterize classes of languages that are not closed under taking well-motivated subclasses, where I would like to define 'well-motivated' formally by such operations as intersection with regular sets and applying homomorphisms; and (**ii**) they are slightly too strong, where the best counterexample is probably Old Georgian.

In the literature, it has been argued that what is missing is the capacity to describe forms of *reduplication*, in the form of languages of the form $\{\ w\ h(w) \mid w \in L\ \}$ where L is context-free or regular. In fact, MANASTER-RAMER has even proposed that while reduplication is an essential construction in natural language, the "mirror image" construction $\{\ w\ h(w)^{\mathrm{R}}\ \}$ is not,[2] and a suitable class of languages is obtained by closing the regular languages under forms of reduplication rather than embedding. This seems unlikely, because mirroring is the canonical form of the German verbal clause.

While weakly speaking, reduplication plus applications of homomorphisms and inverse homomorphisms seems to be able to generate crossed dependencies in Dutch, I have doubts as to whether this leads to strongly correct systems (in contrast, probably, to cases such as old Georgian and the Chinese number names). Hence, a class like **PMCFL** seems to be appropriate weakly speaking, but not strongly. This puts the search for a suitably defined class of mildly context-sensitive *languages* in a slightly dubious light—what is really searched for is a class of structural descriptions, but then one runs into the lack of an all-comprising structural meta theory.

To conclude this discussion, I propose to define an alternative class of mildly context-sensitive languages by the following definitions, the most important of which is *finite pumpability*, which is as universal *k*-pumpability, without specifying the fixed integer number *k* in advance. The lack of a proper reflection of underlying structure is in these definitions reflected in the requirement that the class is closed under the **AFL** (*abstract family of languages*) operations.

8-8 deÆnition: Ænite pumpability. A language L is FINITELY PUMPABLE if there is a constant c_0 such that for any $w \in L$ with $|w| > c_0$, there are a finite number k and strings $u_0, \ldots, u_k$ and $v_1, \ldots, v_k$ such that $w = u_0 v_1 u_1 v_2 u_2 \cdots u_{k-1} v_k u_k$, and for each i, $1 \leq |v_i| < c_0$, and for any $p \geq 0$, $u_0 v_1^p u_1 v_2^p u_2 \cdots u_{k-1} v_k^p u_k \in L$.

*

Finite pumpability excludes quadratically and exponentially growing languages, while admitting the conjoined partial verb clauses and Radzinski's Chinese number names.

[2] By convention, w^{R} denotes the mirror image of w, *i.e.*, the string obtained by reversing the terminal symbols in the string w.

It puts the boundary on language growth between the number names and the strict-monotone growing language (8.3) which approaches quadratic growth.[3]

(8.3) $\{\ \texttt{abaab}\cdots\texttt{ba}^{k-1}\texttt{ba}^{k}\ \}$

The following tentative definitions are two ways of further formalizing (**i**) the notion of 'limited crossed dependencies', including a limited form of co-ordination that excludes the 'square' pattern I argued needed to be excluded, and (**ii**) the fact that the pumpability restriction needs to be preserved under taking subclasses and homomorphic projections.

8-9 deÆnition. **O-MCSL** (optimistic mild context-sensitivity) is the largest class of languages that

1. is a substitution-closed **AFL**, *i.e.* closed under union, concatenation, iteration, intersection with a regular set, and substitution.
2. contains only languages that are finitely pumpable
3. is included in **PTIME-DTM**

*

Note that this class exists, and subsumes at least linear **MCFL**, but not all of **PMCFL**. One might raise the objection that this definition is indeed 'optimistic', and might generate languages that we have not envisaged. On the other hand, it needs to be proved that it does not collapse into **MCFL**. Therefore I give the following, again tentative, parametrizable alternative:

8-10 deÆnition. (tentative) A class $\mathcal{C}$ of languages is said to GENERATE A CLASS OF **P-MCSL** (pessimistic mild context-sensitivity) $\mathcal{M}$ if the closure $\mathcal{M}$ of $\mathcal{C}$ under the **AFL** operations contains only languages that are finitely pumpable and is included in **PTIME-DTM**.

*

Candidate members for the class $\mathcal{C}$ are formal equivalents of linguistic constructions like embedding and reduplication, encoded in such a way that dependencies are sufficiently stressed. At least the fragment **J**, and the classes of homomorphic 2-reduplications and homomorphic mirror images over regular languages should be included.

Finally, there is a construction for which it has argued to be unlikely that it can at all be analyzed by a polynomial time algorithm.

[3] I have slight doubts as to whether such a language should actually not be *allowed* to count as not excessively growing, on psychological grounds. It seems clear to me that a human should in general not be able to process a fragment that consists of k^2 letters $\texttt{a}$, but in the same way as we can distinguish whether a number sequence is correctly decreasing, we should be able to check that a series is increasing or decreasing by one occurrence of *thousand* or *zhao* at a time.

8-11 example: scrambling. While the standard order in the German verb clause is embedded, in some cases German shows different orders of the objects; this phenomenon is known as SCRAMBLING—the example (8.4) is taken from [RJ94].

(8.4) a. . . . daß der Detektiv niemandem den Mann des Verbrechens
zu überführen verspricht
b. . . . daß [des Verbrechens]$_k$ der Detektiv [den Mann]$_j$ niemandem
[ε_j ε_k zu überführen] verspricht

"that the detective has promised no-one to indict the man of the crime"

If it is assumed that all objects in the German clause can in principle occur in *any order*, then scrambling is shown to be beyond the capacity of **MC-TAG** in [Ram94]. It is currently unknown whether scrambling in this form can be tractably processed at all. Similar phenomena occur in the Dutch verb cluster, where the order of verbs (as opposed to the objects) can in some cases be reversed. Even the simple **LMG** formalism does not seem to provide any methods that can be immediately recognized as solving such problems, because it still splits up constituents into boundedly many clusters.

In the case of scrambling however, it must be noted that examples from real German texts are obviously all of limited size, and the fact that complements can be scrambled is licensed by the overt morphology of the different cases. I have not come across any examples that shared two or more objects in one of the four German cases that could be swapped, while even simple examples as (8.4b) above are already at the border of acceptability.

Let's assume for now that it is empirically provable that when a German verbal clause contains two object NPs in the same case, *e.g.*, both accusative, then the deeper embedded NP must appear after the higher NP. In this case it is easy to construct a loosely interpreted **XG** along the lines of the examples for Dutch cross-serial clauses (sections **2.2** and **6.2**) with one bracket type for each of the four German cases, that generates precisely the admissible scrambled sentences. So when this restriction is valid, German scrambling can be analyzed in polynomial time!

8.2 Strong generative capacity and the dependency domain

It is left open in the discussion in the previous section exactly what combination of sharing or reduplication with a subset of multi-component rewriting is sufficient for known natural languages. An important question in the chapters to follow is, supposing for example that simple **LMG** is a sensible point of departure to build a principled system based on axioms that further characterize properties of natural language, how many components or *clusters* would be required for the nonterminals in an underlying **LMG** model. Another question which has not really been solved is whether reduplication is really adequate, in a strong or explanatory sense, to describe the examples in the previous section.

Dutch long-distance topicalization and MHG

The following argument shows that if modified head grammar, which splits up constituents into two surface clusters, is to generate a fragment of Dutch sentences with a bounded dependency domain, the resulting structural analyses are unorthodox. Instead of then marking **HG** as 'incapable', I will use this observation in chapter **10** as a motivation for inverting some traditional dominance relations in relative clauses.

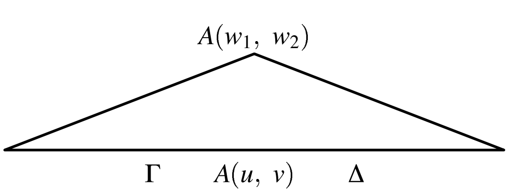
The fragment under investigation in this section is the *wh*-interrogative sentence (8.5), which can be thought of as corresponding structurally to a relative clause, without the requirement that either the extraposed pronoun and the finite verb are boundedly far apart in the structural representation—this is consistent with the idea expressed earlier that localizing selectional dependencies is to be preferred over localizing elements that are 'close' in a surface-structural representation.

(8.5) Wat zag Frank Julia^{k+1} zien^{k} drinken?
what saw see drink

I will now examine systematically how an **MHG** could generate such a small fragment, which, if they are to generate Dutch in the strong sense, must be the case by the classificatory capacity requirement one can reasonably make in the case of an **sgc** test.

Suppose there is an **MHG** of the fragment. Then a sufficiently long member of the fragment must be pumpable, that is, its derivation has a recursive step (8.6).

(8.6)



where $w_1 = t_1(u,v)$ and $w_2 = t_2(u,v)$ can be thought of as terms where u and v are variables. By linearity and non-erasingness, both u and v are projected into w_1w_2

precisely once; moreover since the first component in a **MHG** invariantly appears in the sentence before the second component, the residue of u must precede that of v in $w_1 w_2$.

The rest of w_1 and w_2 must consist of an equal number m of the terminals `Julia` and `zien`. Since all occurrences of `Julia` are to precede all occurrences of `zien` in the sentence, w_1 can contain only `Julia` and w_2 only `zien`.

Now, suppose the analysis has a bounded dependency domain. Then the `Julia`'s and the `zien`'s generated must be separated by boundedly many dependency arcs; that is, the following possibilities remain:

1. $w_1 = \texttt{Julia}^m\, u, \quad w_2 = \texttt{zien}^m\, v$
2. $w_1 = u\, \texttt{Julia}^m, \quad w_2 = v\, \texttt{zien}^m$
3. $w_1 = \texttt{Julia}^{m_1}\, u\, \texttt{Julia}^{m_2}, \quad w_2 = \texttt{zien}^{m_1}\, v\, \texttt{zien}^{m_2},$

Now suppose that *drinken* is produced at the bottom of the derivation. Then because of locality, so must *wat*. But *drinken* must appear right of all occurrences of *zien*, and *wat* must appear left of all occurrences of *Julia*. This leads to a contradiction. The same holds for *wat*.

Therefore, if the fragment is to be produced by an **MHG** with a bounded dependency domain, the deepest embedded verb *drinken* and its complement *wat* must be generated at the *top level* of the derivation! This may be counter to one's intuition, but is not surprising, since it is directly linked to the topic *wat* of the sentence. One is now left with two possibilities.

1. *zag* and *Frank* are produced below the recursive step; this implies that the whole derivation is "turned up side down" with respect to traditional analyses:

$$\dfrac{\langle\, \texttt{zag Frank Julia}, \varepsilon \,\rangle}{\langle\, \texttt{zag Frank Julia Julia}, \texttt{zien} \,\rangle}$$

$$\vdots$$

$$\dfrac{\langle\, \texttt{zag Frank Julia}^{k+1}, \texttt{zien}^{k} \,\rangle}{\langle\, \texttt{wat zag Frank Julia}^{k+1}, \texttt{zien}^{k}\, \texttt{drinken} \,\rangle}$$

2. *zag* and *Frank* are produced above the recursive step. This results in a derivation that has both the deepest embedded verb and the main verb at the top of the derivation, and whose bottom is in the middle of the verb embedding.[4]

$$\dfrac{\langle\, \texttt{Julia}, \varepsilon \,\rangle}{\langle\, \overbrace{\texttt{Julia Julia }\underbrace{\texttt{Julia}, \texttt{zien}}\texttt{ zien}} \,\rangle}$$

$$\vdots$$

$$\dfrac{\langle\, \texttt{Julia}^{2k+1}, \texttt{zien}^{2k} \,\rangle}{\langle\, \texttt{wat zag Frank Julia}^{2k+1}, \texttt{zien}^{2k}\, \texttt{drinken} \,\rangle}$$

[4] The braces indicate which occurrences of *zien* and *Julia* are connected through immediate dependency.

with an extra step to produce verb embeddings of odd depth—this is redundancy in the grammar.

The first of these remaining options is investigated in chapter **10**.

Reduplication and partial verb clause conjunctions

I have expressed, several times, doubts as to whether in the strong sense, reduplication can be the mechanism responsible for generating the parallelism in the partial verb clause conjunctions in the examples in chapter **3**. Furthermore, the proof that **PMCFG** generate this set weakly, is based on the proof in [SMFK91] which, as said in the conclusion to chapter **3**, lacks detail.

A **PMCFG** that generates the conjunctive phrases was argued to exist on the basis of a grammar generating a structural backbone, and then applying an inverse homomorphism to produce the actual set. The first objection one might raise is that the set of noun phrases in Dutch is not a regular set, so a homomorphism will not do, and one must apply a substitution. If Kasami et al.'s proof is correct, this objection does not present a problem.

The details of argumentation on **PMCFG** are dazzling, because pumping can be applied, but because of nonlinearity, the pumping lemma does not take a concrete, concatenative form. However, it seems that a strong generative capacity analysis will show that a **PMCFG** that describes Dutch sentences necessarily has a number of components for nonterminals that is proportional to a function that is at least exponential in the number of lexical entries for verbs (or noun phrases, if the fragment is limited to single-word lexical NPs). It needs to produce all possible orders of the finitely many lexical entries for raising verbs. The argument then naturally leads to a contradiction when requiring a bounded dependency domain.

It now depends strictly on what one takes to be the "underlying structural basis of natural language" to establish whether one thinks of reduplication as being what is really happening in the case of partial verb clause conjunctions. If it is, then surely this would establish the necessary existence of a further step of syntactic processing after analyzing the underlying structure of a conjunctive sentence.

Conclusions to chapter 8

There are two conclusions to this section that strongly influence the approaches in the rest of part **III**. The first is, that in a *principle-based* approach to grammar specification, a form of the mild context-sensitivity definition can be taken as a core set of axioms. I suggest to consider a number of the aims expressed in the **Prologue** and the first chapter of this thesis also as axioms, which then leads to the idea that simple **LMG** is a reasonable generic framework which could be further restricted through principles and parameters so as to finally result in a grammar with a reasonable amount of explanatory quality, and for which some computational tasks are tractable. This is done in chapter **10**.

The loose **XG**-account of scrambling is new, but works only for a restricted case, which is hard to evaluate empirically because of the difficulty of scrambling example sentences. What was found here is not dissimilar to results of analyses done by OWEN RAMBOW ([Ram94] and p.c.). From the existence of a loose **XG** analysis, it follows that $\mathcal{S}$-**LMG** can also *weakly* generate scrambling, but a strongly correct analysis is unlikely.

If the closure result of **PMCFL** under inverse homomorphism is indeed found to be true, then for all examples except scrambling, it may be sensible to add that *strongly*, simple **LMG** is a good point of departure, but *weakly*, the languages under investigation should be in **PMCFL**. Adding reduplication to simple **LMG**, which is a conservative extension from a weak perspective, may further increase its relevant strong capacity.

Another, minor, conclusion following from the second section of this chapter has had influence on the proposed method of describing relative clause attachment in chapter **10**.

Chapter 9

Government-Binding theory

What is a systematic description of a language? Clearly, this is not just a list of the correct sentences of the language, because such a list does not have any explanatory qualities. A rule-based grammar as I have shown examples of in parts **I** and **II** is better, but suffers from the same problem: such grammars seem to describe a language fairly accurately, but some sets of rules show similarities which are not explained, and the question remains *why* the rules look like they do. A *principle-based* grammatical framework tries to formulate general properties of language from which the linguistic structures, or the rules that describe these structures, can be derived.

An example of a generalization missed in a rule-based grammar is that in English, all verbs tend to immediately precede their objects. Although every rule in the grammar constructing verb phrases is consistent with this observation, the rule itself is not stated. Among many other theories, such as **GPSG** [GKPS85], GOVERNMENT-BINDING (**GB**) THEORY captures such word order generalizations, by giving rules only for phrase containment, and specifying word order constraints separately. Note also that phrase containment is subject to less variation among different languages, whereas whether complements precede or follow their heads depends strongly on the language under investigation. Therefore such a separation also helps dividing the components of a grammatical description into universal and language-particular properties, and as a side effect helps understanding multi-lingual analysis and (automated) translation.

*

In its presentation in parts **I** and **II**, literal movement grammar and its way of describing surface order lacks such a profound principled background. It was introduced without much motivation and its properties were investigated in a theoretical fashion that looked primarily at the resulting languages, and even this was done mostly on the basis of *weak generative capacity*.

This chapter on **GB** theory, and the next, about FREE HEAD GRAMMAR, will make the move from the practice of directly writing the grammar clauses, to an *axiomatic* system where the precise form of the grammar is not given but rather follows from a set of constraints, divided into PRINCIPLES of language in general and PARAMETERS restricting the grammar to a particular language or a fragment of it.

Such axiomatized systems aim at giving a set of properties that language must satisfy, and at providing a concise motivation for the exact statement of these properties. Both **GB** theory and forms of head grammar (which, with its close relative **GPSG**, was superseded by **HPSG** [PS94]) are such *explanatory* systems, and in particular, they offer explanations for what exactly is the rôle of *movement* in linguistic structure—

which is exactly what part **III** of this thesis is after.

This chapter represents the *conservative* approach to finding a principled basis for the tuple-based description of movement. It will introduce the reader to some concepts of the principled, explanatory approach in general and of **GB** theory in particular. Benefits of such a conservative approach are that (**i**) it is good to have a way of assessing the value of the grammar formalisms from part **I** and what is done in the principle-based tuple grammar system **FHG** in chapter **10** and (**ii**) **GB** provides, among many theories, the most satisfactory, and the farthest elaborated empirical analyses, so it is worthwhile to investigate how results of generative Linguistics can be reformulated in principled accounts based on literal movement.

The account given here is a very sketchy, simplified version of a framework in Government-Binding *style*, and in order to allow for a short presentation, it makes use of non-standard terminology. A good textbook is [Hae91] (on which this chapter is loosely based).

9.1 Phrase structure and X' syntax

GB is a phrase-structure based, strictly hierarchical, projective theory. That is, one of its most basic principles is that a sentence has a SURFACE STRUCTURE that can be drawn as a tree diagram spelling out the sentence from left to right at its leaves, or equivalently, by means of LABELLED BRACKETS. A complete labelled bracketing as in (9.1) fully specifies a tree structure; however, in practice, one often looks at partially specified trees by omitting brackets (9.2).

(9.1) $[_{\mathrm{S}}$ $[_{\mathrm{NP}}$ Frank] $[_{\mathrm{AUX}}$ will]
$[_{\mathrm{VP}}$ $[_{\mathrm{V}}$ poor] $[_{\mathrm{NP}}$ Julia] $[_{\mathrm{NP}}$ $[_{\mathrm{Det}}$ a] $[_{\mathrm{N}}$ cup] $[_{\mathrm{PP}}$ $[_{\mathrm{P}}$ of] $[_{\mathrm{NP}}$ $[_{\mathrm{N}}$ tea]]]]]].

(9.2) $[_{\mathrm{S}}$ $[_{\mathrm{NP}}$ Frank] $[_{\mathrm{AUX}}$ will] $[_{\mathrm{VP}}$ poor Julia a cup of tea]]

Furthermore, categories and phrases play rôles similar to those sketched in section **1.3** of the introduction.

9-1 principle: surface structure. Every sentence has a projective surface structure.

9-2 principle: categories. Words belong to SYNTACTIC CATEGORIES like nouns and verbs. There are TERMINAL CATEGORIES X (those of words) and corresponding PHRASAL CATEGORIES X', XP.

*

The notion of *selection* mentioned briefly in the introduction, is called *subcategorization* in **GB** theory, and is restricted to terminal categories, *i.e.*, to words in the *lexicon*. Subcategorization on **GB** consists of ARGUMENT STRUCTURE as well as a further refinement, θ-theory of THEMATIC STRUCTURE, which aims to explain *why* a word selects certain complements by assigning the complements rôles such as *actor*, *patient*. I will gloss over thematic structure here because it can be considered optional in this and the next chapter.

Each lexical entry for a word contains a subcategorization frame, that is a list of category specifications the word *selects for*.

9-3 principle: Lexicon and selection. There is a LEXICON specifying the (terminal) categories and properties of words. A lexical entry specifies a SUBCATEGORIZATION FRAME; The subcategorization frame specifies a finite number of ARGUMENTS, their category, and in the case of an NP arguments, optionally specifies a case assignment.

*

An important component of **GB** also often referred to by other frameworks is X'-theory, which says that there may be levels between a phrasal category XP and the lexical equivalent X. In general, one assumes a single intermediate level X' (“x-bar”). These levels are then characterized by three rules that specify phrase containment, but no order:

9-4 principle: X′ projection. Phrase containment satisfies the following dominance restrictions, for any basic categories X, Y, Z, S.

$$\begin{array}{lcl} XP & \rightarrow & SP;\ X' \\ X' & \rightarrow & X';\ YP \\ X' & \rightarrow & X;\ ZP_1\ ZP_2, \ldots \end{array}$$

X is the HEAD of XP; SP is the SPECIFIER and ZP_i are the ARGUMENTS of the head X.

*

The first rule of the X′ schema says that a phrasal category consists of an obligatory SPECIFIER and an intermediate level. At the intermediate level, other phrasal categories are optionally adjoined (such as adjectives or adverbs). The third rule says that an intermediate level consists of the terminal head and its complements as defined by its subcategorization information in the lexicon. In all these rules, the order is not specified, but the specifier, adjuncts and complements either *precede* or *follow* the X-projection, so operations like *wrapping* as discussed earlier are not considered.

Precisely what phrasal categories are allowed as specifiers and adjuncts, as well as whether these precede or follow, is considered a language-dependent *parameter* of the grammar. In English for example, the specifier precedes the head, and the complements follow it.

For some languages, the X′ view is problematic, because their word order is too free to fit in the model. It is often claimed that those languages are *non-configurational*, that is, they do not have a hierarchical structure. In the next chapter, I will discuss Latin, and show that when the X′ structure as posited by **GB** is less tightly connected to surface order, hierarchy and free word order do not have to be contradicting.

Finally, important principles describe how the presence of words is licensed by the presence selecting or case-marking heads. Subcategorization and the lowest X′-rule interact as follows: briefly speaking, everything (say a noun phrase) that occurs in a sentence must be there because something else, say a verb, selects for it; and everything can be selected at most once.

9-5 principle: theta criterion. Every maximal projection in an argument or specifier position must be subcategorized for by precisely one lexical head.

*

The presence of NPs is further restricted; each noun phrase must have a case, and this case is assigned by its environment.

9-6 principle: case Ælter. Every NP must be assigned case.

9.2 Movement and adjunction

Government-binding theory chooses to assume a rigid relation between structure and word order, and describes word orders that do not conform directly to its projectivity constraints through MOVEMENT. An example is the following. A sentence contains an INFLECTIONAL PHRASE IP. The specifier of this IP is the subject of the sentence, and the head has the category I or INFL (for *inflection*). INFL is sometimes an auxiliary such as in (9.3), but in the majority of cases, it is just the verbal tense/agreement marker (9.4a).

(9.3) [$_{\text{IP}}$ [$_{\text{NP}}$ Frank] [$_{\text{I}}$ will] [$_{\text{VP}}$ [$_{\text{V}}$ poor] [$_{\text{NP}}$ Julia] [$_{\text{NP}}$ a coffee]]]

(9.4) a. [$_{\text{IP}}$ [$_{\text{NP}}$ Frank] [$_{\text{I}}$ -ed] [$_{\text{VP}}$ [$_{\text{V}}$ poor] [$_{\text{NP}}$ Julia] [$_{\text{NP}}$ a coffee]]]
(Frank poored Julia a coffee)

In the second sentence, the past tense marker *-ed* is attached to the verb *poor*. This is traditionally done by LOWERING the tense marker to the V node, resulting in (9.4b), or alternatively by RAISING the verb to INFL. In both cases, one gets a node of the form [$_{\text{V}}$ [$_{\text{V}}$][$_{\text{I}}$]], or alternatively [$_{\text{I}}$ [$_{\text{V}}$][$_{\text{I}}$]], that does not correspond to the X' principle **9-4**; this is called *adjunction*.

(9.4) b. [$_{\text{IP}}$ [$_{\text{NP}}$ Frank] [$_{\text{I}}$ ε] [$_{\text{VP}}$ [$_{\text{V}}$ [$_{\text{V}}$ poor] [$_{\text{I}}$ -ed]] [$_{\text{NP}}$ Julia] [$_{\text{NP}}$ a coffee]]]
(Frank poored Julia a coffee)

9-7 principle: adjunction. The following non-X' rules are allowed under certain conditions:

XP → XP; YP (phrasal adjunction)
X → X; Y (head-to-head adjunction)

Another example of movement in **GB** is the following refinement: the IP is contained in a COMPLEMENTIZER PHRASE CP, whose head is of category C or COMP as in (9.5), but in a *yes-no* question such as (9.6), the head *will* of the IP is raised to the position of the C.

(9.5) [$_{\text{CP}}$ [$_{\text{C}}$ that] [$_{\text{IP}}$ Frank will poor Julia a coffee]]

(9.6) a. [$_{\text{CP}}$ [$_{\text{C}}$ $\varepsilon_{y/n}$] [$_{\text{IP}}$ [$_{\text{NP}}$ Frank] [$_{\text{I}}$ will] [$_{\text{VP}}$ [$_{\text{V}}$ poor] [$_{\text{NP}}$ Julia] [$_{\text{NP}}$ a coffee]]]]
b. [$_{\text{CP}}$ [$_{\text{C}}$ Will$_{y/n}$] [$_{\text{IP}}$ [$_{\text{NP}}$ Frank] [$_{\text{I}}$ ε] [$_{\text{VP}}$ [$_{\text{V}}$ poor] [$_{\text{NP}}$ Julia] [$_{\text{NP}}$ a coffee]]]]

Hence **GB** theory arrives at more than one structure for each linguistic expression. The structures (9.4a) and (9.6a) are called the DEEP STRUCTURE or D-structure, and the analyses (9.4b) and (9.6b) are called the SURFACE STRUCTURE or S-structure.

The difference between these structures is minimal; the movement operations that transform D-structure to S-structure are said to be STRUCTURE-PRESERVING, and there are very strict rules of what is allowed to move to what positions.

So on the one hand these structures are almost identical, but on the other this notion of *transformation* has consequences for (**i**) the readability of grammatical explanations and (**ii**) their computational tractability. Therefore it is worth looking at whether the *literal movement* from previous chapters can be used to obtain a version of the **GB** theory that has only one of the two structures. This is done in section **9.4**.

The complete scheme of the structural representations recognized in **GB** is drawn in *figure 9.1*; the actual situation is that one structure is derived from the other through a number of successive movement operations, so in a computational model, several more structures may be constructed. Also, **GB** posits a form of *derivation* of the structures that corresponds to the way the human brain constructs a linguistic expression—hence the term *generative linguistics*. In this point of view it is essentially harder to look at the opposite construction in a simple and elegant way—*i.e.* the perspective of the listener or reader, who has to analyze an expression that is received as spoken or written text without any given structure: **GB** *parsing* is a difficult and relatively underdeveloped field of research.

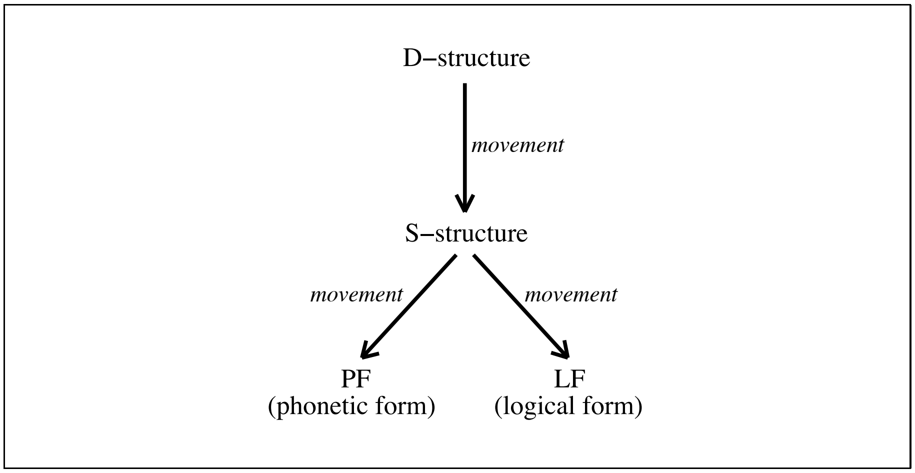


Figure 9.1: Structural representations in **GB**.

Chomsky et al. have recently developed successors of the **GB** theory (the MINIMALIST PROGRAM, [Cho96]) that take away some of these problems, but are still abundantly dependent on ‘movement’ in the sense of transformation of structures. One of the differences is that structures are not *moved* but rather *copied*. Also, the scheme of structural representations has been simplified, and a more direct path is now suggested to connect PF and LF.

9.3 Examples

The leading example in this thesis, a simple selection of sentences with so-called RAISING VERBS[1] in English, German and Dutch, is a good illustration of a fragment of **GB** grammar that shows (**i**) the way universal principles and language-specific parameters are used and (**ii**) how abundant use is made of movement constructions.

To make this a smooth story, I shall look, initially, only at two types of verb; transitives and raising verbs. The latter already occurred in previous parts of this thesis, but the term was not explained. In this simple account, the verbs can be given the following SUBCATEGORIZATION FRAMES[2]

(9.7) $\langle \textit{transitive verb} \rangle : \quad (1)\ \mathrm{NP},\ (2)\ \mathrm{NP}[\text{assign } \textit{acc}].$

(9.8) $\langle \textit{raising verb} \rangle : \quad (1)\ \mathrm{NP},\ (2)\ \mathrm{IP}[\text{assign } \textit{acc} \text{ to subj}]$

For clarity, and as done before, in tree diagrams I will give these verbs the categories V^{T} and V^{R}, respectively. In **GB**, it would be sufficient to say that they have the category V, and the subcategorization frames given above.

As said, these subcategorization frames license certain deep structures, without specifying a particular order at the bottom of the verbal projections. To specify such an order, one can give LINEAR PRECEDENCE specifications. For English, one can give the very general parameter **ELP**, with an amazing amount of accuracy.

9-8 parameter: ELP. Complements in English follow their heads.

*

Hence, the familiar English structure in *figure 9.2* is licensed. Note that the subject plays a special rôle, since it is subcategorized for by the verb, but does not appear in the bottom of the verbal projection, but under the IP. This is because it is not assigned its case by the verb, but by INFL, and case can, roughly, only be assigned 'downward' in the tree structure.

The German case is similar; **WGLP** swaps the argument order as suggested in the introduction; a resulting tree[3] is in *figure 9.3*.

9-9 parameter: WGLP. Complements in West-Germanic languages precede their heads, with the exception of the complements of C.

[1] The verbs called V^{R} in this thesis are, syntactically speaking, only raising verbs in an analysis of Dutch, and should not be confused with English raising verbs, such as *to seem*, in the **GB** literature. On the other hand, they are called raising in Pollard's **HG** program ([Pol84], chapter **10**) too, motivated by semantic properties.

[2] These frames are not presented as they usually are in **GB** theories, because I do not include a form of θ theory.

[3] This analysis is known as the *base-generated* analysis of the German verb phrase; other analyses contain a verb cluster like the analysis of Dutch sketched below. Note that the IP might here (at the current level of detail) just as well have been collapsed completely into the VP, making the subject VP-internal and base-generated in the right position for subordinate clauses.

*

In the German and English cases, the S-structure for relative clauses is nearly identical to the deep structure, be it that the verbal inflectional suffixes are lowered to the verbs.

This is as far as underlying structure goes, without the need of transformations to get to the surface order. A frequently occurring phenomenon is rightward EXTRAPOSITION, discussed here for the Germanic VP. Sentence (9.9) disobeys the constraint that complements in German precede the verb.

(9.9) . . . daß Frank [Julia ε_i] verbot [Kaffee zu trinken $]_i$

The verb *verbot* is called an EXTRAPOSITION VERB, because the embedded VP *Kaffee zu trinken* here is assumed to be obligatorily raised to a position right of the IP, by phrasal adjunction $[_{\mathrm{IP}}\ [_{\mathrm{IP}}\ ]\ [_{\mathrm{VP}}\ ]]$, creating a position that did not exist in the D-structure. This breaks the structure-preservingness suggested in the previous section, be it in a way that is usually ‘approved’ by **GB** theorists.

*

The analysis of Dutch is, as before, more complex than that of German and English;

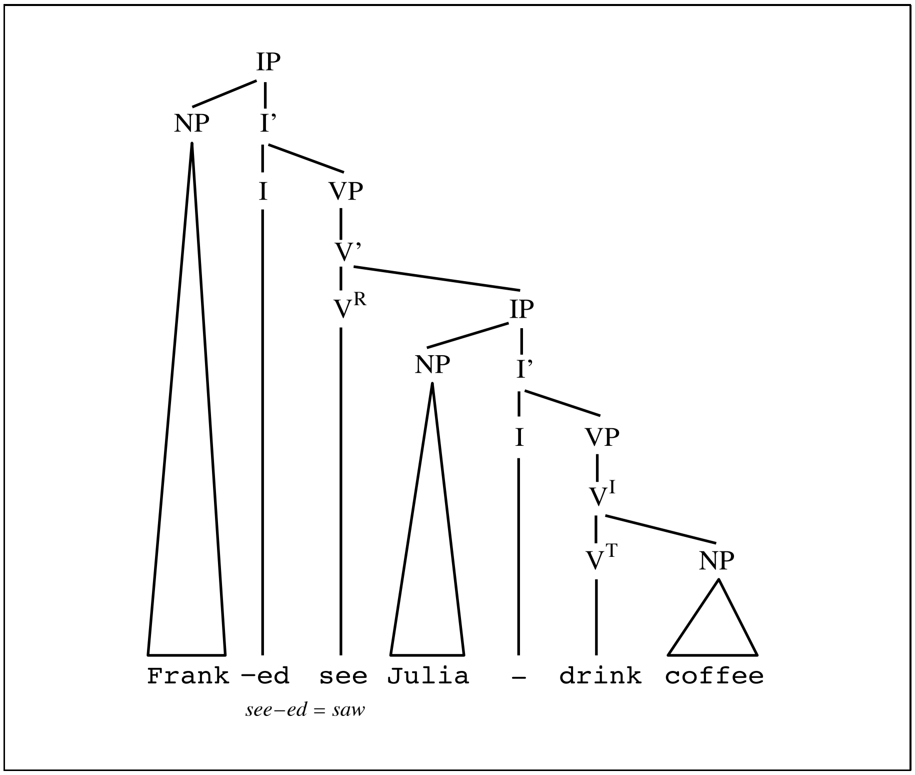


Figure 9.2: D-structure for English

here, head-to-head adjunction is required to form even the basic structure of the verb phrase. Conventionally, an SOV structure like that of German is assumed; however, the order of the verbs in Dutch is reversed w.r.t. the German analysis. This is most often described by a process called VERB RAISING [Eve75]. So, the Dutch deep structure is like that of German, but the S-structure is as in *figure 9.4*.[4] Apart from equivalents of the German extraposition verbs (9.11), there are so-called PARTIAL EXTRAPOSITION VERBS [dBR89] that optionally allow only *part* of the embedded VP to be phrasally right-adjoined to IP (9.12). For comparison, the standard cross-serial raising case is repeated here in simplified schematic form as (9.10).

(9.10) . . . dat Frank Julia [koffie ε_i] zag [drinken]$_i$

[4] An interesting alternative seems to me, but apparently not to the literature on Dutch sentential complementation, to assume an SVO order and adjoin the *objects* to the right of [Spec, IP]. This corresponds more closely to the analysis found elsewhere in this thesis, and has a number of advantages, namely that the movement it involves is less anomalous, because it resembles movement to [Spec, CP].

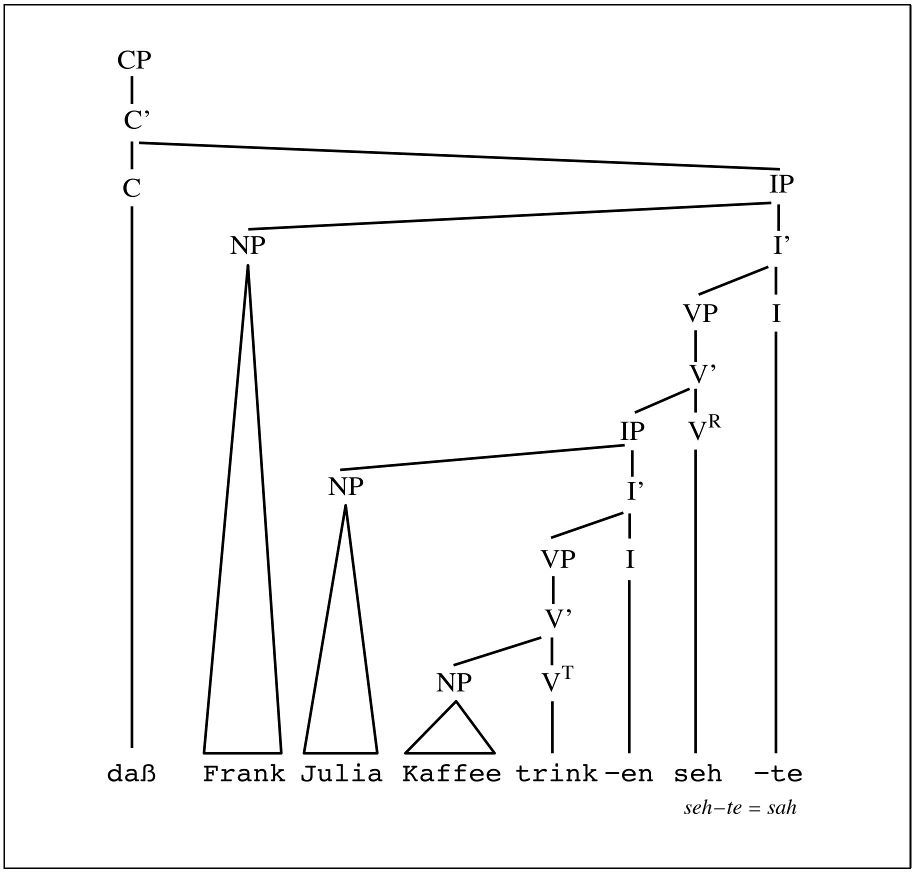


Figure 9.3: D-structure for German

(9.11) a. . . . dat Frank Julia ε_i verbood [koffie te drinken $]_i$
b. $*$. . . dat Frank Julia [koffie ε_i] verbood [te drinken $]_i$

(9.12) a. . . . dat Frank ε_i probeert [Julia koffie te geven $]_i$
b. . . . dat Frank [Julia ε_i] probeert [koffie te geven $]_i$
c. . . . dat Frank [Julia koffie ε_i] probeert [te geven $]_i$

Partial extraposition is one of many problematic cases that lead to various highly involved proposals for the description of Dutch.

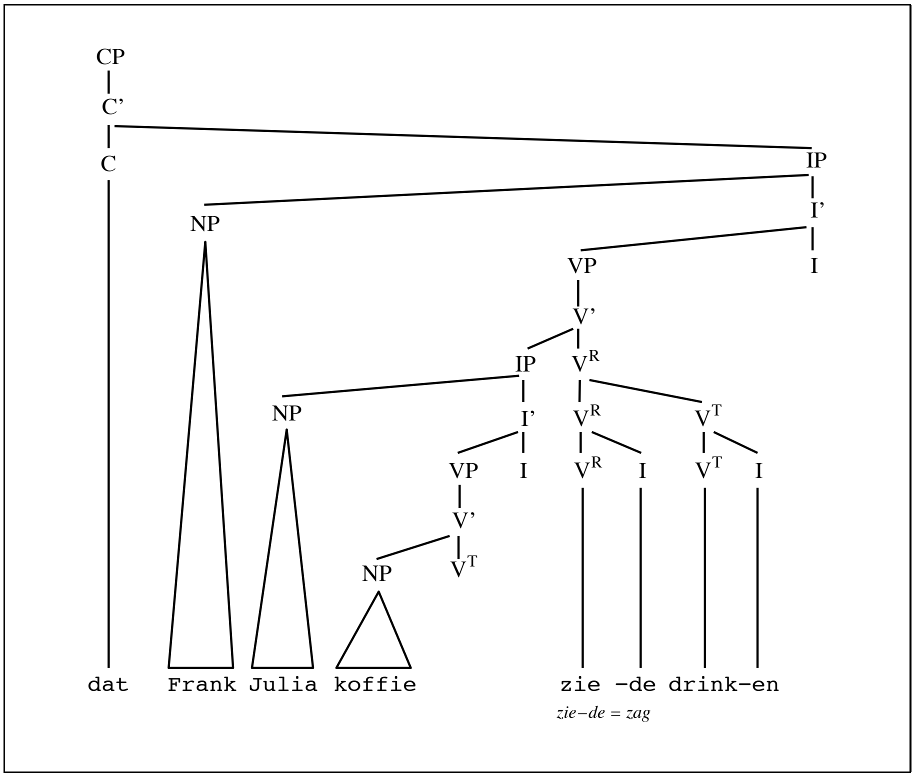


Figure 9.4: S-structure for Dutch

9.4 Replacing S-structure with literal movement

I now sketch how literal movement may be used to implement a theory or grammatical fragment within the **GB** program in a way that is as *conservative* as possible—*i.e.*, respecting the most possible concepts of **GB** theory. The most important motivation for such a sketch is that grammars with a bounded feature domain (section **7.1**) and literal movement are known to guarantee, at least from a formal perspective, tractable computer analysis.

Of course, implementation, weak and strong generative capacity of **GB** descriptions have been studied in the literature. Most such approaches rely on the concept of GRAMMAR CODING, which is, informally speaking, to code the relevant features of the grammatical categories into the set of nonterminals of an underlying grammar, as explained in remark **7-7** on page 136. In particular, KRACHT [Kra95] and ROGERS have independently developed formal elaborations from which it can be concluded that in the absence of head-to-head movement (see below), variants of **GB** can be coded into context-free grammars. The relevance of what is done in this section is twofold: (**i**) grammar coding is a notion that is, initially, of formal significance only, because the resulting coded grammars have nonterminal spaces of intractable sizes; (**ii**) it seems that restricted forms of head-to-head movement are harmless; while the literal movement characterization of this section is less general and lacks detail, it provides hints as to how one might capture some of these acceptable exceptions.

A principle of **LMG** descriptions independent of the motivating linguistic framework, an axiom so to speak, is that the terminal yield of a constituent is generated at its deep-structural position. The **LMG** clusters provide a way to carry this yield, which may be any tuple of terminal strings, to its landing site or sites. Along with this terminal string however, a number of features of the constituent may need to be carried along to be matched against constraints on the landing site. The expected benefit of using literal movement in this way is that the number of features to be 'carried' in this way will be less[5] than in the more traditional approach to grammar coding (of **GB** or unification-based theories), in which a constituent is generated at its final landing site, and *all* its features are carried to, or even structure shared with, its deep-structural position.

The following types of movement are of essential relevance; assume a parametrized set of **GB** principles **T** that describes a fragment of a language, the types of movement in which are limited to the following;

1. *Head movement*, mostly subject-auxiliary inversion: the auxiliary is moved from I to C.

2. *Phrasal movement*, mostly NP-movement;

    (a) *wh-movement and relatives* Complete NP trees are moved to [Spec, CP].

[5] An important question to tackle next is how to automatically deduce from a grammar which features do not need to be carried over long distances. There are numerous techniques for such *dependency analyses*, and I will not discuss those here.

(b) *seems-raising* NPs are moved from [Spec; IP] to [Spec, IP].

(c) Other categories, *e.g.* PP (disregarded here).

3. *Head-to-head adjunction*

(a) *Lowering of inflection* The inflectional marker is lowered to V by head-to-head adjunction.

(b) *Verb raising* in Dutch

The first two types of movement, are strictly structure-preserving: the landing sites are positions in S-structure that are already present in D-structure.

The **LMG** backbone for **T** will have some form of link between nonterminals and category or feature specifications, presumably at the level of basic categories plus their bar level, *i.e.* NP, V′, C, *&c.* Formally, take a number of basic categories, and for each such category X, let N contain $X^0 = X$, $X^1 = X'$ and $X^2 = XP$. The terminals will represent what is in the lexicon. For any possible clause R, the principles of the grammar either license it, and yield a refinement in the form of a feature specification, or refute it. The clause set P of the **LMG** is then the set of licensed clauses. One can go into different levels of detail of this licensing operation, but I will here be especially interested in the following: how many SURFACE CLUSTERS does each nonterminal get,[6] what are these used for, *&c.* I am also interested in clusters of finite attributes with little interaction (called *feature buckets* here) as suggested on page 159—these will often be associated with the surface clusters.

1. subject-auxiliary inversion. This is a relatively simple case; this type of movement is highly local. A technique similar to head-wrapping as in **MHG** is sufficient here. So in the basic setup, each nonterminal X^i has at least three clusters L, H and R, reading out the YIELD of the (traditional) S-structure as L–H–R, and where H is the (lexical) head X. There will be two types of clause that embed IP into CP; a 'normal' clause[7]

$$\mathrm{C}'(\varepsilon,\ c,\ l\ i\ r)\ \mathbf{:-}\ \mathrm{C}(\varepsilon,\ c,\ \varepsilon),\ \mathrm{IP}(l,\ i,\ r).$$

and one that is responsible for inversion:

$$\mathrm{C}'(\varepsilon,\ i,\ l\ r)\ \mathbf{:-}\ \mathrm{C}(\varepsilon,\ \varepsilon,\ \varepsilon),\ \mathrm{IP}(l,\ i,\ r).$$

Selection of which clauses apply in given contexts then depends on feature configurations derived from other principles this account does not go into.

2. NP-movement. For this to be modelled in an **LMG** we must argue that each node in D-structure can be "passed by at most one thread of wh-movement", defined

[6] Note that the *arity* of a nonterminal symbol is not part of the definition of an **LMG**—but by convention, **LMG**s will often use a nonterminal only with a fixed number of arguments.

[7] The appearance of ε in the RHS of this clause is not beyond the capacity of simple **LMG**; it can be circumvented by replacing each occurrence of ε with a variable x and adding a nonterminal Empty with a single production Empty(ε).

formally as follows: a landing site (filler) and its trace form a CHAIN. This chain is said to PASS THROUGH a node if the node is on the shortest path between two nodes in the chain. Now, what needs to be required is that through each node in the structure (D or S, at this point equivalent), are passing BOUNDEDLY MANY chains.

This seems to be guaranteed, for the indicated types of movement, by variants of the SUBJACENCY PRINCIPLE:[8]

9-10 principle: subjacency. Movement cannot cross more than one node of category IP or NP.

Given this principle, a node *n* in a tree structure, and a type of landing site such as [Spec, IP], the size of the LANDING DOMAIN of nodes of this type in the tree structure that *n* could possibly be moved to, is bounded by a constant.[9]

It is now an easy step to provide some characterization for such landing sites. For example, the nodes of type [Spec, IP] are divided into those that are immediately above *n*, or those that have one ancestor labelled IP in between. Each of these finitely characterized classes shall in general need to be assigned a surface cluster. Along with this surface cluster, one shall need to reserve a "feature bucket" that can contain the relevant features (*e.g.* if an NP is moved, the features relevant for an NP at its landing site, except those employed for purposes of movement).

An argument against the claim that the number of movement chains 'passing through' a given node is bounded often presents counterexamples such as (9.13) [PS94]. Here, *Sandy* [Spec, CP] and *Felix* [Spec, IP] are classified differently; this also applies to other examples I have found in the literature.

(9.13) Sandy$_i$, Felix$_j$ was hard for Kim to give ε_j to ε_i.

3. Head-to-head adjunction. This third type is anomalous: the original head-to-head adjunction version of verbal suffix lowering is not structure preserving: it generates extra structure under V. There are two solutions to this. The first takes account of the fact that this lexical 'complex verb' is often taken to be sub-structural. An **LMG** analysis will then have to generate the affix at the V position, but will include the features corresponding to the affix only in the head feature bucket, and not locally. The status of the feature bucket is marked as **+lowered**. The second solution bans *lowering* altogether and looks at solutions (of which there are several in the literature) that look at versions of **GB** that consider *raising* only.

The phenomenon of verb raising in Dutch can also be given a non-structural analysis, by raising verbs one embedded VP projection at a time, and concatenating them to the right of the head of the parent VP, generating a NON-LEXICAL head cluster.

[8] The subjacency principle is subject to variation across languages; but the variants that differ from the one for English (*e.g.* Italian, where IP becomes CP), do not seem to break the conclusions drawn here.

[9] Strictly speaking, additional principles need to be stated to guarantee that this works completely, such as: within each domain of a bounding node IP or NP, movement always occurs to the highest node of a given type.

Conclusions to chapter 9

This chapter served, in the first place, to introduce principle-based grammar. A framework like **GB** cannot be compared to the work on **LMG** and tuple-based formalisms presented in parts **I** and **II**, because the latter do not provide fundamental linguistic explanations. But as **GB** is a principled framework based on a notion of phrase structure that stems from the study of context-free grammars, a theory can be developed that is built on a tuple-based structural backbone. This is done in chapter **10**.

An alternative line that was investigated briefly in section **9.4**, is to use **LMG** as a platform for implementing **GB** theory, with the particular property that S-structure vanishes, to leave D-structure as the only structural representation.

Such a theory is more conservative than a direct formalization, as in chapter **10**, of the intuitions given by tuple grammars, but has the benefit that the results of 'traditional' research in generative linguistics can be transferred fairly easily to a framework based on literal movement, with the additional benefit of, at least theoretically, tractable implementations. Of course, these tractable implementations still have to be constructed, and remain to be shown efficient in practice. Moreover, there is a vast amount of details whose interaction with the proposed simplification of abandoning S-structure needs to be investigated.

To conclude, one further remark is important to make. In this chapter I have looked at S and D structure only, and the standard approach to talking about LF and PF does not fit into the account based on literal movement. But as LF is derived only when S-structure and D-structure are given, perhaps this does not have to be considered a problem—one could think of that phase as post-processing, one could decide to do it altogether in a different way (more standard PTQ style). Along the lines of reasoning from chapter **7**—successive reduction of the number of analyses—one could say that the S/D-analysis filters out 'crashed' derivations, so in the end it is more economical too. The same argument "for the time being, we will consider this an extra-syntactical part of linguistic analysis" can be applied to BINDING THEORY, that also hasn't been discussed here.

Chapter 10
Free head grammar

The previous chapter gave a rather *conservative* approach to finding a principled background for tuple grammars, by looking at how tuple-driven surface generation can be used to support tractable implementations of **GB** theory. However, it was one of the points of departure mentioned in the **Prologue** (conjectures **H** and **J**) to avoid any design implications of an emphasis on the description of English, which is present in **GB** as in many other theories with broad empirical coverage like **HPSG** and **LFG**—in particular the important rôle played by a projective, left-to-right ordered surface structure.

In this chapter, I take the more *progressive* approach of investigating the possibility to provide a *direct* foundation for the use of tuple grammars. The investigation in this chapter is based on Pollard's HEAD GRAMMAR (**HG**), which can be thought of as claiming that the position of the head in a phrase is a psychologically well-chosen mark that could enable humans to split up the "strings" generated in their minds. The FREE HEAD GRAMMARS presented here take the liberating effect of Pollard's generalization from concatenation to operations on headed strings a bit further, by allowing (**i**) slightly more complex, but still head-oriented yield formation and (**ii**) multi-rooted tree structures to achieve a greater flexibility in choosing key dependency relations. While doing this, I aim at obtaining a theory that is perhaps not as broad in coverage as **GB** theory, but has the potential to be elaborated to the level of detail of a modern syntactic framework such as **HPSG** [PS94], and concentrate in particular on the ability to parametrize very general properties of word order.

While this chapter makes use of **HG** to provide a principled background for **LMG**, it also attacks some problems of head grammar. The full version of Pollard's head grammar lacks a claim of tractability, due to the presence of unboundedly deeply embedded SLASH features. Furthermore, while head grammar is capable of localizing crossed dependencies in Dutch, it does not try to localize other long distance phenomena (such as nominal gap–filler dependencies).

Hence another extension is proposed, to achieve complete localization of relative clause dependencies: an optional relaxation of the notion of hierarchy, motivated by the result proved in chapter **8** that if an **MHG** is capable of describing Dutch relative clauses, it will necessarily describe these with a structure that looks "turned inside out" with respect to traditional analyses. This attaching of the *wh*-relative clause not at its "top" but at the *wh*-word it quantifies over is further motivated by the structures we saw in examples of **XG** in chapter **6**, which also had this link explicitly represented in their graph-structured derivations. Both extensions to **HG** proposed remain suitable for an implementation based on the tractable $\mathcal{S}$-**LMG** formalism.

10.1 Basic setup for a HG of English

HEAD GRAMMAR (**HG**) is a principled grammar framework that is tightly related to **GPSG**, but makes use of *head wrapping operations* to enhance the device that generates surface strings. After Pollard's [Pol84], research on head grammar has focused on the simplified version **MHG** that appeared frequently in parts **I** and **II** of this thesis, and research on which is limited to formal language-theoretical results. Although **MHG** and **HG** are weakly equivalent, some **HG** analyses have no strong equivalent in **MHG**.

To get started, I will define **HG** in its original formulation by Pollard, but glossing over features that are not essential to this chapter, and adding some of my own. To be precise, I make the following restricting modifications:

▷ a simplified, strictly extensional semantic component

▷ a slightly different view on Pollard's CONTROL TYPES.

▷ a bounded feature space (no SLASH features)

▷ no ID/LP rules [GKPS85]

The additional features are defined separately and yield a version which I call FREE head grammar (**FHG**), that has slightly more powerful yield formation operations.[1] Every **HG** is an **FHG**. The key point of this chapter is to show how an unbounded feature space and the ID/LP rules can be replaced with a combination of more free yield formation, and later, with a more free attitude towards hierarchy and phrase containment.

The principles of free head grammar

A (free) head grammar is, formally, a simple **LMG** whose clauses satisfy a long list of properties. As in **GB**, some of these properties are UNIVERSAL PRINCIPLES, others are language dependent PARAMETERS.

10-1 principle: features. The set of nonterminals *N* of an **FHG** is a finite feature space whose elements are (partially) characterized by a list of feature specifications in the style of **GPSG** [GKPS85], such as [+AUX, NUM **PL**, PERS **3RD**]. Partial feature specifications are used only informally, and specify *sets* of nonterminals.[2]

*

The following two principles express BINARY BRANCHING, and how the surface clustering is limited to what one can build by associating with each node in a structural

[1] Such a free construction is also used in GERTJAN VAN NOORD's dissertation [vN93], but differs on a number of parameters, such as the following: the verb second operation in Van Noord's analysis is not order preserving (parameter **10-4**).

[2] Alternatively, from a more implementational point of view, only part of the feature space may be encoded into the nonterminals and the rest encoded into finite annotations as proposed in chapter **7**.

representation, in a systematic manner, a string divided into a LEFT CONTEXT, its HEAD and a RIGHT CONTEXT. Furthermore, the only phrase containment relation is head–dependent.

10-2 principle: tree principle. A sentence is associated with a tree structure (its DERIVATION) in which each node is either terminal, or the parent of two other nodes, one of which is the head, and one of which is the complement or dependent.

10-3 principle: surface clusters. Nonterminals in an **FHG** occur only 3-ary; all clauses must be simple (definition **3-26**), and of the form

$$A(t_1,\ h_1,\ t_2)\ \texttt{:-}\ H(l_1,\ h_1,\ r_1),\ D(l_2,\ h_2,\ r_2).$$

(head-complement clause) where l_i, h_i and r_i are variables, and t_1, t_2 are arbitrary terms over these variables; or of the form (lexical clause)

$$A(u, v, w).$$

where u, v and w are strings.[3] H is the HEAD DAUGHTER of A, and D is the DEPENDENT. Consequently, **FHG** derivation trees are right-branching.[4]

In the non-lexical case, the head daughter H has a feature RULE, whose value is the tuple (t_1, t_2), uniquely determines the applied **LMG** clause.

*

The following are important parameters to be "switched on and off" *ad lib.* to vary the strictness of the underlying clause system to levels between **CFG** and simple **LMG**.

10-4 parameter: order preserving. A rule is ORDER PRESERVING when if t_1, t_2 are as in principle **10-3**, then in the term $t_1 h_1 t_2$, the left-to-right order of l_1, h_1, r_1 and l_2, h_2, r_2 is preserved; that is, no occurrence of r_1 precedes one of h_1, no occurrence of h_1 precedes one of l_1, *&c.*

10-5 parameter: linearity and nonerasingness. The terminology of definition **3-26**, except simplicity, which is required by definition, can be required parametrically of **FHG**.

10-6 parameter: head-right adjacency. A rule is HEAD-RIGHT ADJACENT IN ITS HEAD if $t_2 = r_1 t_2'$. A rule is HEAD-RIGHT ADJACENT IN ITS DEPENDENT if either t_1 or t_2 contains the subterm $h_2 r_2$.

[3] This is a terminal production in the structural sense, but allows the clusters generated under a head to be more than a single terminal symbol. The reason for allowing u and w to be nonempty, *i.e.* allowing a lexical item to be split up, is to allow *e.g.* for Dutch verbs like *aanbieden*, whose prepositional prefix is separated from the verb under movement, to be described (u = *aan*; v = *bieden*). The lexical entry for *aanbieden* as *to offer something* has $w = \varepsilon$; the entry meaning *to offer to do something* will have $w = \texttt{te}$, equivalent to the English infinitive marker *to*. See (10.18) for an illustration.

[4] Note that right-branching in the structure poses no restrictions to the surface order.

*

10-7 definition. An **FHG** is a HEAD GRAMMAR (**HG**) if it is linear, non-erasing, order preserving and head-right adjacent in its head. A head grammar is MODIFIED (that is it corresponds to an **MHG**) if it is also head-right adjacent in its complement.

The following six clause types then remain for **HG**;[5] a modified **HG** will only use **HC**, **CH** and **HWL**.[6]

HC	$A(l_1,\ h_1,\ r_1 l_2 h_2 r_2)$	$:-\ H(l_1,h_1,r_1), D(l_2,h_2,r_2).$	*(head-complement)*
CWR	$A(l_1,\ h_1,\ l_2 h_2 r_2 r_1)$	$:-\ H(l_1,h_1,r_1), D(l_2,h_2,r_2).$	*(compl. wrapped right)*
HWL	$A(l_2 l_1,\ h_1,\ r_1 h_2 r_2)$	$:-\ H(l_1,h_1,r_1), D(l_2,h_2,r_2).$	*(head wrapped left)*
HWR	$A(l_2 h_2 l_1,\ h_1,\ r_1 r_2)$	$:-\ H(l_1,h_1,r_1), D(l_2,h_2,r_2).$	*(head wrapped right)*
CWL	$A(l_1 l_2 h_2 r_2,\ h_1,\ r_1)$	$:-\ H(l_1,h_1,r_1), D(l_2,h_2,r_2).$	*(compl. wrapped left)*
CH	$A(l_2 h_2 r_2 l_1,\ h_1,\ r_1)$	$:-\ H(l_1,h_1,r_1), D(l_2,h_2,r_2).$	*(complement-head)*

10-8 principle: head feature convention. The feature specifiers are divided into two groups: HEAD FEATURES and NON-HEAD FEATURES. As a rule, all features, except CON, RULE and COMP are head features. Parametrically, a limited set of additional features may be allowed as non-head features.

10-9 principle: selection. A finite domain of CONTROL TYPES is constructed on the following principles:

1. A type has an ORDER
2. There is a fixed finite set of order 0 types, including at least D (determiner), N (noun) and V (verb).
3. A type of order $i+1$ is a tuple (t_1, t_2) where t_1 is a type of order i and t_2 is a type of order less than or equal to i.
4. The order of a type is not to exceed 3.

The categories A, H and D from definition **10-3** satisfy the following: there are control types t_1 and t_2 such that one of the following schemes is satisfied:

$$\begin{aligned} H &= [\text{CON } (t_1,t_2), \alpha, \beta, \text{COMP } \gamma] \\ A &= [\text{CON } t_2, \alpha, \delta] \\ D &= [\text{CON } t_1, \gamma] \end{aligned} \tag{10.1}$$

$$\begin{aligned} H &= [\text{CON } t, \alpha, \beta, \text{COMP } \gamma] \\ A &= [\text{CON } t, \alpha, \delta, \text{COMP } \gamma] \\ D &= [\text{CON } (t,t), +\text{MODIFIER}, \eta] \end{aligned} \tag{10.2}$$

[5] Pollard's original formulation speaks of twelve forms, but I have chosen to consider only those rules where the first item on the RHS is the head daughter.

[6] The identifiers on the left are abbreviations for the tuples (t_1, t_2); *e.g.* **CWR** $= (l_1,\ l_2 h_2 r_2 r_1)$.

where $\alpha, \beta, \gamma, \delta, \eta$ are feature specifications, and β and δ contain only non-head features.

*

Although I will not be concerned with semantics, a very strict correspondence between semantics and surface order generalizations plays a rôle in the control type design of head grammar, so a brief definition of semantic interpretation is in place.[7]

10-10 principle: interpretation. Each control type *t* is associated with a SEMANTIC TYPE $[\![t]\!]$ where

1. A type is constructed as usual from the basic types *e* and *t*, and the function operator →. Type expressions are right-associative, that is when we write $T_1 \rightarrow T_2 \rightarrow T_3$, we mean $T_1 \rightarrow (T_2 \rightarrow T_3)$.
2. If $[\![t_1]\!] = T_1$ and $[\![t_2]\!] = T_2$, then $[\![(t_1, t_2)]\!] = T_1 \rightarrow T_2$.
3. The semantic type for a basic category does not need to be a basic type.

Each lexical clause specifies an interpretation in the form of a λ-expression extended with boolean operators $\wedge$, $\vee$, *&c.* as usual in an (extensional) **PTQ** setting [Jan86]. Parametrically, these expressions can be required to be extensional.

For a head-complement clause, the semantic interpretation of parent node A is obtained by applying the meaning of the head H to that of the complement C.

*

This concludes the core principles of head grammar as I will present it here. The list in *figure 10.1* summarizes the control types I will be concerned with, and their associated semantic types. What is called N^0 elsewhere in this thesis is represented here by the type D^0. There is no distinction between the levels V^0 and C. A raising verb, unlike in other parts of this thesis, selects for an intransitive phrase to build a transitive phrase.

Parameters for English

A simple fragment of English is now specified in a few more statements, saying which yield formation operators are to be used. In principle, complements in English follow their heads, but the heading phrase may contain some 'right context'; *e.g.* the transitive phrase *convince to leave* should wrap in its object to the right of its head *convince*, but left of *to leave*.

An exception to this default operation is the subject, which precedes the intransitive phrase that selects for it, except when a sentence shows subject-auxiliary inversion. So

[7] Definition **10-10** is stricter than Pollard's interpretation principle, which distinguishes two semantics operations in addition to functional application: *functional composition* and *equi*. Pollard makes use of these different classes to capture word order generalizations, but as he points out, there are anomalous cases. These get worse in a language like Dutch, therefore I chose to encode these operations lexically. Since there will be practically no semantics in this chapter, this definition serves merely as an indication.

control type	name and semantic type
$\mathrm{N^C = N}$	common noun $e \to t$
$\mathrm{D^0 = D}$	determiner phrase (noun phrase) $(e \to t) \to t$
$\mathrm{D^N = (N^C, D^0)}$	determiner $(e \to t) \to (e \to t) \to t$
$\mathrm{V^0 = V}$	saturated verbal clause t
$\mathrm{V^I = (D^0, V^0)}$	intransitive verb or phrase $((e \to t) \to t) \to t$
$\mathrm{V^T = (D^0, V^I)}$	transitive verb or phrase $((e \to t) \to t) \to ((e \to t) \to t) \to t$
$\mathrm{V^A = (V^I, V^I)}$	auxiliary or modal verb (subject to subject raising) $(((e \to t) \to t) \to t) \to ((e \to t) \to t) \to t$
$\mathrm{V^R = (V^I, V^T)}$	raising verb (subject to object raising) $(((e \to t) \to t) \to t) \to$ $((e \to t) \to t) \to ((e \to t) \to t) \to t$

Figure 10.1: Basic control types

I conclude with the following, extremely simplified, parameter for English, and refer to [Pol84] for a more sophisticated analysis of English verb constructions. A further elaboration concerned with relative clauses is included in this chapter.

10-11 parameter: English yield formation. Subjects in English precede their heading phrase. All other complements in English follow their heads, but precede the right context of the heading phrase.

(10.3) $N \subseteq [\textsc{rule}\ (\mathbf{CH} \cup \mathbf{CWR})]$

(10.4) $N \cap [\textsc{con}\ \mathrm{V^I},\ -\textsc{inv}] = N \cap [\textsc{rule}\ \mathbf{CH}].$

An example derivation tree is shown in *figure 10.2*; note that instead of displaying tuples, a more lean notation is used that separates the strings generated at each node into three clusters using dots $(\cdot)$. This is further reflected in the following: whereas for the formal derivational interpretation of **LMG** I would have written

(10.5) $\vdash \mathrm{V^0}(\texttt{Frank}, \texttt{saw}, \texttt{Julia drink coffee}).$

I prefer here to think of the **HG** triples as strings with possible insertion points left and right of their head. The result is a relation $\vdash$ more like its context-free counterpart.

(10.6) $\mathrm{V^0} \vdash \texttt{Frank} \cdot \texttt{saw} \cdot \texttt{Julia drink coffee}$

10.2 Sentential structure of Dutch

The price of the psycho-linguistic motivation of head grammar, even in the free form defined here, is that it substantially restricts the effect of dividing the yield of a nonterminal into clusters with respect to what is allowed in arbitrary **LMG**. So it is now left to show that what can be described using this limited capacity is sufficient. As a first step, I will now use head-driven clustering to generate the surface order of basic Dutch sentential structure. Let the examples in this thesis suffice to show that although [Pol84] contains an appendix with a small, isolated fragment of Dutch subordinate clauses, it requires much more to make this adequacy *really* plausible.

This section will treat a bit more than the previous chapter on **GB**, but for the time I will disregard (**i**) adjunction (such as adverbial modification) and (**ii**) long surface distance nominal dependencies (topicalization, *wh*-dependencies and relative clauses); these will get attention in the last section of this chapter. To get started, let's reformulate the minimal English fragment of the previous section, but for subordinate clauses only.

Basic raising and extraposition order

In Dutch, a verb taking an embedded verb complement is wrapped in left of the complement's head. In other verbal cases, all complements precede their heads (canonical order is SOV). Determiner phrases correspond to the English order.[8]

(10.7) $$N \subseteq [\textsc{rule}\ (\mathbf{CH} \cup \mathbf{CWR} \cup \mathbf{HWL})]$$

[8] The **CWR** rule is used for determiners because in section **10.4**, a relative clause is attached to the determiner and moved into its right context.

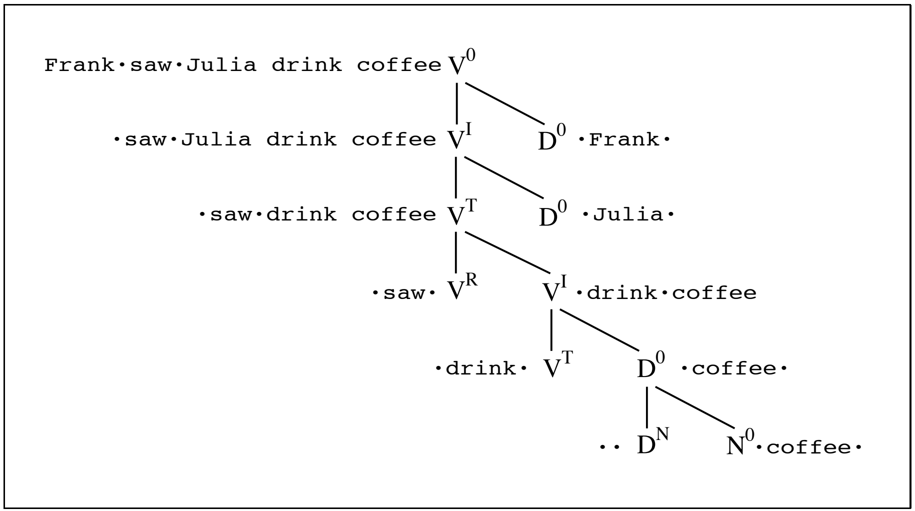


Figure 10.2: English **FHG** analysis.

(10.8) $[\textsc{con}\ (\mathrm{V}^{\mathrm{A}} \cup \mathrm{V}^{\mathrm{R}}), \textsc{stype}\ \mathbf{SUB}, +\textsc{raising}] \subseteq [\textsc{rule}\ \mathbf{HWL}]$

(10.9) $[\textsc{con}\ (\mathrm{V}^{\mathrm{I}} \cup \mathrm{V}^{\mathrm{T}}), \textsc{stype}\ \mathbf{SUB}] \subseteq [\textsc{rule}\ \mathbf{CH}]$

(10.10) $[\textsc{con}\ \mathrm{D}^{\mathrm{N}}] \subseteq [\textsc{rule}\ \mathbf{CWR}]$

(10.11) $[\textsc{con}\ (\mathrm{V}^{\mathrm{I}} \cup \mathrm{V}^{\mathrm{T}} \cup \mathrm{V}^{\mathrm{A}} \cup \mathrm{V}^{\mathrm{R}})] \subseteq [\textsc{comp stype}\ \mathbf{SUB}]$

This set describes basic cross-serial verb order without the extraposition verbs from the previous chapter, in a way closely resembling that of the **MHG** in **3.2**, except that the head is now a separate cluster. The following are examples of underlying **LMG** clauses licensed by parameters (10.7–10.11).

(10.12) $\mathrm{V}^{\mathrm{R}}[+\textsc{raising}, \textsc{pers}\ \mathbf{3RD}, \textsc{num}\ \mathbf{SG}](\varepsilon,\ \texttt{zag},\ \varepsilon).$

(10.13) $\mathrm{V}^{\mathrm{T}}[\textsc{stype}\ \mathbf{SUB}](l_2 l_1,\ h_1,\ r_1 h_2 r_2)\ \texttt{:-}\ \mathrm{V}^{\mathrm{R}}(l_1,\ h_1,\ r_1),\ \mathrm{V}^{\mathrm{I}}(l_2,\ h_2,\ r_2).$

The following is an example of a derivation using these clauses.

(10.14) $\mathrm{V}^{\mathrm{I}}[\textsc{stype}\ \mathbf{SUB}] \vdash\ \cdot\ \texttt{zwemmen}\ \cdot$

(10.15) $\mathrm{V}^{\mathrm{T}}[\textsc{stype}\ \mathbf{SUB}] \vdash\ \cdot\ \texttt{helpen} \cdot \texttt{zwemmen}$ by (10.8)

(10.16) $\mathrm{V}^{\mathrm{I}}[\textsc{stype}\ \mathbf{SUB}] \vdash \texttt{Julia} \cdot \texttt{helpen} \cdot \texttt{zwemmen}$ by (10.9)

(10.17) $\mathrm{V}^{\mathrm{T}}[\textsc{stype}\ \mathbf{SUB}] \vdash \texttt{Julia} \cdot \texttt{zag} \cdot \texttt{helpen zwemmen}$ by (10.8)
"saw · help Julia swim"

Now I proceed to extraposition as discussed in **9.4**. Two features, RAISING and EXTR, indicate whether a verb allows for the raising or extaposition constructions, or for both. The strict extraposition verb *verbieden* is an example of a "complex" lexical entry, *i.e.* one that does not just have a single word in the head cluster, but has, in its right context, the infinitive marker *te*.

(10.18) $\mathrm{V}^{\mathrm{R}}[+\textsc{extr}, -\textsc{raising}, \textsc{pers}\ \mathbf{3RD}, \textsc{num}\ \mathbf{SG}](\varepsilon,\ \texttt{verbood},\ \texttt{te}).$
"disallowed to"

Applying the raising rule (10.13) would yield the incorrect sentence (10.19a); the correct version is (10.19b). To model this, an **FHG** rule **EXTR**, that is not one of the six wrapping and concatenation rules, is convenient.[9]

(10.19) a. $*$. . . dat Frank $[_{\mathrm{V}^{\mathrm{I}}}$ Fred Julia · verbood · te helpen zwemmen]
b. . . . dat Frank $[_{\mathrm{V}^{\mathrm{I}}}$ Fred · verbood · Julia te helpen zwemmen]
". . . that Frank did not allow Fred to help Julia swim"

EXTR $A(l_1,\ h_1,\ l_2 r_1 h_2 r_2)\ \texttt{:-}\ H(l_1,\ h_1,\ r_1),\ C(l_2,\ h_2,\ r_2).$ *(extraposition)*

(10.20) $[\textsc{con}\ (\mathrm{V}^{\mathrm{A}} \cup \mathrm{V}^{\mathrm{R}}), \textsc{stype}\ \mathbf{SUB}, +\textsc{extr}] \subseteq [\textsc{rule}\ \mathbf{EXTR}]$

[9] Convenient, but not necessary, for the infinitival marker *te* can also be modelled as a verbal category, which is done by Pollard for the analysis of English; it does seem to make less sense for west-Germanic languages; personally I find it stylistically infelicitous to assign categories to small things such as markers, especially there where surface clustering provides a more "lean" model, in which a verb like *teach* gets a lexical entry in which it reads *teach · to*.

However, this rule is reasonably well-behaved, since it is order preserving and head-right adjacent in its complement.

A place where at first sight, order preservingness seems to be difficult to maintain, is declarative and interrogative sentential forms in Dutch. To model verb second, there are two options; the easiest, but least principled is to forget order-preservingness, and add rules that form a V^{0} from a V^{I} by moving the head verb to initial or second position (this is what is done in [vN93]).

An alternative option that respects order preservingness however seems more defendable on a principled basis, and is supported by the head feature convention, which claims that the STYPE feature is present on all the verbal projections anyway. The main verb is prefixed already at the rule that constructs the V^{I} from the higher $V^{T/A/R}$. Not only are the rules used order-preserving, but they happen to even fall within the standard six-pack of strict **HG** rules.

(10.21) $[\mathrm{CON}\ (V^{T} \cup V^{A} \cup V^{R}), \mathrm{STYPE}\ (\mathbf{DECL} \cup \mathbf{QUES})] \subseteq [\mathrm{RULE}\ \mathbf{HC}]$

(10.22) $[\mathrm{CON}\ V^{I}, \mathrm{STYPE}\ \mathbf{DECL}] \subseteq [\mathrm{RULE}\ \mathbf{CH}]$

(10.23) $[\mathrm{CON}\ V^{I}, \mathrm{STYPE}\ \mathbf{QUES}] \subseteq [\mathrm{RULE}\ \mathbf{CWR}]$

The following simple examples are sufficient to check how these rules work.

(10.24) $V^{T}[\mathrm{PERS}\ \mathbf{3RD}, \mathrm{NUM}\ \mathbf{SG}](\varepsilon, \texttt{dronk}, \varepsilon)$.
"drank"

(10.25) $V^{I}[\mathrm{STYPE}\ \mathbf{SUB}] \vdash \texttt{koffie} \cdot \texttt{dronk} \cdot$

(10.26) $V^{0}[\mathrm{STYPE}\ \mathbf{SUB}] \vdash \texttt{(...dat) Frank koffie} \cdot \texttt{dronk} \cdot$

(10.27) $V^{I}[\mathrm{STYPE}\ \mathbf{DECL/QUES}] \vdash \cdot\, \texttt{dronk} \cdot \texttt{koffie}$

(10.28) $V^{0}[\mathrm{STYPE}\ \mathbf{QUES}] \vdash \cdot\, \texttt{dronk} \cdot \texttt{Frank koffie (?)}$

(10.29) $V^{0}[\mathrm{STYPE}\ \mathbf{DECL}] \vdash \texttt{Frank} \cdot \texttt{dronk} \cdot \texttt{koffie}$

This concludes a fairly indisputable basic analysis. Now let's look at how the analysis may be extended to a further level of sophistication. This is no longer straightforward.

Extended coverage and problems

The phenomena of partial extraposition and inverted verb cluster ordering often pose problems to a syntactical analysis because they force "flexible" application of the principles. The same happens in the current **FHG** analysis. Recall the examples of partial extraposition from the previous chapter:

(10.30) a. . . . dat Frank probeerde Julia koffie te geven
b. . . . dat Frank Julia probeerde koffie te geven
c. . . . dat Frank Julia koffie probeerde te geven
". . . that Frank tried to give Julia coffee"

Although (10.30b) already raises some question marks for some Dutch speakers, it is generally accepted as correct. The problem is that *proberen (te)* seems logically to be of type V^{A} that takes an intransitive phrase as its complement, to yield a new intransitive phrase. But such a complement V^{I} would be split up for example as `Julia koffie` · `laten` · `drinken`; so there is no way to let the verb *probeert* end up between *Julia* and *koffie*. The only solution that seems to work is to give *proberen* a second control type $(\mathrm{V}^{\mathrm{T}}, \mathrm{V}^{\mathrm{T}})$. But there may be Dutch speakers who approve of higher "types" for proberen: in sentence (10.31), *hun voer te leren uitdelen* would already be ditransitive.

(10.31) ??. . . dat Frank Julia de dieren probeerde hun voer te leren uitdelen
". . . that Frank tried to teach Julia to give the animals their food"

As to inversion of the order in verbal clusters, this seems to be unproblematic in the absence of ADVERBIAL MODIFICATION, which is postponed to section **10.4**.

For verbs that allow inversion, one can simply replace the rule **HWL** that wraps in the head verb to the *left* of the embedded verb, to **HWR** that wraps it in to the right; *cf.*

(10.32) a. Julia koffie · wilde · zien drinken
b. Julia koffie drinken · wilde · zien
c. Julia koffie drinken zien · wilde ·
"wanted to see Mary drink coffee"

In the presence of modification, these phenomena lead to problems that are not straightforward to solve; in the following sentences, the modifier between square brackets must intervene a string that is fully contained in the left cluster of the modified constituent, and can hence according to the principles not be interrupted. Sentence (10.33b.) provides evidence that a type raising like proposed for partial extraposition verbs will not solve the issue here either. A restriction to what types of verb can be sequenced "in the wrong order" is infeasible too, *cf.* (10.33c.). A possible solution, but also with its problems, may be to allow non-lexical strings to be produced in the head cluster.

(10.33) a. . . . dat Patijn de burgers [voor 16 februari] opgeroepen · wilde · hebben te stemmen
". . . that Patijn wanted to have [called the citizens to vote] before the 16th of February"

b. . . . dat Patijn het comité de burgers [niet] verleiden · liet · tegen te stemmen
". . . that Patijn did not let the committee seduce the citizens into voting against"

c. . . . dat Frank Fred Julia [niet] zwemmen helpen · zag ·
". . . that Frank did not see Fred help Mary swim"

10.3 Sentential structure of Latin

LATIN is known as a language with a highly free word order; in **GB** theory, one often talks of a NON-CONFIGURATIONAL language, because attempts to cast a language like Latin into an X' framework seem to fail, and one then reverts to a theory that does not posit an X' schema. But with strict surface order, one then also throws out **GB**'s notion of hierarchy. Other arguments for the freeness of Latin word order proceed along similar lines—incapacity of currently known descriptive mechanisms.

Nonetheless, even a language like Latin has a number of indisputable, very global constraints on word order, such as the property that words of the main clause cannot be inserted between those of a subordinate clause; furthermore, word order has been shown to depend strongly on pragmatic factors. In a completely non-configurational account of Latin, it is difficult to see how such constraints and factors should be expressed. Finally, a non-configurational analysis does not seem to lend itself well for automatic processing. A substantial overview, and arguments that Latin is *not* a free word order language, can be found in PINKSTER [Pin90], chapter 9.

This section will show that in the free form of head grammar of this chapter, large amounts of Latin surface order can be described while maintaining a deep structure that is equivalent to that of languages like English and Dutch. Because **FHG** extends the projective capacities of an X' schema only very mildly, this seems to indicate that there is an unexpectedly large benefit in relaxing the constraint of projectivity. I will look, very globally, at what relationships occur if one assigns headed strings to the nodes in a Latin dependency structure. This has the following implications:

1. For a node in the dependency structure, this means that the words in the dependency cluster it dominates must together consist of a head and two *continuous* substrings of the input. Since the dependency structure of Latin is fairly indisputable,[10] this property can be empirically checked.

2. In an analysis such as in **FHG**, where a head takes its dependents consecutively like in X' syntax, it must be studied in which orders a head can take its complements and optional dependents.

To facilitate this analysis, I will take a step back and leave out the formal designations in terms of features, and merely look at how dependency and surface cluster triples generated by head operations interact.

Tendency versus rule

In contrast to modern West-Germanic languages, Latin has very few strict rules, but there are some. One of these is the restriction on subordinate clauses mentioned above; this seems to be a strictly *context-free* constraint. Another is that in a prepositional

[10] Perhaps with the exception of the NP/DP distinction.

phrase, when spelled out in a sentence, at least one element of the PP must immediately follow the preposition. Here are some data.

(10.34) a. magna cum laude
b. cum magna laude
c. ∗ magna laude cum
d. has miseras, . . . , ad inferras
e. ??per multum arenosos rivos

The most general constraint, allowing the poetic example (10.34d), could be stated as

(10.35) The rule combining a preposition with its complement noun phrase, either puts the head noun in the the right context, or insists otherwise that the right context is nonempty. Every rule attaching a PP to a larger phrase is head-right adjacent in its complement (the PP).

This is the first illustration of an important difference between Latin and modern languages: the examples in which the head noun appears *before* the preposition, are judged more or less 'difficult' according to the status of the word immediately following the preposition. Whereas modern languages have evolved to strictly forbid difficult cases, Latin is of a more lenient nature. An underlying structural mechanism would preferably be one that generates all possible structures, including very poetic ones, but allows for this level of difficulty to be expressed formally in terms of which head-driven operations have been applied.

Other similar examples, discussed in more detail in [Pin90], where Latin has a strong *tendency* for a certain order, while that choice is compulsory in modern languages, are (**i**) the position of causal subordinate clauses (tendency to follow the verb) versus causal adverbial modifiers (free); (**ii**) adverbs have a tendency to appear immediately next to the verb they modify (**iii**) some adjectives whose semantic contents is determiner-like have different word order properties.

This concludes the concrete discussion of these tendencies themselves; henceforth I will be interested solely in the most general possible word orders; these can obviously not be specified in the way they were for English and Dutch: giving a single rule label for certain classes of words. Rather, more general *properties* of these rules, such as order preservingness and head-right adjacency need to be varied over different types of construction.[11] In this way, one gets a picture of the properties of one of the most free forms of word order, and thus one gets close to investigating universal minimal principles of human surface generation.

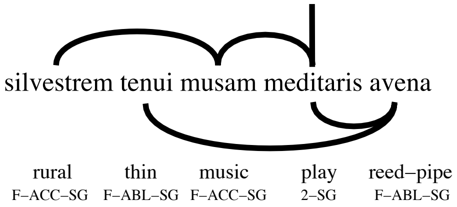
A brief empirical analysis

As before, the analysis I give here will concentrate on verbal complementation. I will give a number of examples, (10.36) taken from [Mel88] and the others from

[11] It is not unthinkable that a deeper analysis than given here will conclude that this can, due to the graduality of good and bad word order, only be done in a stochastic sense.

[Pin90], which are all used as examples "of the worst[12] kind", and then look at how the full verb phrases in the sentences can be obtained by successively adding the SATELLITE PHRASES, that is either complements or non-obligatory dependents, using the operations allowed in an **FHG** clause. It will turn out that a degree of freedom is required in the order in which satellites are bound to the head phrase.

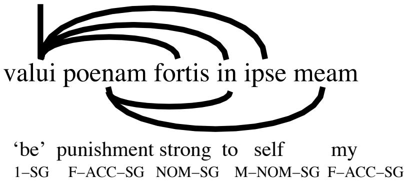
(10.36)


silvestrem tenui musam meditaris avena
rural thin music play reed–pipe
F–ACC–SG F–ABL–SG F–ACC–SG 2–SG F–ABL–SG

''...(Tityrus, you...) are playing rural tunes on a thin reed–pipe''
(Vergil, **??**) [Mel88]

Sentence (10.36) can be constructed either by first wrapping `tenui · avena ·` around *meditaris*, and then wrapping in `silvestrem · musam ·` . These operations would be order preserving. If however preference is given to taking obligatory complements first (assuming that *meditaris* selects for an object), a non-order-preserving operation is needed: first, `silvestrem · musam ·` is wrapped around *meditaris*, to form `silvestrem · meditaris · musam`, and then while wrapping in `tenui · avena ·` , the right context *musam* is moved leftward over *meditaris*.

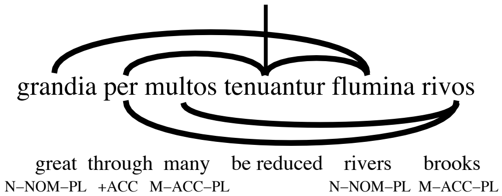
(10.37)


valui poenam fortis in ipse meam
'be' punishment strong to self my
1–SG F–ACC–SG NOM–SG M–NOM–SG F–ACC–SG

''I have been brave, to my own disadvantage'' (Ov. *Am.* 1.7.26)

Sentence (10.37) shows the opposite effect. If the PP `poenam · in · meam` is wrapped in first, it is merged into *two* disjoint clusters, while it is broken up into three clusters in the sentence, *i.e.*, there is no successful derivation. If *ipse* and *fortis* are wrapped in first to form `ipse · valui · fortis`, there is a successful derivation, but one that is not order preserving.

(10.38)

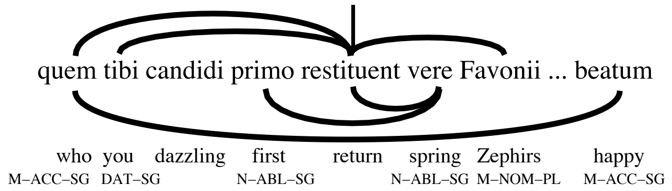

grandia per multos tenuantur flumina rivos
great through many be reduced rivers brooks
N–NOM–PL +ACC M–ACC–PL N–NOM–PL M–ACC–PL

''Great streams are chanelled into many brooks'' (Ov. *Rem.* 445)

[12] It should be born in mind, however, that it is also Pinkster's objective to show that Latin word order is *not* free.

Example (10.38) is head-constructible only if *multos* and *rivos* in the PP are split; *i.e.* the PP is derived as `multos` · `per` · `rivos` or `rivos` · `per` · `multos`. The rule binding the PP to the verb phrase will insist on putting either left or right context of the preposition immediately right of the preposition. This sentence is another example of a construction that is inherently non-order-preserving.

(10.39)


quem tibi candidi primo restituent vere Favonii ... beatum
who you dazzling first return spring Zephirs happy
M–ACC–SG DAT–SG N–ABL–SG N–ABL–SG M–NOM–PL M–ACC–SG

''whom the dazzling west winds will give back to you in a happy state... ... at the beginning of the spring'' (Hor. *C.* 3.7.1–3)

The very symmetrical, multiply embedding sentence (10.37) gives another hint: the components of the subject phrase `candidi` · `Favonii` · appears in the middle of the left and right context of the head verb *restituent*. This implies that while the dependents can be attached in almost any order, the subject can *not* be attached *last*, as is conventionally done in analyses of modern configurational languages.

Conclusions

The type of analysis given here, *i.e.* a look at (the most excessive cases of) what is *possible* in word order formation, is hardly ever found in the literature. The majority of other studies of word order in Latin seems to investigate what the prosodial, rhetorical, *&c*, effect of order variation is, without having a notion of how to *talk* about word or phrase order, if phrases can be highly discontinuous.[13] One of the aims of the study in this chapter is to provide a concise description of the general situation in such a way that such motivations for certain word order preferences can be formulated at a higher level of formal adequacy.

The analysis I gave is insubstantial, but it does seem to indicate that free head-driven surface generation is capable of generating a large amount of Latin word orders, and that there is a trade-off between restricting the order in which complements and optional dependents are bound to a head, and the strictness of the principles specifying which yield formation operations are allowed. It is on purpose that I have looked at a small selection of very free word orders—more prosaic Latin sentences will fit into the **FHG** operations much more easily, and can lead one to pursue preliminary conclusions that obstruct an investigation of the more difficult cases. Of course, investigating simpler sentences based on the minimal framework thus obtained is nonetheless clearly the next step to take.

A preliminary conclusion is that there is a preference for attaching adverbial modifiers after the complements, *including the subject.* This is supported both by the

[13] A similar remark is made in [Pin90].

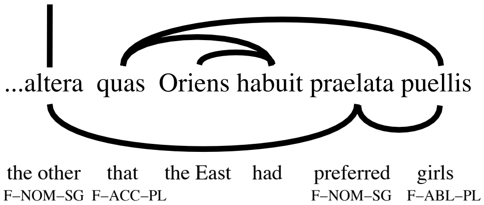
examples and the idea that adverbial phrases (and likewise for optional dependents such as the ablative phrases in the examples) have a tendency to appear close to the verb they modify. The only exception is example (10.36) in which final attachment of the ablative phrase *tenui avena* is in conflict with order preservingness. This is not surprising, since the occurrence of *tenui* so early in the phrase seems to mark it with an exceptional amount of stress. Another idea that needs further investigation is whether head-right adjacency w.r.t. the complement, *i.e.* the property of complement daughters to be split up only into two strings, is perhaps a property of higher weight than order preservingness.

Other analyses of Latin [Ros67] complement order have been made in terms of the *scrambling* process that has been argued to apply to the German verb phrase. The remarks in example **8-11** (page 171) hold equally well for Latin: it is worthwhile to investigate the possibility that the freedom of word order is restricted in such a way that mildly configurational formalisms such as loosely interpreted **XG** and tuple grammars can produce all the allowed orders (while giving strongly adequate analyses). This does seem to be supported by the fact that tuple grammars can *not* describe unrestricted scrambling, but I have not found any examples of Latin constructions that can't be dealt with using the 3-tuple based operations of **FHG**!

10.4 Localization of relative clause dependencies

The Latin sentence (10.40) that was already mentioned in the introduction introduces no difficult bindings. The phrase `quas Oriens habuit · puellis ·` is wrapped around *praelata*, the resulting phrase right-concatenated to *altera*.

(10.40)



...altera quas Oriens habuit praelata puellis
the other that the East had preferred girls
F–NOM–SG F–ACC–PL F–NOM–SG F–ABL–PL

''...the other a frequent winner of the Miss Middle–East contest''
(Ov. *Met.* IV 56.)

It does, as said in the introduction, introduce a so-called DOUBLE DEPENDENCY. The word *quas* depends depends on both *puellis* and *habuit*. This observation, together with the strong generative capacity construction for **MHG** in section **8.2**, naturally leads to the idea of implementing a tree structure in **HG** that allows for multiple heads.

While Latin provides good examples because of its rich morphology, I will concentrate here on English examples, because the distinction noun/determiner in Latin is somewhat cumbersome. The English analysis will not have multi-headedness, but a more general form of MULTIPLE DOMINANCE; the determiner in the matrix clause will be the head of a D^0, and the complement of a relative pronoun, at the same time. Consider the sentence

(10.41) Frank saw the man (that/whom) Julia loved

The accusative morphology of *whom* shows that the object dependency is present here too, and since case is a head feature, I propose an analysis in which the relative *that*/*whom* is the head of the object in the embedded clause. For this purpose, I introduce a new control type D^D for relative pronouns (without a semantic type for the time being). The proposed analysis of sentence (10.41) is shown in *figure 10.3a*. As usual, vertical lines denote head relations; while drawing the trees in this section it was unavoidable to put some complement arrows left of the head.

The grammatical principles put forward in section **10.1** need to be changed to allow for these types of analysis; I will not go into formal details here, since the proposal is of a highly tentative nature; rather will I attempt to sketch why it is feasible, and what its benefits could be.

Of central importance is the following relaxed notion of how dependency relations correspond to derivation trees.

10-12 principle: revised tree principle. The underlying structure of a sentence is always a tree; however, the tree need not be binarily branching, a node may be a dependent of a daughter node, or the head of its parent.

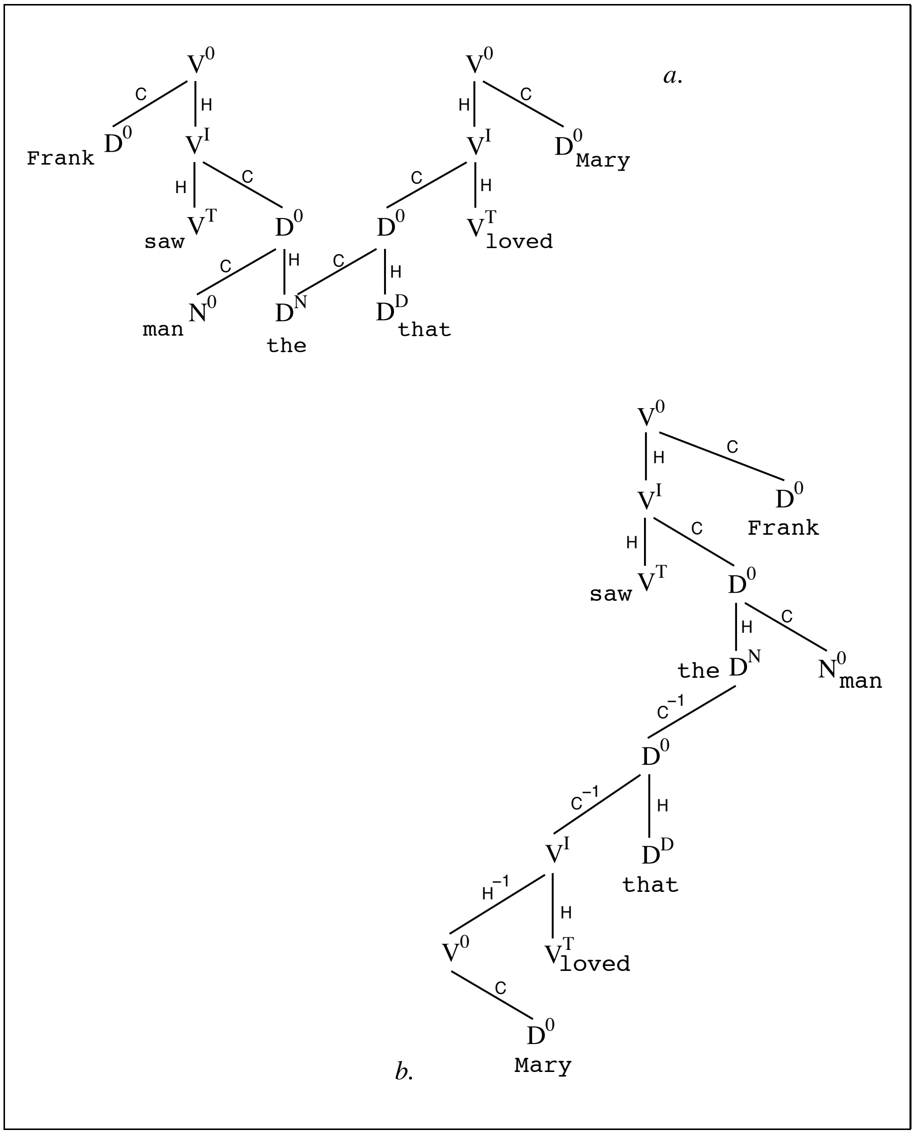


Figure 10.3: Multi-dominance relative clause analysis and the underlying **LMG** structure.

*

The underlying tree structure for sentence (10.41) is shown in part *b* of *figure 10.3*. There is an analogy between these *intuitive* and *underlying* structures and the *derived* and *derivation* structures of a tree adjoining grammar—see the **Epilogue**, section **E.2**.

In such a version of **FHG**, where trees can be "turned upside down", an additional feature PARENT is needed which selects the parent node. In the standard case, one has [PARENT **matrix**]; the two additional cases are [PARENT **head**] and [PARENT **complement**].

The yield formation principle **10-3** is rephrased to differentiate between the different values of the PARENT feature; it remains the same for [PARENT **matrix**]; nonterminals that have [PARENT **head/complement**] have 4 surface clusters as opposed to 3, because they act as the *context* to their parent node, and thus the parent node needs to be wrapped into such a node, instead of the other way around.

10-13 principle: revised yield formation. Let order preservingness (**10-4**) and head-right adjacency in complement be given.

Let $A \subseteq$ [PARENT p, RULE (t_1, t_2)]. Then

▹ If $p =$ **matrix**, then the underlying grammar contains, as in **10-3**, the clause

$$A(t_1,\ h_1,\ t_2)\ \ \texttt{:-}\ \ H(l_1,\ h_1,\ r_1),\ C(l_2,\ h_2,\ r_2).$$

▹ If $p =$ **head** and $t_1 = s_1 l_1 s_2$; $t_2 = s_3 r_1 s_4$,

$$A(p_1 s_1,\ s_2 p_2,\ p_3 s_3,\ s_4 p_4)\ \ \texttt{:-}\ \ M(p_1,\ p_2,\ p_3,\ p_4),\ C(l_2,\ h_2,\ r_2).$$

where A is the head of M; or in case that A is not a head or dependent, but a TERMINAL ROOT:

$$A(s_1,\ s_2,\ s_3,\ s_4)\ \ \texttt{:-}\ \ C(l_2,\ h_2,\ r_2).$$

▹ If $p =$ **complement**, the rule applied is undefined (depends on parameters such as head-right adjacency).

The situation [PARENT **complement**] is highly complex, and intentionally left open; in the case of a relative clause, features will mark special nature of the pronoun *that*, which causes the complement (*noun*) to be wrapped in between the determiner (*the*) and the relative clause with the complement relative pronoun *that* on its left.

The semantic interpretation principle is modified accordingly; the interpretation of a phrase with [PARENT **head**] for example is a lambda expression that abstracts over an expression of the type of the missing head.

*

Thus, relative clause dependencies can be localized, banning completely the notion of structural long-distance dependency. A similar mechanism can be used for adjunction:

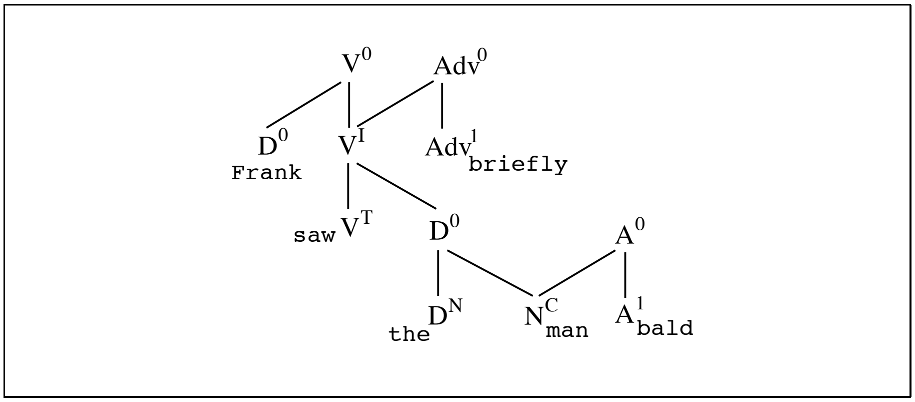


Figure 10.4: Adjunction.

an adjunct is a second head to the phrase it is adjoined to; this situation is sketched in *figure 10.4*. This would eliminate optional dependents, simplifying the feature system and eliminating alternative (10.2) of the selection principle. There is, however, a difference between the cases of relative clauses and adjunction. A relative clause analysis introduces a situation in which a node is dominated by two nodes, whereas adjunction can occur unboundedly many times. This makes it hard to guarantee that the underlying **LMG** remains finite. I expect that such a model of adjunction can overcome some of the problems in describing the difficult Dutch sentences at the end of section **10.2**.

Scope readings

The tree inversion paradigm is good for more than modelling adjunction and relative clause attachment. *Figure 10.5* shows two possible **LMG** backbones for the **FHG** analysis of sentence (10.42), which by English native speakers seems to be given two scope readings: one where one doctor visited every patient, the other where there can be more doctors, but every patient was visited by one doctor.[14]

(10.42) A doctor will interview every patient

The different scope readings are obtained from the two different structural analyses by use of the following rule: the semantic interpretation of a node is obtained bottom-up in the underlying structure, and at each individual node, a normal form is computed w.r.t. the familiar α and β reduction, and the following reduction rule:

Lift∃ $\exists x.\ (\cdots P\lambda y.\ \phi \cdots) \rightsquigarrow P\lambda y.\ \exists x.\ (\cdots \phi \cdots)$ when P is free

I.e., an existential quantification over a term containing a predicate applied to a lambda expression, the predicate and lambda expression are lifted over the existential quantor.

[14] For ease of presentation, *will interview* has been contracted to a single V^T in the figure; modelling auxiliaries properly is not a relevant issue here and requires event semantics.

The interpretation for the D^0 is derived in the traditional extensional fashion as follows:

(10.43) $\llbracket \mathrm{N}^0(\texttt{patient})\rrbracket \leadsto \lambda x.\ \texttt{patient}(x)$

(10.44) $\llbracket \mathrm{D}^{\mathrm{N}}(\texttt{every})\rrbracket \leadsto \lambda N_1.\ \lambda N_2.\ \forall x.\ (N_1(x) \rightarrow N_2(x))$

(10.45) $\llbracket \mathrm{D}^0(\texttt{every patient})\rrbracket \leadsto \lambda N.\ \forall x.\ (\texttt{patient}(x) \rightarrow N_2(x))$

(10.46) $\llbracket \mathrm{N}^0(\texttt{doctor})\rrbracket \leadsto \lambda x.\ \texttt{doctor}(x)$

(10.47) $\llbracket \mathrm{D}^{\mathrm{N}}(\texttt{a})\rrbracket \leadsto \lambda N_1.\ \lambda N_2.\ \exists x.\ (N_1(x) \wedge N_2(x))$

(10.48) $\llbracket \mathrm{D}^0(\texttt{a doctor})\rrbracket \leadsto \lambda N.\ \exists x.\ (\texttt{doctor}(x) \wedge N_2(x))$

The transitive verb has the high type announced in *figure 10.1*:

(10.49) $\llbracket \mathrm{V}^{\mathrm{T}}(\texttt{will interview})\rrbracket = \lambda D_2.\ \lambda D_1.\ D_1 \lambda x.\ D_2 \lambda y.\ \texttt{interview}(x,y)$

The derivation for the uninverted structure is unsurprising:

(10.50) $\llbracket \mathrm{V}^{\mathrm{I}}(\texttt{will interview every patient})\rrbracket$
$\leadsto \lambda D.\ D \lambda x.\ \forall y.\ (\texttt{patient}(y) \rightarrow \texttt{interview}(x,y))$

(10.51) $\llbracket \mathrm{V}^0(\texttt{a doctor will interview every patient})\rrbracket$
$\leadsto \exists x.\ (\texttt{doctor}(x) \wedge \forall y.\ (\texttt{patient}(y) \rightarrow \texttt{interview}(x,y)))$

but in the inverted structure, the **Lift**∃ rule is triggered:

(10.52) $\llbracket \mathrm{V}^0[\textsc{parent}\ \textbf{head}](\texttt{a doctor})\rrbracket$
$\leadsto \lambda D^N.\ D^N \lambda N.\ \exists x.\ (\texttt{doctor}(x) \wedge N(x))$

(10.53) $\llbracket \mathrm{V}^{\mathrm{I}}[\textsc{parent}\ \textbf{complement}](\texttt{a doctor will interview})\rrbracket$
$\leadsto \lambda D.\ \exists x.\ (\texttt{doctor}(x) \wedge D \lambda y.\ \texttt{interview}(x,y))$
$\overset{\textbf{lift}\exists}{\leadsto} \lambda D.\ D \lambda y.\ \exists x.\ (\texttt{doctor}(x) \wedge \texttt{interview}(x,y))$

(10.54) $\llbracket \mathrm{D}^0[\textsc{parent}\ \textbf{none}](\texttt{a doctor will interview every patient})\rrbracket$
$\leadsto \forall y.\ (\texttt{patient}(y) \rightarrow \exists x.\ (\texttt{doctor}(x) \wedge \texttt{interview}(x \wedge y)))$

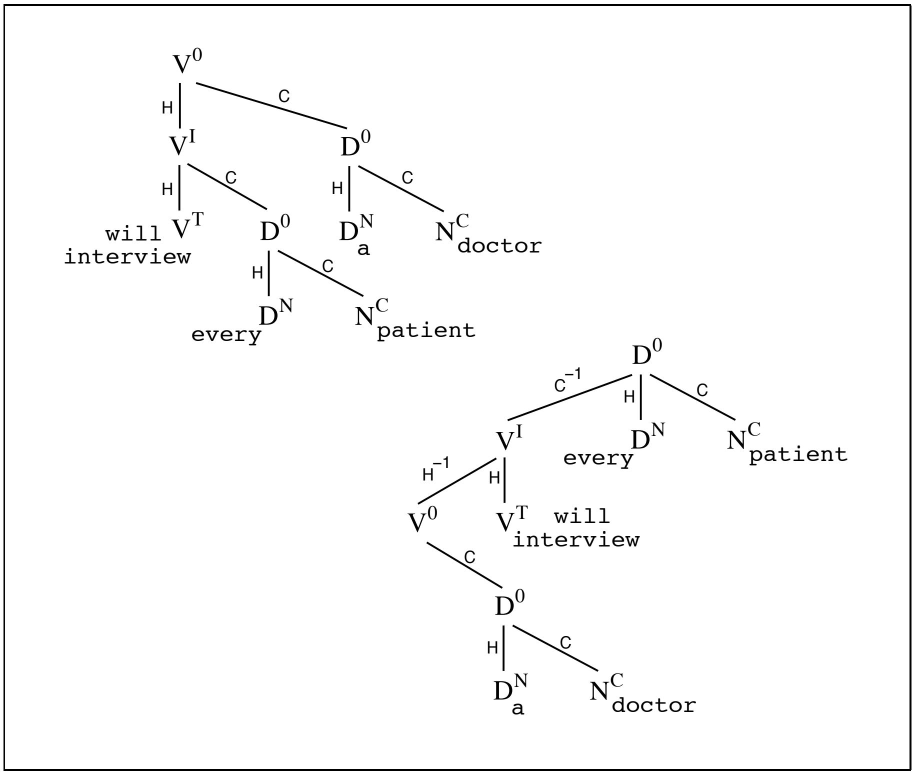


Figure 10.5: Scope readings.

Conclusions to chapter 10

This chapter discussed three major issues: (**i**) it showed that a very restricted form of tuple grammar, in which the cluster divisions are well-motivated, turns out to have a very reasonable capacity for describing complex word order phenomena, including those in languages that are often considered to have a totally free word order; (**ii**) this method allows for very general statement of word order phenomena such as order preservingness and head-right adjacency, which is especially useful in describing almost-free word order phenomena and (**iii**) it proposed a very tentative method to transform traditional structures in such a way that the main dependencies in relative clause attachment are localized. The last item should not be considered as inherent to **FHG**; in the **Epilogue**, the use of alternative methods to get this complete localization are discussed.

Various ideas taken together have appeared at other authors; order preservingness is a key notion in the *sequence union* based grammars of REAPE [Rea90]; the free form of head grammar splitting up constituents into three parts also appears in [vN93]. I did not find the intermediate characterization of head-right adjacency, which yields levels between the ultimately free form and the stricter form originally proposed by Pollard, in the literature.

In the spirit of the polynomial bound calculation done in section **5.2**, something can be said about the expected complexity of **LMG** grammars derived from sets of **FHG** principles and parameters. Disregarding the tree inversion of section **10.4**, there are 3 clusters and 6 variables, resulting in a first estimate of $\mathcal{O}(n^9)$. However, two of these variables generate a head that is always terminal, so if the implementation used indeed recognizes these as a single symbol, then two indices are dependent, so the space complexity would be $\mathcal{O}(n^5)$ and the time complexity $\mathcal{O}(n^7)$. In the presence of inverted dominance, this will increase to at least $\mathcal{O}(n^{10})$. But of course these are theoretical estimates, and a two-stage analysis based on a backbone **CFG** may cut down the average case by a polynomial factor.

It has been glossed over in this chapter, which in its last section claims to achieve ‘complete localization’ in the form of a universally bounded dependency domain, that there are still some phenomena for which no satisfactory solution has been proposed. An important example is *parasitic gapping*. For another example frequently cited, of the type “Sandy$_i$, Felix$_j$ was hard for Kim to give ε_j to ε_i” mentioned in the previous chapter, is a typical case that is treated in **HG** style frameworks at a semantic level: it is said that there is no *syntactic* dependency between *give* and *Felix*, but rather that *hard to give to Sandy* is an adjectival phrase.

Appendix

*

<table>
<tr><th colspan="2"></th><th><b>XG</b></th><th colspan="2"><b>Tuples</b></th><th><b>CG</b></th><th><b>TAG</b></th></tr>
<tr><td colspan="2">Formalisms</td><td>Chapter <b>2</b>
Standard (<b>2.1</b>)
Loose/Island (<b>2.2</b>, <b>2.3</b>)</td><td colspan="2">Chapter <b>3</b>
<b>MHG</b>, <b>MCFG</b>, <b>PMCFG</b>, <b>LMG</b></td><td><b>E.1</b></td><td><b>E.2</b></td></tr>
<tr><td>Tractability</td><td>Theory</td><td>Chapter <b>6</b>
Standard: r.e.
Bounded: EXPTIME
Loose: POLY</td><td colspan="2">Chapter <b>5</b>
Generic: r.e.
$\mathcal{S}$-<b>LMG</b>: POLY
<b>iLMG</b>: POLY
Bounded: EXPTIME</td><td>Unknown</td><td><b>TAG</b>, <b>MC-TAG</b>: POLY</td></tr>
<tr><td></td><td>Practice</td><td>—</td><td colspan="2">Chapter <b>7</b></td><td>—</td><td></td></tr>
<tr><td>Principles</td><td><b>wgc/sgc</b></td><td>Chapters <b>2</b>, <b>8</b>
German scrambling</td><td colspan="2">Chapters <b>3</b>, <b>8</b>
Mild context-sensitivity;
Implications of strong adequacy for <b>MHG</b> analysis: inverted dominance</td><td>Lambek: <b>CFL</b>
tuples (<b>LT</b>): <b>?</b></td><td><b>TAL</b> = <b>MHL</b>
Inverted dominance</td></tr>
<tr><td></td><td>Descriptive</td><td>—</td><td>Chapter <b>9</b>
<b>GB</b> without S-structure</td><td>Chapter <b>10</b>
<b>FHG</b></td><td>—</td><td></td></tr>
</table>

abbreviation	system	see
AGFL	finite lattice **CFG**	p134
CCG	combinatory **CG**	(p37)
CFG, **CFL**	CF grammar/language	p14
CG	categorial grammar	p218
cLFP	least fixed point calculus	p117
FHG	free head grammar	ch **10**
GB	government-binding theory	ch **9**
GPSG	generalized PS grammar	—
HG	standard head grammar	p196
HPSG	head PS grammar	—
iLFP	least fixed point calculus	p99
iLMG, **iLML**	index **LMG**	p88
L	Lambek **CG**	p218
LCFRS, **LCFRL**	linear CF rewriting system	p61, 64
LFG	lexical-functional grammar	(p35)
LIG	linear indexed grammar	(p37)
LMG, **LML**	literal movement grammar	p71
MC-TAG, **MC-TAL**	multi-component **TAG**	(p224)
MCFG	multiple **CFG**	p59
MHG, **MHL**	modified **HG**	p57, 62
PMCFG, **PMCFL**	parallel multiple **CFG**	p59, 70
TAG, **TAL**	tree adjoining grammar	p224
TGFL	finite lattice **LMG**	p134
XG, **XL**	extraposition grammar	p44
n-**LMG**, n-**LML**	**LMG** over $\leq n$-tuples	p75
$\mathcal{S}$-**LMG**, $\mathcal{S}$-**LML**	simple **LMG**	p76

abbreviation	stands for	see
ATM, **DTM**, **NTM**	Turing machines	p97
$\mathcal{BU}$, $\mathcal{TD}$, $\mathcal{LIN}$, $\mathcal{NE}$, $\mathcal{NC}$	clause types	p76
BAM	bounded activity machine	p93
CF	context-free	
CH, **HC**, **CWL**, **CWR**, **HWL**, **HWL**	**HG** clause types	p196
CNF	Chomsky normal form	p24
$\mathcal{HGY}$	head grammar operators	p59
ID/LP	immediate dominance/ linear precedence	
INFL, COMP	the categories I, C	
LHS, RHS	left/right hand side of clause	
NC, VC	noun/verb cluster	p49
PS	phrase structure	p29
AFL	abstract family of languages	p68
CKY	parsing algorithm	p94
GLR, **LALR**	parsing algorithms	p148
LOGSPACE, **PTIME**, **EXPTIME** *&c.*	complexity classes	p86
MCS, **MCSL**	mildly context-sensitive	p166
PTQ	λ semantics	p195
RAM	random access machine	p91
S, VP, NP, VR, VT, VI, CN, DP, PP, IP, CP	traditional categories	
wgc, **sgc**	weak/strong generative capacity	p24

Epilogue
Parallel and hybrid developments

The central subject of this thesis can be summarized as the investigation of grammar formalisms that *slightly* relax the principle of *projectivity* that is inherent to many approaches in linguistics. The three parts of this book have in particular shed light on one such family of formalisms, *tuple grammars*, contained in the generic family of *literal movement grammar*, from several perspectives inspired by the observations **A**-**J** made in the **Prologue**. The underlying motivations can be summarized as putting stress on two major issues that are important in the design of concrete language engineering systems: computational tractability and explanatory quality.

In the manner and order of presentation chosen, a number of connections, both between the different perspectives and between the investigated approaches and other frameworks in the literature, were not given attention. The figure on page 214 is a summary of what has been treated where; the fields marked **E** have been left open until now and will be discussed in this **Epilogue**.

To fill up the gaps in this landscape, I will first discuss briefly, in the sections **E.1** and **E.2**, what this thesis might have looked like when it would have concentrated on CATEGORIAL GRAMMAR (**CG**) or TREE ADJOINING GRAMMAR (**TAG**), instead of taking the traditional context-free phrase structure as a point of departure. Both **TAG** and **CG** are approaches that feature an emphasis on LEXICALIZATION that this thesis has not devoted much attention to, but which, as I intend to show, has nonetheless been implicitly present.

The final section **E.3** is a general discussion summarizing the perspectives of part **II**, part **III** and this **Epilogue**, then taking them together to draw a number of conclusions about the design of language engineering systems and highlight issues that need further investigation.

E.1 Categorial grammar

CATEGORIAL GRAMMAR (**CG**) is the favourite framework for many logicians in attempts to "find natural language syntax a home" in the domains of mathematical logic. Classical **CG**, introduced by BAR-HILLEL and AJDUKIEWICZ [BH53] is strongly equivalent to **CFG**; the well-known extension in the form of a calculus **L** by LAMBEK [Lam58] still generates the context-free languages, but has a form of *structural completeness* that allows the calculus to classify partial constituents. This correlation with **CFG** and the additional benefits of Lambek's system make it interesting to investigate **CG** as an alternative basis for the developments investigated in this thesis. Tuple-driven analysis in **CG** appears in the literature only in a very limited way (*cf.* extraction and the W marker in [Mor94], p. 106*ff*). A less logically oriented, more rule-based version of **CG** called COMBINATORY categorial grammar (**CCG**), mentioned elsewhere in this thesis for its computational properties and connections to **TAG** and **HG**, is beyond the scope of this **Epilogue**.

The discussion here is divided into the same three parts as the chapters of this thesis.

Formalisms

A standard CATEGORIAL GRAMMAR (**CG**) consists of a number of BASIC TYPES, a DISTINGUISHED TYPE s, which takes the function of a start symbol, and a LEXICON. COMPLEX TYPES are formed inductively from the basic types: a basic type a is a type; if x and y are types, then so are x/y and $x\backslash y$. The lexicon now associates with each element of a terminal alphabet T one or more types. From the types assigned to terminal symbols, types are now assigned to strings of terminal symbols as follows:

1. If a terminal symbol a is associated in the lexicon with type x, then $a : x$,
2. If $u : x$ and $v : x\backslash y$, then $uv : y$,
3. If $u : y/x$ and $v : x$, then $uv : y$ (where u and v are terminal strings).

The grammar now derives a string w if $w : s$ can be derived.

An example **CG** is displayed in *figure E.1*. It is straightforward to see that each standard **CG** has a strongly equivalent **CFG**—since the lexicon is finite, only finitely many types play a rôle, and a context-free grammar can be constructed that has these types as its nonterminal symbols. The mechanism of type formation in fact closely resembles the way the *control types* work in the **FHG** system in chapter **10**. In this formulation, **CG** is just a linguistically well-motivated way of organizing a context-free grammar—in particular, **CG** has the same clean interface between syntax and λ-semantics as **FHG** where heads are functions and complements are arguments.

Since Bar-Hillels publication, **CG** has been extended with various rules allowing types to derive new types in non-standard fashions, such as in the following lifting distinguished type is s

`Frank` :	n
`Julia` :	n
`Fred` :	n
`slept` :	$n\backslash s$
`hates` :	$(n\backslash s)/n$
`admires` :	$(n\backslash s)/n$

$$\dfrac{\texttt{Frank} : n \qquad \dfrac{\texttt{admires} : (n\backslash s)/n \quad \texttt{Julia} : n}{\texttt{admires Julia} : n\backslash s}}{\texttt{Frank admires Julia} : s}$$

Figure E.1: Small categorial lexicon and a derivation

rule:

(10) $x/y \rightsquigarrow (z\backslash x)/(z\backslash y)$

and following that, interest in **CG** has gradually diminished, until a while after LAMBEK introduced the calculus **L**, that adds *introduction* rules for the slash operators, in addition to Bar-Hillel's two rules that only *eliminate* the slashes. In Lambek's calculus, all proposed extended rules can be derived.

Lambek's system is as a standard **CG**, but for the grammar to recognize a string $w = a_1 \ldots a_n$, a SEQUENT $\Gamma \vdash s$ must be proved in the calculus L in *figure E.2*, where $\Gamma = t_1 \ldots t_n$ is a sequence of type such that type t_i is associated with terminal a_i. Unlike classical **CG**, Lambek system is not *strongly* equivalent to **CFG**, because it turns out that due to the introduction rules, Bar-Hillel and Lambek categorial grammars with the same lexicon do not always generate the same string set. It was proved long after Lambek's 1958 paper in [Pen] that Lambek's strongly more powerful calculus nonetheless generates precisely the context-free languages.

axioms : $x \vdash x$

$$\dfrac{\Gamma \vdash x \quad \Delta_1, y, \Delta_2 \vdash z}{\Delta_1, \Gamma, x\backslash y, \Delta_2 \vdash z}\ \backslash\mathbf{L} \qquad \dfrac{x, \Gamma \vdash y}{\Gamma \vdash x\backslash y}\ /\mathbf{R}$$

$$\dfrac{\Gamma \vdash x \quad \Delta_1, y, \Delta_2 \vdash z}{\Delta_1, y/x, \Gamma, \Delta_2 \vdash z}\ \backslash\mathbf{L} \qquad \dfrac{\Gamma, x \vdash y}{\Gamma \vdash y/x}\ /\mathbf{R}$$

Figure E.2: Lambek's calculus **L**.

Lambek derivations take some practice to read, because the sequent notation introduces a notion of *hypothetical reasoning*—the following corresponds to the classical **CG** derivation in *figure E.1*.

(11)
$$\dfrac{n \vdash n \qquad \dfrac{n \vdash n \quad s \vdash s}{n,\ n\backslash s \vdash s}\ \backslash\mathbf{L}}{n,\ (n\backslash s)/n,\ n \vdash s}\ /\mathbf{L}$$

An introduction rule can now be used to derive a new type for a partial constituent consisting of a subject and a transitive verb, that forms a sentence when combined with an object to its right:

(12)
$$\dfrac{n,\ (n\backslash s)/n,\ n \vdash s}{n,\ (n\backslash s)/n \vdash s/n}\ /\mathbf{R}$$

If the word *and* is now given the following lexical entries,

(13) $\texttt{and}:\ s\backslash(s/s)$
$\texttt{and}:\ (n\backslash s)\backslash((n\backslash s)/(n\backslash s))$
$\texttt{and}:\ (s/n)\backslash((s/n)/(s/n))$

the lambek calculus will recognize, apart from conjoined sentences and intransitive verb phrases, the partial conjunction (14), which is missed by Bar-Hillel's formulation, using 3 applications of the **Cut** rule in (15) which can be added to the calculus without increasing the set of derived sequents.

(14) Frank admires and Fred hates Julia

(15)
$$\dfrac{\Gamma \vdash x \quad \Delta_1, x, \Delta_2 \vdash y}{\Delta_1, \Gamma, \Delta_2 \vdash y}\ \mathbf{Cut}$$

This ability of dealing with non-constituent co-ordination is one of the more important features of Lambek grammar. Various approaches to phenomena such as the Dutch crossed dependency structure have also been proposed, but often suffer from overgeneration—many stronger systems turn out to describe only the regular languages—and detailed linguistic analysis is either disregarded or leads to systems that get either excessively complex and lose their elegant logical formulation [Moo88] [Mor94] [BvN95]. One approach, that of *multi-modal* categorial grammar, has been suggested to me to provide for analogues of the tuple paradigm. The relationship is not straightforward, because multi-modal **CG** tend to identify scopes within a traditional left-to-right tree analysis in which binding of some elements may be postponed, hence achieving long-distance dependencies. But this is exactly what tuple grammars were designed *not* to do. Nonetheless, an in-depth analysis of the relationship between multimodal **CG** and Reape's word order domains, whose connection to the tuple grammars of this thesis suggests itself more clearly but is not entirely trivial, can be found in [Ver96].

axioms: $\phi : x \vdash \phi : x$
structural rules: cut, contraction, permutation, weakening

$$\frac{\Gamma \vdash \phi : x \quad \psi : y,\ \Delta \vdash \chi[\psi] : z}{\Gamma,\ \eta : x \xrightarrow{f} y,\ \Delta \vdash f(\eta, \chi[\phi])} \to \mathbf{L}$$

$$\frac{\Gamma,\ \phi : x \vdash \chi[\phi] : y}{\Gamma \vdash x \xrightarrow{g} y} \to \mathbf{R}$$

where ϕ, χ, ψ, η are tuples over terminal strings,$f, g \in \mathcal{F}$ and g is defined by $g(\psi) = \chi[\psi]$.

Figure E.3: Lambek-tuple calculus **LT**.

The close resemblance to the control types of an **FHG** (chapter **10**) and the types assigned to lexical entries in a categorial grammar makes it worthwhile to investigate whether **CG** and the tuple paradigm can be combined in a systematic fashion. A simple tuple-based operation appears in [Mor94], but is given in the form of an extra operator in addition to $\backslash$ and $/$ and needs explicit extra rules. The resulting systems, which often introduce a wealth of such extra operators, lose the elegant compactness of the original **L** calculus. One key to a solution is, as is already done in [Mor94], to abandon the implicit rule of associative concatenation in the sequents of the calculus, and move to a system of *labelled deduction*; but an essential further step is to let all operators collapse into a single functional *arrow* operator annotated with a yield function that operates on tuples of strings.

Let the class $\mathcal{F}$ be a set of yield functions, and let a set of basic types and a distinguished type be given. Then complex types are either basic types or of the form $x \xrightarrow{f} y$ where $f \in \mathcal{F}$ and x and y are types. Let the lexicon assign sets of tuples of strings to a finite subset of the types, and let the rules of inference be as in the calculus **LT** in *figure E.3*. A prudent choice for $\mathcal{F}$ may be the linear, nonerasing subset of the **FHG** operators, resulting in a form of "free Lambek head grammar". If $\mathcal{F}$ consists only of the left and right concatenation operators over 1-tuples, and the lexicon assigns types only to single terminal symbols, the resulting grammar system corresponds to the Lambek calculus. If the concatenation operators are given the same labels as in chapter **10**, then as an advantage of the **LT** notation, the confusion[1] about the directionality of the categorial backward slash between different authors is eliminated: the forward slash becomes $\xrightarrow{\mathbf{HC}}$, and the backward slash $\xrightarrow{\mathbf{CH}}$, and in both cases the argument appears left of the arrow, the result to its right. The type expressions can be declared right-associative by the well-known convention for writing down functional

[1] Some authors write $y\backslash x$ where I would write $x\backslash y$—that is, there are different opinions as to whether the numerator should always be 'above' or always be 'left' of a slash denoting categorial division.

type expressions (*cf.* chapter **10**).

The following derivation step is the **LT** equivalent of (12), but in an analysis that models the Dutch sentential order as in the **FHG** examples of chapter **10**:

(16) $$\frac{\begin{array}{l}(\varepsilon,\texttt{Frank},\varepsilon):n,\ (\varepsilon,\texttt{bewondert},\varepsilon):n\xrightarrow{\mathbf{CH}}n\xrightarrow{\mathbf{CH}}s,\\ (\varepsilon,\texttt{Julia},\varepsilon):n\quad\vdash(\texttt{Frank Julia},\texttt{bewondert},\varepsilon):s\end{array}}{\begin{array}{r}(\varepsilon,\texttt{Frank},\varepsilon):n,\ (\varepsilon,\texttt{bewondert},\varepsilon):n\xrightarrow{\mathbf{CH}}n\xrightarrow{\mathbf{CH}}s\\ \vdash(\texttt{Frank},\texttt{bewondert},\varepsilon):n\xrightarrow{\mathbf{CWL}}s\end{array}}\rightarrow\mathbf{R}$$

The **LT** grammar system can be used to give a model of non-projective surface order phenomena in the spirit of the solutions proposed in this thesis, with the additional benefit of an account of non-constituent co-ordination. Of course, connectives like *and* still need to be explicitly assigned polymorphic types, since they cannot be derived within the calculus [Moo88]. This is more difficult in a tuple-based system as finding the entries for multi-cluster constituents may not be straightforward.

I did not find the time to investigate the details of a system that allows *f* and *g* in the **LT** rules to be arbitrary *relations* rather than functions, which would lead to systems weakly equivalent to brands of **LMG** that are closed under intersection.

Computational tractability

While Lambek grammars weakly generate the context-free languages, a transformation of a Lambek grammar to a **CFG** results in exponential blow-up. The complexity of universal recognition, based directly on the **L** calculus is unknown—up to now, nondeterministic polynomial algorithms are the best known.

These results will carry over straightforwardly to the **LT** system, with the exception perhaps of the weak equivalence proof between sensibly chosen classes of tuple grammars and the corresponding classes of **LT** systems. Problematic may also be the fact that in the current formulation, the lexicon can assign the empty string to types—but this is easily revised.

An important practical problem of Lambek **CG** is *spurious ambiguity*: while ordinary **CG** assigns only single tree structures to a sentence, Lambek grammars derive each possible binary branching analysis. These must ideally be contracted to a single analysis, but this is not a simple task. While this is a valid objection from the perspective of an adherent of **CFG**, it should be noted that an **FHG** analysis of Latin as proposed in chapter **10** suffers from the same problem: the surface order in Latin is so free that a sentence with a fairly straightforward word order will get various analyses, from which preferably only the simplest, non-wrapping analysis should be extracted. There is one notable difference between these two types of spurious ambiguity—in the **FHG** case, the different equivalent derivations will differ only on the RULE feature but are identical otherwise, and can hence be merged easily.

The move from a standard tuple-based framework such as **FHG** to the **LT** system will need to accept degradation of computational tractability properties and spurious ambiguity in exchange for the ability to describe and study non-constituent co-ordination in combination with the tuple-driven descriptive methods of chapters **9** and **10**.

It should finally be noted that unlimited polymorphic types for a word like *and* further deteriorate the computational properties—but as before, there is no essential difference between **L** en **LT** here.

Principles

There is a large resemblance to the way **FHG** in the previous chapter abstracted over the application of concrete **LMG** productions, and the way a categorial grammar shifts all information to the lexicon while retaining only a small number of explicit rules. The **LT** system takes this shift a bit further, by abstracting away over the difference between the operators, moving those to the lexicon too. Principled approaches to non-constituent co-ordination have been investigated elaborately in the context of Lambek grammars. The use of **LMG** sharing for modelling co-ordination phenomena becomes redundant as soon as categorial type lifting is used. It is to be expected that the rôle of sharing is thus reduced considerably. I have not been able to find out whether the remaining phenomena that are beyond the reach of linear **MCFG**, such as Old Georgian suffix stacking and Chinese number names (*cf.* chapter **8**) can be dealt with in a Lambek tuple grammar without use of terminal sharing or reduplication.

Conclusion

Large parts of the work done in this thesis could have been laid out in a categorial setting, and this is likely to lead to interesting benefits. However, it is unlikely that I would have found an analogue for part **II** on computational complexity and the possibility to add lexical sharing ‘for free’: categorial grammar is designed to perform well as a logical system and is thus fairly automatically too heavy to be used in practice in its pure form: it is probably NP-complete. Furthermore, the low cost of the sharing construction is considerably less interesting for a grammar system like Lambek’s that is not known to, and likely not to, be processable in polynomial time, and which has mechanisms that reduce the need for sharing.

E.2 Tree adjoining and head grammar

TREE ADJOINING GRAMMAR (**TAG**) is a formalism that manipulates surface trees through the operations of substitution and adjunction. It has already been mentioned briefly at various places in this thesis, and is weakly equivalent to **MHG**. A progression similar to that from **CFG** to linear **MCFG** has been made for **TAG**—the result is the MULTI-COMPONENT **MC-TAG** formalism, which is weakly equivalent to linear **MCFG**. Slight extensions of the standard form of **TAG** are popular in fields of research similar to that reported on in this thesis.

I will now give a brief definition of **TAG**, and show some parallels between what was done in sections **8.2** and **10.4** and the approach **TAG** take in describing Dutch crossed dependencies and localizing relative clauses. This is a good point of departure for an investigation of what the results in this thesis, for as much as they have not already been obtained in the literature, would look like in a **TAG** context. Finally, I propose ways of embedding tuple-based surface generation and lexical sharing into a **TAG** framework, using the parallel between **TAG** and **MHG** that is often used to construct simple recognition procedures for **TAG**.

Definition and examples

In a context-free grammar, surface structure and derivational and deep structure are taken to be identical. This is known to raise various problems (see chapter **1**), which were solved in this thesis by dividing the yield of constituents into tuples; tree adjoining grammars solve these problems by assigning to a sentence *two* structures: a *derivation tree* and a *derived tree*. According to the definitions in section **1.3**, the derivation tree is a dependency structure and the derived tree a traditional surface structure.[2]

E-1 definitions. A TREE ADJOINING GRAMMAR (**TAG**) is a tuple (N,T,S,E^I,E^A) where N, T and S are as for context-free grammars, and

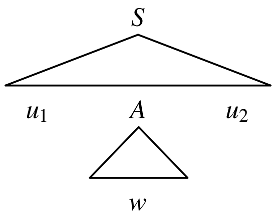
- ▷ An INITIAL TREE is a tree whose root node is labelled with the start symbol S and whose leaves are labelled with terminal strings only.
- ▷ An AUXILIARY TREE is a tree with a root node labelled with a nonterminal symbol A, *one* leaf, the FOOT NODE, also labelled A and all other leaves labelled with a terminal string.
- ▷ $E = E^I \cup E^A$ is a set of ELEMENTARY TREES, divided into a set of initial trees E^I and a set of auxiliary trees E^A.

A **TAG** derives tree structures, by the following operation of ADJUNCTION. Let t be an initial or auxiliary tree, with an internal node labelled A, and let s be an auxiliary tree

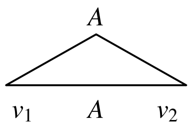
[2] To keep this discussion brief, I have omitted in definition **E-1** the substitution operation and OA/NA constraints that are usually added in concrete **TAG** examples.

whose root is A:

(17) $t =$

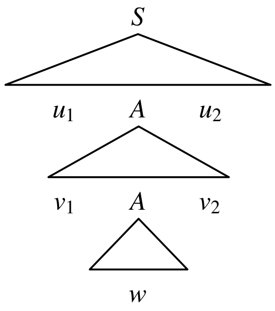


$s =$



then the result of adjoining s into t at the A node is

(18) $s \oplus^d t =$



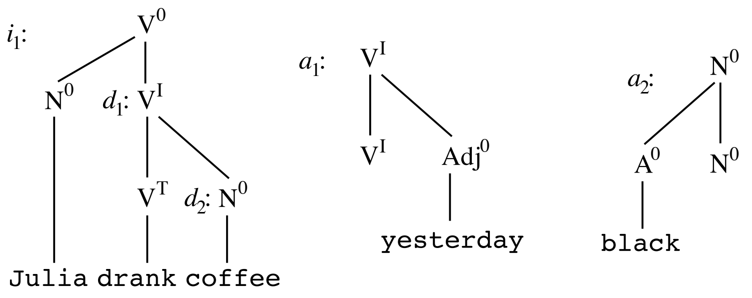
where d is the address of the node labelled A in t.

Now, inductively, a DERIVED INITIAL TREE is an initial tree that is either elementary or obtained by adjoining a derived auxiliary tree into a derived initial tree; and a DERIVED AUXILIARY TREE is either elementary or obtained by adjoining a derived auxiliary tree into another derived auxiliary tree.

A **TAG** now derives a sentence w if there is a derived initial tree whose yield is w. This tree is called the DERIVED TREE FOR w. The term constructing this tree from elementary trees using the adjunction operator $\oplus$ is called the DERIVATION TREE for w.

E-2 example: English complementation. Adjunction is typically used in a **TAG** to describe adverbial and adjectival modification, but also for verbal complementation. In a simple **TAG** without verbal complementation, the initial tree into which trees are adjoined is always the matrix clause. The following are examples of elementary trees in a simple **TAG**.[3]

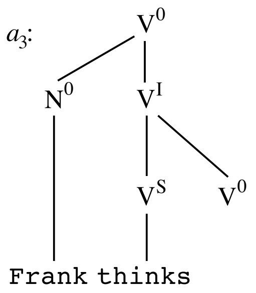
(19)



The sentence *Julia drank black coffee yesterday* is now derived as $(i_1 \oplus^{d_2} a_2) \oplus^{d_1} a_1$.

The complementation verb *think* is now entered in the **TAG** not in an initial tree, but through adjunction.

(20)



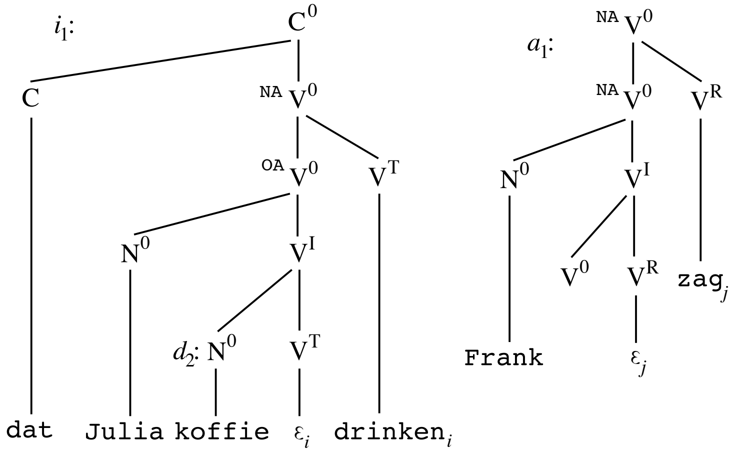
The sentence *Frank thinks Julia drank coffee* is now derived by the adjunction $i_1 \oplus^{d_0} a_3$. Note that the matrix verb frame *Frank thinks* ... is, in the derivation tree, daughter rather than parent. This has some analogy to the inverted dominance relations suggested in section **10.4**, and a similar mechanism is used in **TAG** to completely localize the dependencies involved in relative clause attachment.

E-3 example: crossed dependencies. The very same mechanism makes that **TAG**s easily generate Dutch crossed dependencies. However, the following example set of elementary trees for Dutch reveals a problem that was implicitly present in the English example: the initial tree corresponds to the inner verb clause, but the root of this initial tree nonetheless represents the root of the derived sentence. This corresponds to the "weird" case in the strong generative capacity investigation for **MHG** in section **8.2** in which the lowest node of the derivation contains material from

[3] In practice, one adds the notion of *substitution* and replaces the concrete subject and objects under the N^0 in the initial trees with the substitution marker ↓.

the top level of the sentence.

(21)



If the Dutch set is to be extended with declarative and interrogative sentences, there must be additional initial trees for *Julia koffie drinken* to yield each of these situations. But in parallel with this, the auxiliary tree must also come in various versions to get initial and second position of the finite verb.

*

The latter example highlights a more general problem in **TAG**. The elementary trees are, on the one hand, supposed to model *minimal linguistic structures*, for example, a verb with its complement nodes. In this way, all the nodes that are in a dependency relation stem from one lexical item, a lexical item being an elementary tree. But as more complex phenomena are treated, such as relative clause attachment and Dutch crossed dependencies, the elementary trees become complex structures. It is common practice in practical **TAG** grammars to have large macro schemes that generate elementary trees. These schemes quickly become so intricate that they become part of the grammar system. It is unclear to me whether this is a desirable feature, and the Dutch example seems to highlight this particularly well.

Tags and tuples

It seems reasonable to assume that, even though the mechanisms employed to obtain localization of relative clause attachment and crossed dependencies in **TAG** are essentially the same, the analysis is acceptable in the first case, but becomes a burden in the second. It seems, at least, evident that the tuple-based method of generating crossed dependencies is less involved—and it has the benefit that word order generalizations can be stated easily in terms of order preservingness and head-right adjacency, *cf.* chapter **10**. On the other hand, the tuple-based (**9.4**) and tree-inversion (**10.4**) methods of topicalization and *wh*-extraction are more involved and no more defendable than the **TAG** analysis. Therefore it interesting to investigate to what extent both methods can be combined. This would yield a system that generates word order in verb phrases through tuples, and has an elegant way of turning around the dominance relation in the derived tree while maintaining derived trees in the traditional shape, replacing the tree inversion paradigm proposed in section **10.4**.

In a traditional **TAG**, the elementary and derived trees correspond to context-free derivation trees. It is easy to make the step to **TAG**s over **LMG** or **FHG** derivation trees. It seems to follow straightforwardly from the way **TAG**s can be translated to head grammars, that such grammars have tractable recognition, as is illustrated in the following sketches.

E-4 sketch: TAG recognition and MHG [WVSJ86]. This is a simplified sketch of the procedure outlined by Weir, Vijay-Shanker and Joshi. Assume that the elementary trees in the **TAG** have binary branching and no adjunction constraints (**NA**, **OA**). The structure of the elementary trees of a **TAG** can be described straightforwardly using **MHG** productions. There is a nonterminal X_d for each node d of each elementary tree t. For each internal node d with daughters d_1 and d_2, there is a production

$$X_d \rightarrow \textit{head-complement}(X_{d_1}, X_{d_2})$$

if the tree t is initial or t is auxiliary and the foot node of t is in the subtree rooted by d_1; or

$$X_d \rightarrow \textit{complement-head}(X_{d_1}, X_{d_2})$$

when the foot node of t is in the subtree rooted by d_2. With these productions, the nonterminals X_d for root nodes d derive the yield of the initial and auxiliary trees, split up, in the auxiliary case, at the point of the foot node.

Now, to model adjunction, for each internal node d labelled with a nonterminal A and each auxiliary tree with root r also labelled A, there is an additional production

$$X_d \rightarrow \textit{head-wrap}(X_d, X_r)$$

which has the effect of wrapping the yield of the auxiliary tree around that of the subtree under d.

The start symbol of the **MHG** is rewritten to the root nodes of all the initial trees. Now the **MHG** recognizes the same strings as the original **TAG**.

*

This idea of considering a **TAG** as a phrase structure grammar that derives strings with “holes” in them, *i.e.*, head grammars, can be extended to adjoining systems over tuple derivations. In this case however, these “holes” are best represented directly, rather than implicitly as the split points in an **MHG**.

E-5 sketch: HG-TAG. Let a set of initial and auxiliary trees be given, whose nodes are consistently annotated with **MHG** productions. From these trees, an **iLMG** is derived which encodes each node in each elementary tree as a predicate over 8 indices:

$$X_d(l_1, r_1,\ l_2, r_2,\ i_1, j_1,\ i_2, j_2)$$

representing the yield of a normal **MHG** predicate split up into a left component $a_{l_1} \cdots a_{r_1}$ and a right component $a_{l_2} \cdots a_{r_2}$ with two holes, stretching over $a_{i_1} \cdots a_{j_1}$ and $a_{i_2} \cdots a_{j_2}$. These holes can both be in either component, and correspond to the two components of the foot node. It is now straightforward to write down the productions that make up the elementary trees and the productions that define adjunction.

*

It appears that this can be extended without many problems to linear **MCFG**, and possibly also to arbitrary simple **LMG** trees, which would lead to an extension of **TAG** that describes precisely the polynomial-time recognizable languages like simple **LMG**. It is also interesting to investigate whether an **MC-TAG** over 1-$\mathcal{S}$-**LMG** trees, *i.e.* context-free trees with sharing, will generate **PTIME**.

Another extension of **TAG**, called *D-tree grammar*, has been proposed in [WVSR95] which provides a solution for the fact that adjunction is used in **TAG** both for modification and verbal complementation, and this yields derivation structures that capture almost, but not completely the right dependency structures. It is expected that it will be possible to carry out constructions similar to the ones proposed here for **DTG**.

E.3 Concluding discussion

The TUPLE GRAMMARS of chapter **3** form the basis for the bulk of the work done in parts **II** and **III**. The essential qualities of the **LMG** formalism, which extends the previously known formalisms **LCFRS** and **PMCFG**, are (**i**) a notation which makes them a more accessible tool for concrete use in writing simple grammars or in building substantial grammatical frameworks, and (**ii**) the ability to model sharing of substrings of the input between different parts of a derivation in a restricted version that describes precisely the languages recognizable in polynomial time.

Precisely this ability to describe sharing is also a problem of simple **LMG**. At a formal level, it is a problem because generic Turing machine implementations of recognizers have an essentially worse time complexity than equivalents for non-sharing formalisms such as **PMCFG** (chapters **4** and **5**). In more practical approaches, based on real-world, random access models of computation, this problem does not occur, but here the sharing is a source of possibly undesired parallelism in phases of post-structural analysis that assign finite attributes to **LMG** derivations. A related problem is that Nederhof/Sarbo's algorithm for calculating finite attributes runs into a problematic case with some sharing **LMG**s. An additional problem is that sharing complicates the application of **LMG** to input in the form of a *lattice* or finite automaton as is often done in speech analysis; this problem is analogous to the possible misinterpretations made by a sharing analysis of co-ordination: sentences like ... *dat Anne Frank binnenkwam en groette* (*that Anne Frank came in and [Anne] said hello to [Frank]*) are found correct. In the lattice equivalent, different choices for unclearly pronounced morphemes may be selected where shared material is used in different parts of the same derivation.

Altogether, sharing is a property of **LMG** that is to be used, if at all, with extreme caution. From weak-generative and computational points of view however, is has also been shown that simple **LMG** clearly has its merits (chapters **8** and **5**).

*

EXTRAPOSITION GRAMMAR was introduced as a candidate for providing solutions for the surface order phenomena under investigation, and in chapter **6**, some gaps in the available knowledge on **XG** were filled in that showed that unless some modifications were made, **XG** was intractable. Let me summarize briefly why the thread on **XG** abruptly ends here while the tuple grammars proceed to play a major rôle in part **III**.

At the level of attribute evaluation, there is a problem with **XG**—the tractable methods of analysis based on *loose* **XG** proposed in chapter **6** yield tree structures in which the trace-filler connections are not explicitly connected. It is, at this point, unclear to me if and how these connections could be re-established in a forest representation. It seems clear to me that, to be interesting at all, an annotated **XG** would have to be allowed to make use of affix constraints between trace and filler positions. The current knowledge on dealing with annotation of ambiguous *tree* representations is still highly limited—it is hard to imagine what new problems will be introduced if one should attempt to define attribute phases over ambiguous *graph* representations that are to be the output of a generic **XG** parser. This encourages an optimistic interpretation of 'tractable syntax' as formalisms that achieve a bounded dependency domain *and* produce tree-shaped structures.

Both **XG** and the proposed modified versions also seem to lend themselves less well for a principled, explanatory approach to the desired classification of 'mild projectivity' and **XG** did not return as a major topic in part **III**. Of course this is not meant to say that a study in the spirit of chapters **9** (**GB**) and **10** (**FHG**) should a priori be excluded as not promising. STABLER [Sta87] has looked at **XG** and **GB** principles and let these influence the design of extensions to **XG** that seem to be aimed mainly at application in simple **DCG** based grammar systems than at being a basis for explanatory approaches to word order.

Nonetheless, loosely interpreted **XG** was used in section **8.2** to argue that scrambling in German can be processed in polynomial time. It is an entertaining observation that I have not found an equivalent simple **LMG** description.

*

PART **II** (chapters **6**–**7**) already put itself in a larger context, when briefly discussing approaches in the literature that are in collision with the claims and results I presented. These discrepancies are concentrated around the familiar debate whether, on the one hand, tractability claims such as the ones made in this thesis are convincing, and on the other hand, whether evidence for the *in*tractability presented in various other literature is conclusive.

A representative series of arguments for the intractability of natural language can be found in works by RISTAD, BARTON and BERWICK [BBR87] [Ris90]. The general, justified, worry of these arguments is that as soon as formally minded, 'platonic', grammar systems are studied as a tool for concrete description, performance parameters deemed unimportant in the optimistic formal literature become a bottle-neck. Most of the case studies in [BBR87] and [Ris90] however suffer, as said before, themselves from 'formal optimism': they do not investigate the size of the input and various grammar measures independently. Many references to intractability of natural languages proceed along these lines: (**i**) a doubt or series of doubts is expressed, which is insufficiently formalized; (**ii**) a series of formal results is presented to substantiate the doubt, but what is proved is not conscientiously brought in connection with the doubts.

Strictly speaking therefore, the conclusions drawn can often at best be read separately as unsubstantiated claims, and the formal results as mathematics lacking a motivating context. In the case of simple **LMG** recognition, we even have proofs of both tractability and intractability living happily together (section **5.3**).

Another delicate matter, but one less important vis-a-vis the tractability claims in this thesis, is whether it is justified to draw conclusions about natural language in general from a study of certain grammar formalisms. In his PhD thesis [Ris90], Ristad puts stress on the importance of a so-called *direct complexity analysis*, which takes indisputable knowledge about language as a basis for an (in)tractability proof, and does not rely on a particular language model. However, as an instance of such an analysis, Ristad (page 42) proposes to take a phenomenon (agreement and lexical ambiguity); to show that independent of whether one attempts to describe it using a unification based formalism or a transformational theory, this leads to an intractable task; and to conclude that hence the task in general is intractable. So a direct analysis is one that does not depend on one syntactic theory but on two theories. Evidently, if the complexity of a *phenomenon* is to be investigated, independently of the grammatical framework, this type of analysis is also to be avoided. In particular, Ristad seems to have to rewrite his chapter 3 after reading chapters **9** and **10** of this thesis—in fact the chapter needs to be rewritten every time a new approach is sketched that seems to tackle the problem he studies.

The following arguments in the literature do *not* suffer from the problems mentioned above.

1. From a perspective of weak generative capacity and fixed recognition, adding finite attributes to a grammar system equivalent to or strictly stronger than **CFG** is free, because the attributes can be encoded into the set of nonterminal symbols. It is well known that in the worst case, the resulting expanded grammar whose nonterminals form an attribute space has at least exponentially more rules than the original grammar, taking the grammar beyond the reach of any real-world computer system. Direct processing has been proved in general not to be able to improve on this situation: recognition of finitely annotated context-free grammars is EXP-POLY time hard in the size of the grammar. Moreover, there are convincing arguments that the attribute spaces that are typically used in natural language grammars often trigger this worst case ([BBR87] appendix B).

2. Binding phenomena or anaphora ([Ris90] chapter 4) are problematic, and have in this thesis simply been put away as 'post-syntactic'. This is in line with the idea of dividing language analysis in a series of dedicated small steps.

3. The one truly *direct* complexity analysis in Ristad's thesis is in chapter 2, *structure of phonological knowledge*. If I put anaphora away as post-syntactic, phonological and morphological analysis has been put aside as *pre-syntactic*, which I find a more worrying lack on my side, because it considerably weakens the desired conclusion of tractability of the syntactical analysis of written language.

   A number of aspects that make up arguments for intractability of complex morphology that *do* refer to known grammatical theories, such as reduplication in [BBR87] chapter 5, are treated by the formalisms in this thesis (and in fact already by parallel **MCFG**).

Many systems used for concrete grammar writing today however are based on theoretically intractable frameworks, with varying degrees of success. Many such systems are indeed slow and cover only very limited fields or fragments of language, but many also show good benchmarks on representative concrete examples. In the case of Nederhof and Sarbo's algorithm, discussed in chapter **7**, it can be concisely assessed based on representative example grammars, how much benefit multi-stage analysis can give.

The examples seem to indicate that while the overall time and space complexity of such multi-stage approaches remains of the same order, the time and space benefits grow at the same rate as the complexity function of the algorithm. So such approaches are not mere engineering optimizations but deserve to be used as concrete data in the discussion on tractability of natural language processing.

A pressing question is whether it is not the case that whatever is added to the growing amount of lexical information in natural language systems, this information can always be divided into sets of attributes or feature paths that do not interact. An example is subcategorization and finite selectional-semantic properties versus morphology. If this is the case, such groups can be processed in successive stages of attribute evaluation, rather than simultaneously. If such chunks of information are found to have a concrete bound, predictions can be made about the concrete feasibility of natural language tasks, since the exponential size parameters will then have been fixed. This line of reasoning is feasible only under a strict regime of finite attributes, and is many times more complicated in a realm of generic feature structures and unification—this has been an important motivation for my decision to put stress on finite attribute analysis.

*

CHAPTER **8** on generative capacity is a bridge between the mathematically oriented parts **I** and **II**, and part **III** and this **Epilogue** which have a very broad perspective. The construction in section **8.2** done for **MHG** is reflected twice later, in the tree inversion paradigm proposed in section **10.4**, and in **E.2**, where it is mentioned that in **TAG**, relative clauses are also described by virtue of the fact that in **TAG**'s derivation structures, the verbal clause subsumption relation is reversed. The construction in **8.2** can be read as proving that in **TAG**, like in **MHG**, this reversal is a consequence of the desire to localize relative clauses. In combination with a description of languages like Dutch and German that have verb second, this implies that lexical frames for infinitive verbs must necessarily be multiplied out over a rather large set of combinations of finite verb positions and topicalization types. In **E.2**, it is proposed that it is therefore a worthwhile idea to *hybridize* **TAG** and **HG**.

*

This idea of *hybridization* is present throughout this thesis, and is a result of my attempt to achieve a broad coverage while keeping the terminology, meta-theoretical notions, and notation consistent. This has resulted in a book that consists for large parts of the presentation of known formalisms and results, but adds small new results at various places, and, in particular, is at the end able to draw parallels that were difficult to make had all material presented here not been put together under a single cover.

The proposal of hybrid systems in the previous two sections, dealing with **CG** and **TAG**, is similar to the conclusions drawn in chapters **7** and **10**: after careful assessment of each formalism for its particular qualities, and the possible interfaces between different worlds, a good approach may be to take a component from each of a small number of systems, and build something that applies each of these components for what it is good at—so as to obtain a system that is good at everything. A general example is the capability of many paradigms to describe crossed dependencies in Dutch: in **TAG** for example, showing this capacity in the literature is done as a formal exercise, but in practical systems, the exercise no longer satisfies 'grammar quality' standards, as the way the minimal trees and the operation of adjunction are used is nonstandard. Nonetheless, **TAG** provide a well-motivated model for adjunction and localization of relative clause attachment. A similar story holds for the capacity of a Lambek-style calculus to deal with non-constituent co-ordination, and again the same goes for the stages of context-free prediction and true **LMG** parsing, and to various levels of specification of finite morphological attributes or semantic domain information.

This is in line with the original objective of the project in which I carried out the research for this thesis: to investigate various approaches to trans-context-sensitivity and aim at forms of "cross-fertilization". A possible point of objection is that the rôle reserved for methods from a Computer Science or Software Engineering background is at best present implicitly—however, it is, as such, present very strongly, because the software engineering tools I looked at in the first years of my research led strongly to the desire for a syntactic formalism that eliminated completely those ambiguities that are introduced by methods used to describe phenomena of surface discontinuity in analyses based on a context-free grammar.

*

A FURTHER EXAMPLE OF HYBRIDIZATION is the design of a complete system based on **FHG**. It seems clear that the **FHG** paradigm as set out in chapter **10** is very well suited to tackle the basic word order of languages like Latin and Dutch on a principled, explanatory basis. However, for the description of the remaining long-distance dependencies, and for co-ordination, various options are open. It has been shown now that **FHG** can be successfully merged with techniques borrowed from **TAG** and Lambek categorial grammar. One might equally well prefer to use additional surface clusters, *i.e.*, larger tuples, as proposed in chapter **9** on **GB** theory. When efficiency of processing is at stake, the Lambek versions are an unlikely candidate. It should also be noted that while a surface cluster with a finite SLASH feature, as proposed in the **GB** chapter, may lead to a loss of efficiency, since this feature information needs to be maintained in a forest representation, and this is a computationally expensive doubling of the size of the feature space (see chapter **7**). While the tree-inversion paradigm proposed in section **10.4** is not worked out in enough detail here to predict whether and how a practical or descriptive framework may use it precisely, the way **TAG** generate relative clauses is ready-to-use, and a practical prototype system that recognizes the suggested **HG-TAG** formalism from **E.2** can be constructed quickly, without many problems.

*

THE TREE INVERSION PARADIGM may not be worked out in enough detail here, it does seem to have one consequence on a slightly smaller scale, that cannot be reproduced with the tree adjoining solution. When relative clauses and other long-distance dependencies are not taken into consideration, ENGLISH can be described accurately in a formalism consisting of a bilinear *context-free* backbone with a finite feature space—think of **GPSG** [GKPS85]. If the tree inversion paradigm is added to such a grammar, it will be able to describe a variety of long-distance dependencies, and the underlying non-inverting grammar can, in contrast to the case of **FHG** where the result is a grammar over 4-tuples, be shown to be again context-free (this is due to the bilinearity). This leads to respectable evidence for the claim that the core syntax of English (as opposed to Dutch, German and Latin) is strongly context-free.

*

Underlying derivational and representational formalisms, their computational tractability, and the possibility to give accurate, explanatorily satisfactory grammatical descriptions using the formalisms, are the three major ingredients of a EXECUTABLE DESCRIPTIVE LANGUAGE SYSTEM. By investigating various options for each of these topics I hope to have opened new channels for the design of natural language systems that, following immediately from their theoretical underpinnings, have sufficient explanatory capacities *and* are computationally tractable.

Samenvatting in het Nederlands
Oppervlakte zonder Structuur

Woordvolgorde en computationele uitvoerbaarheid in natuurlijke- taalanalyse

Dit boek is het resultaat van een promotieonderzoek binnen het NWO-project *incrementele ontledergeneratie en context-afhankelijke disambiguatie*, dat de doelstelling heeft een brug te bouwen tussen kennis over en technieken voor de beschrijving van *talen* in Software Engineering en Linguistiek. In beide gebieden hebben talen zogenaamde *niet-contextvrije* elementen, maar de manieren om deze elementen te benaderen zijn in de Informatica en de Linguistiek nogal verschillend. De Linguistiek zou kunnen profiteren van kennis in de informatica over snelle prototypering, het *incrementeel parseren*: ontleden terwijl de ontwerper nog aan de grammatica werkt. De Informatica zou kunnen profiteren van de kennis van linguisten over het werken met *ambiguïteit*. Ambiguïteit is een fenomeen dat in de Informatica vaak op adhoc manieren wordt geëlimineerd om snelle verwerking te garanderen, terwijl deze eliminatiemethoden de *onderhoudbaarheid* van grammatica's in de informatica noemenswaardig aantasten.

Hier wordt het deel van dit project beschreven dat kijkt naar toepassingen op linguistisch gebied. Terwijl het andere deel [Vis97], dat is uitgevoerd door EELCO VISSER, juist een in de linguistiek veel voorkomende techniek bestudeerde, namelijk die van het ontleden aan de hand van een contextvrije grammatica, gevolgd door een fase van *disambiguatie*, is in het hier beschreven deel een bijna omgekeerde beweging te zien. Door, in plaats van gebruikelijke *contextvrije grammatica's*, aan de basis veel sterkere formalismen te gebruiken, wordt de moeilijkheidsgraad van fases die volgen op de structurele analyse (het primaire ontleden van de zin) wezenlijk vereenvoudigd.

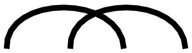
Een CONTEXTVRIJE GRAMMATICA (**CFG**) is een systeem van *herschrijfregels* die aangeven hoe uit een serie kleine zinsdelen een groter zinsdeel kan worden opgebouwd, en wel door de kleinere zinsdelen letterlijk achter elkaar te zetten. Zo'n ononderbroken deel van een zin dat een logische groep woorden vormt wordt in de linguistiek een CONSTITUENT genoemd. Een voorbeeld van een **CFG** is (22) op de volgende pagina.

(22) S → NP VP
VP → VT NP
VP → VI

NP → `Frank`
NP → `Julia`
VT → `kuste`
VI → `sliep`

De eerste regel van deze grammatica zegt dat een S (*sentence*, zin) verkregen wordt door een NP (*noun phrase* of zelfstandig-naamwoordsgroep) en een VP (*verb phrase*, gezegde) achter elkaar te zetten. De tweede regel legt uit dat zo'n VP weer te verkrijgen is door achter elkaar te zetten een VT (transitief werkwoord) en een NP. De derde regel zegt dat een VP ook uit een enkel intransitief werkwoord kan bestaan. De laatste vier regels zijn het *lexicon* van de grammatica. Deze eenvoudige grammatica beschrijft slechts vier zinnen: *Frank sliep*, *Julia sliep*, *Frank kuste Julia* en *Julia kuste Frank*.

Contextvrije grammatica's kunnen alleen zinsdelen kunnen opbouwen door kleinere zinsdelen naast elkaar te zetten. Daarom is het gebruik van **CFG** in de basis van grote linguistische systemen afhankelijk van de aanname dat elementen in een zin die op de één of andere manier 'gerelateerd' zijn (een linguist noemt die dan *afhankelijk*), in de volledige zin ook direct naast elkaar staan. Dit heet ook wel een CONSTITUENT-GEBASEERDE ANALYSE of de eigenschap van PROJECTIVITEIT. Er is een aantal empirische voorbeelden dat de projectiviteit van taal in het algemeen onwaarschijnlijk maakt—één van de meest belangrijke daarvan is het fenomeen van *gekruiste afhankelijkheden* in het Nederlands. Het volgende plaatje (23) is daarvan een voorbeeld. De dikke lijnen geven aan welke woorden 'gerelateerd' zijn zoals boven bedoeld.



(23) ...dat Frank Julia koffie zag drinken

Deze zin is niet projectief, omdat de werkwoordsgroep *koffie drinken* niet ononderbroken terug te vinden is in de hele zin: het woord *zag* staat tussen *koffie* en *drinken*.

*

In de literatuur zijn talloze methoden te vinden om zulke niet-projectiviteit te beschrijven; het meest gebruikelijk is aan te nemen dat elementen die aanvankelijk naast elkaar stonden zijn *verhuisd* naar de plek waar ze werkelijk in de zin voorkomen. De LITERAL MOVEMENT GRAMMARS (**LMG**) die als een rode draad door dit proefschrift lopen, generaliseren een klasse van grammaticale formalismen die op vergelijkbare manieren niet-projectiviteit beschrijven, zij het alle juist *zonder* aan dit idee van verhuizing te appelleren. Het is misschien merkwaardig dat het woord *movement* juist in de naam van zo'n formalisme voorkomt—dit heeft te maken met de observatie, die aan **LMG** ten grondslag ligt, dat het voldoende is om niet hele linguistische structuren, maar slechts 'letterlijke' rijtjes woorden te verhuizen.

Het idee achter **LMG** is eenvoudig, en ligt dichtbij de manier waarop in *logische programmeertalen* zoals **Prolog** tegen contextvrije grammatica's aangekeken wordt. De **Prolog**-vertaling van de contextvrije grammatica (22) is (24); in **LMG**-notatie wordt dit (25).

(24)
```
s(Z) :- np(X), vp(Y), append(X, Y, Z).
vp(Z) :- vt(X), np(Y), append(X, Y, Z).
vp(Z) :- vi(Z).
np([frank]).
np([julia]).
vt([kuste]).
vi([sliep]).
```

(25) S(xy) `:-` NP(x), VP(y).
VP(xy) `:-` VT(x), NP(y).
VP(x) `:-` VI(x).

NP(`Frank`).
NP(`Julia`).
VT(`kuste`).
VI(`sliep`).

Een eenvoudige progressie op deze notatie maakt het mogelijk dat een type constituent zoals S of VP niet uit één, maar uit meerdere rijtjes van woorden bestaat, die dan op verschillende plaatsen in de zin terechtkomen. De grammatica achter (26) is contextvrij en beschrijft de 'taal' die bestaat uit rijtjes van n keer de letter `a`, gevolgd door weer precies n keer de letter `b`, voor ieder willekeurig geheel getal n. De grammatica in (27), die erg lijkt op de vorige, beschrijft rijtjes waar nog n keer de letter `c` achteraan is geplakt. Van deze taal is bekend dat hij niet beschreven kan worden door contextvrije grammatica's. (Het symbool ε staat voor een leeg rijtje).

(26) A(`a`x`b`) `:-` A(x).
A(ε).

(27) A(`a`x`b`, `c`y) `:-` A(x, y).
A(ε, ε).

Op vergelijkbare wijze beschrijft de grammatica achter (28) een fragmentje van het Nederlands waar zin (23) inzit. De grammatica doet dat door een werkwoordsgroep (VP) te laten bestaan uit niet één serie woorden, maar twee groepen, waarvan de eerste bestaat uit een reeks zelfstandig-naamwoordgroepen en de tweede alleen uit werkwoorden.

$$
\text{(28)}\qquad
\begin{array}{lll}
\mathrm{S}(\ldots\,\texttt{dat}\ nmv) & \texttt{:-} & \mathrm{NP}(n),\ \mathrm{VP}(m,v). \\
\mathrm{VP}(nm,vw) & \texttt{:-} & \mathrm{VR}(v),\ \mathrm{NP}(n),\ \mathrm{VP}(m,w). \\
\mathrm{VP}(n,v) & \texttt{:-} & \mathrm{VT}(v),\ \mathrm{NP}(n). \\
 & & \\
\mathrm{NP}(\texttt{Frank}). & & \\
\mathrm{NP}(\texttt{Julia}). & & \\
\mathrm{NP}(\texttt{koffie}). & & \\
 & & \\
\mathrm{VR}(\texttt{zag}). & & \\
\mathrm{VT}(\texttt{drinken}). & &
\end{array}
$$

De vier **LMG**-grammatica's die ik nu heb laten zien zijn equivalent met uit de literatuur bekende grammaticaformalismen, namelijk MULTIPLE CONTEXT-FREE GRAMMARS, ook wel LINEAR CONTEXT-FREE REWRITING SYSTEMS genoemd. De toegevoegde waarde van **LMG** blijkt te liggen in de beter leesbare notatie, waardoor deze grammaticale formalismen geschikter zijn om niet alleen in theoretische zin over taal te praten, maar ook daadwerkelijk concrete grammatica's in te schrijven. Bovendien is de volgende observatie cruciaal: een grammaticaregel van de vorm

$$\text{(29)}\qquad \mathrm{A}(x)\ \texttt{:-}\ \mathrm{B}(x),\mathrm{C}(x).$$

moet worden geïnterpreteerd als: een rijtje woorden x is een A precies dan wanneer x zowel een B als een C is. Dit wordt DOORSNIJDING of INTERSECTIE van talen genoemd, en deze eigenschap hebben de traditionele formalismen uit de literatuur niet.

Globaal overzicht van het proefschrift

De **Proloog** komt grotendeels overeen met deze samenvatting, maar bevat daarnaast een overzicht van wanneer welke delen van dit proefschrift zijn ontstaan en welke mensen daar essentieel aan hebben bijgedragen. De proloog formuleert en passant een lijst van 9 *points of departure* die een groot deel van de motivatie leveren voor de studies die in dit boek worden uitgevoerd. Hoofdstuk **1** is een gedetailleerde inleiding in begrippen als contextvrije grammatica's en projectiviteit. De rest van het boek is opgesplitst in drie delen, die hieronder bondig worden samengevat.

I. Formele structuur

Deel één kijkt op een zeer mathematische, formele manier naar een tweetal groepen van formalismen dat fenomenen van niet-projectiviteit aanpakt, namelijk EXTRAPOSITION GRAMMAR (**XG**) en TUPLE GRAMMARS, waaronder het eerder beschreven **LMG** valt. Hierbij ligt de nadruk op het classificeren van deze merendeels al eerder bestudeerde formalismen en hun eigenschappen in een goed lopend verhaal. Bijzonder in dit deel is dat er veel aandacht wordt besteed aan de mogelijkheid om volledige Nederlandse en Duitse zinnen te beschrijven—een eigenschap die in de literatuur vaak op zeer beperkte wijze wordt onderzocht, omdat men alleen kijkt naar ondergeschikte bijzinnen zoals (23), terwijl declaratieve en interrogatieve vormen zoals (30) en (31) nog meer verhuizing of niet-projectiviteit introduceren en dus meer eisen lijken te stellen voordat men kan zeggen dat de gebruikte grammaticaformalismen deze op een verantwoorde manier beschrijven.

(30) Jan zag Marie koffie drinken

(31) Zag Jan Marie koffie drinken?

II. Computationele uitvoerbaarheid

Het tweede deel houdt zich bezig met de praktische bruikbaarheid van de formalismen uit deel **I** op computers. Dit boek concentreert zich hierbij op de opgave om, gegeven een grammatica, een willekeurige zin te *ontleden*. Er wordt niet gekeken naar de het probleem van het *genereren* van zinnen.

Allereerst wordt voor varianten van **LMG** (hoofdstuk **5**) en **XG** (hoofdstuk **6**) bewezen dat ze in theoretische zin efficiënt uitvoerbaar zijn op computers. Hiervoor kijkt men naar zeer abstracte modellen van computers (zogenaamde RANDOM ACCESS en TURING-machines), en bewijst dat wanneer men de lengte van de ingevoerde zin laat groeien, de tijd die een ontleedprogramma nodig heeft om de zin te ontleden niet te drastisch groeit—dit noemt men COMPLEXITEITSANALYSE. Onder drastische groei verstaat men exponentiële groei, dat wil zeggen dat wanneer een zin 1 woord langer wordt, het programma twee keer zo veel tijd in beslag zal nemen; acceptabele groei is bijvoorbeeld wanneer de looptijd van de ontleder op zijn hoogst evenredig is met het kwadraat, of een willekeurige andere macht, van de lengte van de invoer.

Tenslotte wordt er in hoofdstuk **7** gekeken naar meer realistische uitvoeringen van zulke programmatuur, waarbij bovendien rekening wordt gehouden met het feit dat na een structurele, syntactische analyse van een zin vaak nog eigenschappen als naamvallen en enkelvoud/meervoud moeten worden gecontroleerd, en tenslotte ook iets met de ontlede zin gedaan moet worden. Er wordt onder andere beargumenteerd dat efficiëntie in de praktijk wel dicht bij de eerder besproken theoretische variant ligt, maar dat ook factoren en ontwerpoverwegingen een rol spelen die door theoretici als betekenisloos worden beschouwd.

III. Principes

In deel drie wordt een aantal pogingen gedaan om de intuïties ontwikkeld in delen **I** en **II** op een linguistisch acceptabele manier te formaliseren. Dit gebeurt op drie manieren.

In hoofdstuk **8** wordt gekeken naar macro-eigenschappen van talen, zoals onder welke voorwaarden delen van een zin zijn te herhalen om een nieuwe zin te vormen—denk hierbij aan *Jan zei dat Jan zei dat ... Jan zei dat het een warme dag was*—en wat, volgend uit deze analyse, de precieze klasse van talen moet zijn die in structurele zin overeenkomt met het begrip 'natuurlijke taal'.

In hoofdstukken **9** en **10** worden zogenaamde *axiomatische* systemen opgezet. Het eerste is een zeer beknopte vorm van CHOMSKY'S GOVERNMENT-BINDING THEORY, waarvan vervolgens een variant wordt geformuleerd die in **LMG** kan worden uitgedrukt en daarmee efficiënt kan worden uitgevoerd op computers. Het tweede systeem, FREE HEAD GRAMMAR, is een poging tot een axiomatisering te komen die meer direkt op woordvolgorde is georiënteerd, en daarmee een empirische onderbouwing geeft voor de manier waarop **LMG** tegen linguistische structuur aankijkt. In dit laatste hoofdstuk wordt ook wat uitgebreider ingegaan op de concrete beschrijving van stukken Nederlands en Latijn.

*

In de **Epiloog** tenslotte worden de formalismen behandeld in de delen **I**, **II** en **III** vergeleken met een aantal voor de hand liggende andere onderzoeksgebieden, namelijk CATEGORIALE GRAMMATICA en TREE ADJOINING GRAMMAR. Tenslotte worden de conclusies uit de verschillende delen en de **Epiloog** met elkaar in verband gebracht door een puntgewijs resumé van de behaalde resultaten.

Curriculum Vitae

Annius Victor Groenink werd geboren op 29 december 1971 in Twello. Van 1975 tot 1983 kreeg hij basisonderwijs op de Deventer Montessorischool. In hetzelfde Deventer ging hij van 1983 tot 1989 naar de openbare scholengemeenschap Alexander Hegius, waar hij in 1989 een diploma Gymnasium β behaalde, dat bij gebrek aan een zogenaamd 'maatschappelijk' vak niet ook als Gymnasium α mocht tellen. In 1989 meldde hij zich aan bij de studierichting Wiskunde van de Rijksuniversiteit Utrecht. Na de propaedeutische examens Wiskunde en Informatica af te ronden in 1990 resp. 1991, studeerde hij in het academisch jaar 1991–1992 in Leeds, Yorkshire, waar hij een deel van de M.Sc.-examens aflegde. In de laatste drie maanden van 1992 deed hij een stage van drie maanden bij PTT-Research in Leidschendam, waar hij werkte aan de semantische disambiguatiefase in een prototype van een natuurlijke-taalinterface voor databasesystemen. In 1993 kreeg hij een doctoraaldiploma Wiskunde voor de scriptie *Uism and short-sighted models* die facetten van het zogenaamde *strict-finitisme* bestudeerde. Sinds 1990 werkt hij, naast zijn wetenschappelijke bezigheden, aan de ontwikkeling van een teksteditorpakket dat hij vanaf 1993 binnen het kader van een eenmansbedrijf distribueert. Het onderzoek dat heeft geleid tot deze dissertatie voerde hij tussen 1993 en 1997 uit aan het Centrum voor Wiskunde en Informatica in Amsterdam.

Bibliography

[BBR87] Edward G. Barton, Robert C. Berwick, and Eric Sven Ristad. *Computational Complexity and Natural Language*. MIT Press, 1987.

[BH53] Y. Bar-Hillel. A Quasi-Arithmetical Notation for Syntactic Description. *Language*, 29:47–58, 1953.

[BHK89] J. A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification*. ACM Press Frontier Series. The ACM Press in co-operation with Addison-Wesley, 1989.

[BKPZ82] Joan Bresnan, Ronald M. Kaplan, Stanley Peters, and Annie Zaenen. Cross-serial Dependencies in Dutch. *Linguistic Inquiry*, 13(4), 1982.

[BvN95] Gosse Bouma and Gertjan van Noord. A lexicalist account of the dutch verbal complex. In *Papers from the Fourth CLIN Meeting, Rijksuniversiteit Groningen*, 1995.

[CH] Crit Cremers and Maarten Hijzelendoorn. Counting coordination categorially. Available electronically from `xxx.lanl.gov` as `cmp-lg/9605011`.

[Cho96] Noam Chomsky. *The Minimalist Program*. Cambridge, MA: MIT Press, 1996.

[CKS81] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *JACM*, 28:114–133, 1981.

[dBR89] Hans den Besten and Jean Rutten. On Verb Raising, Extraposition and Free Word Order in Dutch. In *Sentential complementation and the lexicon*, pages 41–56. Foris, 1989.

[Ear70] Jay Earley. *An Efficient Context-Free Parsing Algorithm*. PhD thesis, Dept. of Computer Science, Carnegie-Mellon University, 1970.

[Eve75] A. Evers. *The Transformational Cycle in Dutch and German*. PhD thesis, Rijksuniversiteit Utrecht, 1975.

[GGH69] Seymour Ginsburg, Sheila Greibach, and John Hopcroft. *Studies in Abstract Families of Languages*. Providence, Rhode Island: Memoirs of the American Mathematical Society number 87, 1969.

[Gin75] Seymour Ginsburg. *Formal Languages*. North-Holland/Am. Elsevier, Amsterdam, Oxford/New York, 1975.

[GKPS85] Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. *Generalized Phrase Structure Grammar*. Oxford: Basil Blackwell; Cambridge, Mass.: Harvard University Press, 1985.

[Gro95a] Annius V. Groenink. Formal Mechanisms for Left Extraposition in Dutch. In *Proceedings of the 5th CLIN (Computational Linguistics In the Netherlands) meeting, November 1994, Enschede*, November 1995.

[Gro95b] Annius V. Groenink. Literal Movement Grammars. In *Proceedings of the 7th EACL Conference, University College, Dublin*, March 1995.

[Gro96] Annius V. Groenink. A Unifying Framework for Concatenation-based Grammar Formalisms. In *Proceedings of Accolade 1995, Amsterdam*, 1996.

[Gro98] Annius V. Groenink. Mild Context-Sensitivity and Tuple-based Generalizations of Context-Free Grammar. To appear in the special MOL4 issue of *Linguistics and Philosophy*; available electronically from `www.cwi.nl` as report CS-R9634, 1998.

[Hae91] Liliane Haegeman. *Introduction to government & binding theory*. Basil Blackwell, 1991. Second edition, 1994.

[Hay64] David G. Hays. Dependency theory: a formalism and some observations. *Language*, 40(4):511–525, 1964.

[HS74] J. Hartmanis and J. Simon. On the power of multiplication in random access machines. In *Proceedings of the 15th IEEE symposium on switching and automata theory, New Orleans, LA*, pages 13–23, 1974.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass., 1979.

[Hud] Richard Hudson. Discontinuity. 1996, draft, available electronically by `ftp` from `pitch.phon.ucl.ac.uk` as `/pub/Word-Grammar/disco-ps.zip`.

[Hud90] Richard Hudson. *English Word Grammar*. Oxford: Blackwell, 1990.

[Hui98] Willem-Olaf Huijsen. *Completeness of Compositional Translation*. PhD thesis, Universiteit Utrecht, 1998. Forthcoming.

[Huy76] Riny Huybrechts. Overlapping dependencies in Dutch. *Utrecht working papers in Linguistics*, 1:24–65, 1976.

[Huy84] Riny Huybrechts. The weak inadequacy of context-free phrase structure grammars. In De Haan, Trommelen, and Zonneveld, editors, *Van periferie naar kern*. Foris, 1984.

[Jan86] Theo M. V. Janssen. *Foundations and applications of Montague grammar*. CWI Tract 28. Centrum voor Wiskunde en Informatica, Amsterdam, 1986.

[Jos85] Aravind Joshi. Tree Adjoining Grammars: How much Context-Sensitivity is Required to Provide Reasonable Structural Descriptions? In A. Zwicky, editor, *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, pages 206–250. Cambridge University Press, 1985.

[Kli93] Paul Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering Methodology*, 2(2):176–201, 1993.

[KNSK92] Y. Kaji, R. Nakanishi, H. Seki, and T. Kasami. The Universal Recognition Problems for Parallel Multiple Context-Free Grammars and for Their Subclasses. *IEICE*, E75-D(4):499–508, 1992.

[Kos91] C. H. A. Koster. *Affix Grammars for natural languages*, pages 358–373. Spinger lecture notes in Computer Science no. 545, 1991.

[Kra95] Marcus Kracht. Syntactic Codes and Grammar Refinement. *Journal of Logic, Language, and Information*, 4(4):359–380 (41–60), 1995.

[Lam58] J. Lambek. The Mathematics of Sentence Structure. *American Mathematics Monthly*, 65:154–169, 1958.

[Lee93] René Leermakers. *The Functional Treatment of Parsing*. Kluwer, The Netherlands, 1993.

[McA93] David A. McAllester. Automatic Recognition of Tractability in Inference Relations. *JACM*, 40(2), 1993.

[Mel88] Igor' Mel'čuk. *Dependency syntax: theory and practice*. SUNY series in Linguistics, Albany, NY, 1988.

[MK96] Jens Michaelis and Marcus Kracht. Semilinearity as a Syntactic Invariant. In *Proceedings of the Logical Aspects of Computanional Linguistics meeting, Nancy, 23–25 September, 1996*. To appear in the Springer lecture notes series, 1996.

[Moo88] Michael Moortgat. *Categorial Investigations. Logical and Linguistic Aspects of the Lambek Calculus*. PhD thesis, Universiteit van Amsterdam, 1988.

[Mor94] Glyn V. Morill. *Type Logical Grammar*. Dordrecht: Kluwer, 1994.

[MR87] Alexis Manaster-Ramer. Dutch as a formal language. *Linguistics and Philosophy*, 10:221–246, 1987.

[Nic78] Nichols. Double dependency? *Proceedings of the Chicago Linguistic Society*, 14:326–339, 1978.

[NK92] M. J. Nederhof and C. H. A. Koster. A grammar workbench for AGFLs. In J. Aarts, P. de Haan, and N. Oostdijk, editors, *Proc. of the Thirteenth ICAME Conference, Nijmegen*. Rodopi, 1992.

[NS93] M. J. Nederhof and J. J. Sarbo. Efficient Decoration of Parse Forests. In H. Trost, editor, *Feature Formalisms and Linguistic Ambiguity*, pages 53–78. Ellis Horwood, 1993. Short, earlier version in the proceedings of the workshop *Coping with Linguistic Ambiguity in typed Feature Formalisms* at the 10th European Conference on Artificial Intelligence (ECAI), Vienna, Austria, August 1992.

[Pen] Mati Pentus. Lambek Grammars are Context-Free. Manuscript Dept. of Mathematics, Univ. of Moscow, 1992.

[Per81] Fernando C. N. Pereira. Extraposition Grammars. *Computational Linguistics*, 7(4):243–256, 1981.

[Pin90] Harm Pinkster. *Latin syntax and semantics*. London, New York: Routledge., 1990. Original title: Latijnse syntaxis and semantiek; Gruener, Amsterdam, 1984. Also Tübingen: Francke, 1988; Madrid: Ediciones Clasicas, 1995; Torino: Rosenberg & Sellier, 1991.

[Pol84] Carl J. Pollard. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*. PhD thesis, Stanford University, 1984.

[PS94] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. Univ. of Chicago Press, 1994.

[PW83] Fernando C. N. Pereira and David H. D. Warren. Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linsuistics (ACL21)*, pages 137–144. Cambridge, MA: MIT press, 1983.

[Rad91] Daniel Radzinski. Chinese Number-Names, Tree Adjoining Languages, and Mild Context-Sensitivity. *Computational Linguistics*, 17(3):277–299, 1991.

[Ram94] Owen Rambow. *Formal and Computational Aspects of Natural Language Syntax*. PhD thesis, University of Pennsylvania (Philadelphia), 1994.

[Rea90] Mike Reape. Getting Things in Order. In *Proceedings of the Symposium on Discontinuous Constituency*. Tilburg: ITK, 1990.

[Rek92] Jan Rekers. *Parser Generation for Interactive Environments*. PhD thesis, University of Amsterdam, 1992.

[Ris90] Eric Sven Ristad. *Computational Structure of Human Language*. PhD thesis, MIT dept. of Electrical Engineering, 1990. Available as MIT technical report AI-TR 1260.

[RJ94] Owen Rambow and Aravind Joshi. A Formal Look at Dependency Grammars and Phrase-Structure Grammars, with Special Consideration of Word-Order Phenomena. In Leo Wanner, editor, *Current Issues in Meaning-Text Theory*. Pinter, London, 1994.

[Roa87] K. Roach. Formal Properties of Head Grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 392–347. Amsterdam: John Benjamins, 1987.

[Ros67] John R. Ross. *Constraints on Variables in Syntax*. PhD thesis, Harvard University, 1967. Reprinted by the Indiana Linguistics Club.

[Rou88] William C. Rounds. LFP: A Logic for Linguistic Descriptions and an Analysis of its Complexity. *Computational Linguistics*, 14(4):1–9, 1988.

[Rou97] William C. Rounds. Feature Logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, chapter 8, pages 475–533. Elsevier Science, 1997.

[Sei93] Roland Seiffert. What Could a Good System for Practical NLP Look Like? In H. Trost, editor, *Feature Formalisms and Linguistic Ambiguity*, pages 187–204. Ellis Horwood, 1993.

[Sel85] Peter Sells. *Lectures on Contemporary Syntactic Theories: an Introduction to Government-Binding theory, Generalized Phrase Structure Grammar, and Lexical-Functional Grammar*. Stanford: CSLI lecture notes, 1985. Second edition: 1987.

[Shi86] Stuart M. Shieber. *An Introduction to Unification-Based Approaches to Grammar*. CSLI Lecure notes no. 4, Standord., 1986.

[SMFK91] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229, 1991.

[Sta87] Edward P. Stabler. Restricting Logic Grammars with Government-Binding Theory. *Computational Linguistics*, 13(1):1–10, 1987.

[Tom86] Masaru Tomita. *Efficient parsing for natural language: a fast algorithm for practical systems*. Dordrecht: Kluwer, 1986.

[Tom91] Masaru Tomita, ed. *Generalized LR parsing*. Dordrecht: Kluwer, 1991. selected papers on GLR parsing, presented at the first international workshop on parsing technologies, Pittsburgh.

[Tur90] D. A. Turner. An Overview of Miranda. In Turner, editor, *Research Topics in Functional Programming*, pages 1–16. Reading, Massachusetts: Addison-Wesley, 1990.

[Ver96] Koen Versmissen. *Grammatical Composition: Modes, Models, Modalities*. PhD thesis, Universiteit Utrecht, 1996.

[Vis97] Eelco Visser. *Syntax Definition for Language Prototyping*. PhD thesis, Universiteit van Amsterdam, 1997.

[vN93] Gertjan van Noord. *Reversibility in Natural Language*. PhD thesis, Rijksuniversiteit Groningen, 1993.

[VSW94] K. Vijay-Shanker and D. J. Weir. The Equivalence of Four Extensions of Context-Free Grammar. *Math. Systems Theory*, 27:511–546, 1994.

[VSWJ87] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. On the progression from context-free to tree adjoining languages. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 389–401. Amsterdam: John Benjamins, 1987.

[Wei88] David J. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania, 1988.

[Wei92] David J. Weir. A geometric hierarchy beyond the context-free languages. *Theoretical Computer Science*, 104:235–261, 1992.

[WK96] H. R. Walters and J. F. Th. Kamperman. Epic 1.0 (unconditional), an equational programming language. Technical Report CS-R9604, CWI, january 1996. Available electronically as `http://www.cwi.nl/epic/articles/epic10.ps`.

[WVSJ86] David J. Weir, K. Vijay-Shanker, and A. K. Joshi. The Relationship Between Head Grammars and Tree Adjoining Grammars. In *Proceedings of the 24th Annual Meeting of the ACL*, 1986.

[WVSR95] David J. Weir, K. Vijay-Shanker, and Owen Rambow. D-tree grammars. In *ACL-95*, 1995.

[You67] Daniel H. Younger. Recognition and Parsing of Context-Free Languages in Time n^3. *Information and Control*, 10:189–208, 1967.