Improving the feasibility of precision-oriented HPSG parsing

Dissertation
zur Erlangung des akademischen Grades eines
Doktors der Philosophie
der Philosophischen Fakultäten
der Universität des Saarlandes

vorgelegt von

Bart Cramer

aus Stadskanaal, den Niederlanden

Saarbrücken, 2011

Abstract

This thesis will focus on the feasibility of precision-oriented parsing in the framework of Head-driven Phrase Structure Grammar (Pollard and Sag 1994). Such parsers, traditionally based on hand-written grammars, offer detailed semantic analyses of the language. However, there are a number of barriers that need to be overcome before such a parser can be successfully deployed, most notably the grammar's long development time. Statistical parsers are less prone to this issue, but do not offer the same depth of analysis that hand-written deep grammars can. A number of approaches (in different linguistic formalisms, often highly lexicalised) have been proposed that aim to combine the advantages of both types of parsers, usually by converting/enriching an existing treebank to a deeper linguistic formalism, after which a deep grammar can be learnt from the richer resource of annotated data. This thesis has a comparable aim, but approaches the problem from the perspective of precision-oriented parsing, automating as much as possible, and crafting by hand what is necessary, for instance because it is not learnable from the available resources. The German language is taken as object of study.

A full-fledged deep grammar (in the DELPH-IN toolchain, in which the research is embedded) minimally consists of the following components: a set of constructions, a lexicon, a morphological analyser, a treebank and a disambiguation model based on that treebank. All these components will be created in the first half of the thesis, trying to minimise the effort that is needed for the creation of each component. It is argued that HPSG constructions are too complicated to learn, and are therefore hand-written. Augmented with a small lexicon of syntactically or semantically idiosyncratic lexemes, this forms the core grammar. Naturally, the core grammar is based on available HPSG analyses of German (and Dutch) in the literature, of which an overview will be given. One of the contributions of the specific core grammar in this thesis is the novel treatment of word order and topological fields, based on an implementation of FSAs in the typed feature structure formalism that is used in the DELPH-IN framework.

Consequently, the lexicon is constructed automatically from a detailed dependency treebank (the Tiger treebank: Brants *et al.* 2002), in a deep lexical acquisition step. The syntactic properties of the lexical entries, such as subcategorisation frames and modification constraints, are recognised on the basis of the dependency labels that define the relation between constituents. Additionally, a partial mapping between word forms and lexemes is learnt, which functions as the grammar's morphological analyser.

A link is made between the output of the grammar and the dependencies that can be derived from the Tiger treebank. This entails that the normal output of

a DELPH-IN grammar (Minimal Recursion Semantics: Copestake *et al.* 2005) will not be used. Instead, the grammar and Tiger treebank are interfaced by syntactic dependencies. A novel way to test the chain of core grammar, deep lexical acquisition and MRS conversion is introduced (unit testing), allowing the grammar writer to track the influence of a change in any of the components has on the correctness of the grammar. Furthermore, the link between the output of the grammar and the Tiger treebank makes an automatic disambiguation between licensed readings possible, allowing the automatic creation of an HPSG treebank.

A held-out part of the gold standard is used as an evaluation set, showing the performance of the parser (the combination of a parsing algorithm and the grammar) on unseen text. The performance is measured across multiple dimensions, such as development time, linguistic relevance and coverage, and is the background to a larger discussion, in which the grammar is situated in and compared to an array of hand-written and learnt parsers. A number of areas for improvement were found, and two of them were addressed afterwards: (lack of) efficiency and robustness. In these experiments, the grammar was left mostly unchanged, and the PET parser, that takes the grammar as input, was altered.

In its standard setting, the PET parser executes all parser tasks (unifications). The agenda of parser tasks was changed in such a way that more promising tasks are carried out first. Less promising tasks are deferred or even discarded. A generative model of HPSG rule applications is at the basis of determining the relative order of the tasks' priorities. A number of strategies were introduced to decide which tasks are pruned from the agenda. Good results were achieved using this technology, showing both an increase of accuracy and a manifold speed-up.

The relative fragility of precision-oriented parsers is the second aspect of the parser that was improved. A common approach to find an analysis for an unlicensed sentence is to return a set of recognised fragments. In our experiments, a variant of this fragment parsing strategy functioned as the baseline. A new method was introduced, in which highly over-generating robustness rules were created by the grammar writer. The added rules are meant to take up unrecognised objects, minimising the damage they cause, and are part of the normal parsing process, although highly dispreferred by the statistical models. The use of robustness rules yields better solutions than the fragment parsing approach, but does not cover the entire test set. A combination of both strategies, therefore, resulted in higher f-scores than fragment parsing, retaining full coverage.

Deutsche Zusammenfassung

Das Thema dieser Arbeit ist die Umsetzbarkeit von Präzisions-Orientiertem Parsing im Formalismus der Kopf-Gesteuerten Phrasenstrukturgrammatik (*Head-driven Phrase Structure Grammar*, *HPSG*) (Pollard and Sag 1994). Diese Parser, die traditionell auf handgeschriebenen Grammatiken basieren, ermöglichen detaillierte semantische Analysen einer Sprache. Es gibt jedoch eine Reihe von Schwierigkeiten, die überwunden werden müssen, bevor ein solcher Parser erfolgreich angewendet werden kann, insbesondere die langen Entwicklungszeiten der Grammatik. Statistische Parser sind weniger anfällig für dieses Problem, jedoch bieten sie nicht dieselbe Analysetiefe wie handgeschriebene Grammatiken. Es wurde bislang eine Reihe von Ansätzen (in verschiedenen, oft hoch lexikalisierten, linguistischen Formalismen) vorgeschlagen, welche darauf abzielen, die Vorteile beider Parsertypen zu kombinieren. Diese Kombination geschieht meist durch das Konvertieren von Baumbänken zu einer tieferen linguistischen Ebene, die es ermöglicht, dass auf Grundlage der reichen Ressource an annotierten Daten eine Tiefengrammatik gelernt werden kann. Diese Arbeit verfolgt eine vergleichbare Intention, nähert sich dem Problem aber von der Seite des Präzisions-Orientiertem Parsings. Hierbei wird so viel wie möglich automatisiert, wenn nötig, wird jedoch von Hand erstellt, beispielsweise etwas, was nicht von verfügbaren Ressourcen gelernt werden kann. Das Deutsche wird hierbei als Test-Sprache verwendet.

Eine vollständige Tiefengrammatik (in der DELPH-IN Toolchain, in welche diese Forschung eingebettet ist) besteht mindestens aus den folgenden Komponenten: eine Menge von Konstruktionen, ein Lexikon, ein morphologischer Analysierer, eine Baumbank und ein Disambiguierungs-Modell, das auf der Baumbank basiert. Diese Komponenten werden in der ersten Hälfte dieser Arbeit erzeugt, mit dem Ziel, den Aufwand für die Erstellung jeder einzelnen Komponente zu minimieren. Es wird argumentiert, dass HPSG-Konstruktionen zu kompliziert sind um gelernt zu werden und daher handgeschrieben werden müssen. Dies bildet unter Zufügung eines kleines Lexikons von syntaktisch- und semantisch-idiosynkratischen Lexemen den Kern der Grammatik. Die Kern-Grammatik basiert auf HPSG-Analysen des Deutschen (und Niederländischen), die in der Literatur verfügbar sind. Ein überblick dieser Analysen wird erfolgen. Einer der Beiträge, den die Kern-Grammatik, die spezifisch in dieser Arbeit entwickelt wird, leistet, ist die neuartige Behandlung von Wortreihenfolge und topologischen Feldern. Diese Behandlung basiert auf der Implementierung von FSAs in der TDL-Syntax, die im DELPH-IN-Formalismus verwendet wird.

Konsequenterweise wird das Lexikon automatisch von einer detaillierten Dependenz-Baumbank (der Tiger-Baumbank) durch die Methode des tiefen le-

xikalischen Erwerbs konstruiert. Die Erkennung syntaktischer Eigenschaften der lexikalischen Einträge, wie zum Beispiel Subkategorisierungsrahmen und Modifikations-Constraints, geschieht basierend auf Dependenz-Bezeichnungen, die das Verhältnis zwischen Konstituenten definieren. Zusätzlich wird ein partielles Mapping zwischen Wortformen und Lexemen gelernt, die als der morphologische Analysierer der Grammatik fungieren.

Es wird überdies eine Verbindung hergestellt zwischen dem Output der Grammatik einerseits und den Dependenzen, die aus der Tiger-Baumbank abgeleitet werden können, andererseits. Dies bedingt, dass der normale Output einer DELPH-IN-Grammatik (Minimale Rekursions-Semantik: Copestake *et al.* 2005) nicht benutzt wird. Stattdessen werden die Grammatik und die Tiger-Baumbank mit syntaktischen Dependenzen verbunden. Eine neuartiger Weise, die Kette der Kern-Grammatik zu testen, basierend auf tiefem lexikalischem Erwerb und MRS-Konversion, wird eingeführt werden (Unit Testing). Diese Art des Testens erlaubt es dem Grammatik-Schreiber, zu verfolgen, welchen Einfluss Veränderungen aller einzelnen Komponente auf die Korrektheit der Grammatik haben. Überdies macht eine Verbindung zwischen dem Output der Grammatik und der Tiger Baumbank eine automatische Disambiguierung zwischen lizensierten Lesarten/Interpretationen möglich, was die automatische Erstellung einer HPSG-Baumbank erlaubt.

Ein zurückgehaltener Teil des Gold-Standards wird als Evaluations-Menge genutzt um die Performanz des Parsers (die Kombination eines Parsing-Algorithmus und der Grammatik) für unbekannten Text zu prüfen. Die Leistung wird an Hand multipler Dimensionen, so wie der Entwicklungszeit, der linguistischen Relevanz und der Abdeckung, gemessen. Diese Leistung ist Grundlage für eine weitere Diskussion, in welchem Rahmen die Grammatik betrachtet und mit einem Array handgeschriebener und gelernter Parser verglichen wird. Eine Reihe von Bereichen, die verbessert werden können, sind gefunden worden, von denen zwei nachträglich behandelt wurden: (Fehlen von) Effizienz und Robustheit. In diesen letzteren Experimenten blieb die Grammatik unverändert und der PET-Parser, der dir Grammatik als Input nimmt, wurde abgewandelt.

In seinem Standard-Setting führt der PET-Parser alle Parser-Aufgaben (Unifikationen) aus. Die Agenda der Parser-Aufgaben wurde in dieser Arbeit nun verändert, indem stärker vielversprechende Aufgaben zuerst ausgeführt wurden. Weniger vielversprechende Aufgaben wurden aufgeschoben oder sogar verworfen. Ein generatives Modell von HPSG-Regel-Applikationen bestimmte die relative Reihenfolge der Aufgaben-Prioritäten. Eine Reihe von Strategien wurden eingeführt, um zu entscheiden, welche Aufgaben von der Agenda gekürzt werden. Gute Ergebnisse wurden durch diese Technologie erreicht: sowohl ein Anstieg an Genauigkeit, als auch eine facettenreiche Beschleunigung.

Die relative Fragilität Präzisions-Orientierter Parser ist der zweite Aspekt

des verbesserten Parsers. Eine übliche Herangehensweise, um eine Analyse für einen nicht lizensierten Satz zu finden, ist es, eine Menge nicht erkannter Fragmente zurück zu geben. In den Experimenten fungiert eine Variante dieser Fragment-Parsing-Strategie als Baseline. Eine neue Methode wurde eingeführt, im Rahmen derer stark übergenerierende Robustheits-Regeln vom Grammatik-Schreiber erstellt werden. Die hinzugefügten Regeln verfolgen den Sinn, nicht erkannte Objekte aufzunehmen, wodurch der Schaden, den sie anrichten, minimiert wird. Diese Regeln sind Teil des normalen Parsing-Prozesses, obwohl sie von dem statistischem Modell stark dispräferiert werden. Der Gebrauch der Robustheits-Regeln bringt bessere Lösungen als der Fragment-Parsing-Ansatz, aber er deckt nicht das vollständige Test-Set ab. Eine Kombination beider Strategien resultierte daher in höheren F-Scores als Fragment-Parsing, unter Bewahrung vollständiger Abdeckung.

Acknowledgements

By definition, the acknowledgements section in a thesis is incomplete. Not only is the list of people that I would like to thank not exhaustive, I do not do enough justice to the persons who have contributed to the work described in this booklet, just by putting a sequence of characters here. I hope I have expressed my gratitude for your valuable input often and strongly enough, rendering this piece of text a formality.

I would like to thank Hans, for his courage to take me on as his student, and for the meaningful meetings we had together. I am also grateful to Stephan, who has been inspiring and supportive when we discussed experiments, and conscientious (and strict, when it was necessary!) while I was writing. Yi has been a great mentor (may I say: pseudo-supervisor?) from the beginning, giving me a hand when technical problems arose, and helping me shape my research.

The DELPH-IN community has been a great forum, where I have been able to develop my skills, thanks to the discussions we had, and the constructive feedback you offered, for which I am very grateful.

I am also greatly indebted to Antske, Micha, Rebecca, Rui, Tania and Valia, with whom I had nice and vivid discussions during our group meetings.

The financial means to complete my PhD was provided by the DFG (Deutsche Forschungsgemeinschaft) in the form of a PIRE scholarship, which allowed me to do research, write a thesis and travel to conferences and research meetings. I would like to thank Mark and Eugene for being my sponsors during my stay at Brown University and all people who have made my stay in Providence such a wonderful experience.

The thesis also marks the end of my time in Saarbrücken. It has been a

great time, during which I have met many fun and interesting people. I will not name any of them, as any mention would bring up question marks, but saying y'all have been a great support for me is the least I can do. My gratitude also goes to the staff in cafés (Ubu Roi, Ausländercafé, White Electric, Crunch Café and Coffee Company), who have been serving coffee with a smile while I was working in front of my laptop.

Last but absolutely not least, I would like to thank my parents and Anneke, who have been listening to my complaints during these years, and who have provided me the love and support that any human being needs.

Contents

1 Introduction

Parsing using hand-written, deep computational grammars is sometimes conceived as labour-intensive, inflexible, slow, fragile and hard to evaluate. These parsers are then contrasted with parsers that are learnt from annotated resources, such as treebanks. Learnt parsers are deeply rooted in a belief in statistics as the ordering principle of parsers, and are said to be easy to create, flexible, fast, robust and easy to evaluate. However, hand-written grammars have their merits, too. Explicit coding (and testing) of linguistic hypotheses can offer insights into the language at hand. Moreover, the output that these parsers provide can be more informative or offer a larger degree of abstraction.

It is the tension between the attitudes of these two schools in current parsing research that is at the heart of this thesis. All experiments are nested in the framework of the DELPH-IN collaboration effort[1], which has been particularly focused at exploiting the advantages of hand-written grammars, but which also has been subjected to the allegations listed above. After scrutinising these allegations, the experiments that are presented in the subsequent chapters are meant to address and potentially overcome them.

From a technical perspective, the DELPH-IN framework is a collection of linguistic resources (grammars, treebanks) and tools to make best use of these resources, embedded in Head-driven Phrase Structure Grammar (Pollard and Sag 1994), a heavily lexicalised, non-derivational linguistic theory. A wide array of mature and less mature grammars reside in this framework, mostly hand-written. Two analysis tools are used. First, the Linguistic Knowledge Base (Copestake 2002), a parser/generator written in the Lisp language, is used for development of the grammar, offering a number of visualisation possibilities. The PET parser (Callmeier 2000) (written in C/C++), with a stronger focus on efficiency, is used for large-scale parsing experiments. Both systems accept the same kind of grammars as input (grammars written in a subset of the $\mathcal{TDL}$ framework). Among other possibilities, the `[incr tsdb()]` system (Oepen and Carroll 2000b) is used for treebanking purposes, making optimal use of discriminants to speed up the process.

Creating a grammar by hand (after introspection) can be a costly task. As a rule of thumb, one could say that it takes at least 10 person years to write a grammar with a satisfactory balance between coverage, precision and efficiency. But even given the effort that has been put into the grammar, there will still be linguistic phenomena that are complicated to analyse properly using rule-based methods only. Sentences displaying an uncovered phenomenon are likely to

[1] DELPH-IN is an acronym for DEep Linguistic Processing with HPSG. More information can be found at: `http://www.delph-in.net`

receive no analysis at all, rendering the toolchain fragile when it is applied to realistic input. Moreover, linguistic descriptions of sentences in the DELPH-IN framework (typed feature structures) can be large, and thus computationally expensive, compared to simple category or dependency labels. This entails that in an equally long time span, fewer chart items can be built in order to find the preferred analysis.

Shallow, statistical parsers, learnt from treebanks, are easier to create, but do not provide the fine-grained analyses that parsers based on hand-written grammars can. Several attempts have been made to cross the bridge between shallow and deep parsing, combining the virtues of both approaches, by learning a grammar from a more detailed treebank. Such treebanks were not available traditionally, and therefore had to be created by enriching existing treebanks using heuristic methods. Parsers using these methods (for which I coin the term *deep grammar extraction*: DGE) have provided competitive results on existing benchmark tests. However, the extra information that these enriched treebanks contain is added in a somewhat *ad hoc* manner, leading to linguistically unattractive solutions, especially for less configurational languages, such as German.

1.1 Thesis outline

The thesis is divided in two parts. In the first part, a deep grammar of German will be constructed within the DELPH-IN toolchain, along with a treebank. The grammar is a combination of a hand-written core grammar (including a small core lexicon) and a large lexicon that is automatically derived from the Tiger treebank. In an Interlude, the separation between the two parts of this thesis, the grammar is positioned in the landscape of available parsers. The second part of the thesis is concerned with the PET parser, to address two issues that are often related with hand-written deep parsers: lack of efficiency and lack of robustness.

Chapter 2: Background The next chapter introduces many of the concepts that are used in the thesis. After a basic introduction of the process of parsing in general, a number of state-of-the-art hand-written grammars will be discussed. Then, an overview of a number of treebanks is given, including methods that are meant to speed up the process of treebank creation, for instance by employing a grammar to generate possible analyses. Deep grammars based on treebanks are introduced, and the advantages and disadvantages of DGE grammars and their hand-written counterparts are weighed. Practically all deep parsers use some computational techniques to increase the efficiency and robustness of the parser, and a number of them will be introduced in this chapter. Because all experiments are embedded in

the DELPH-IN framework, a technical introduction is given into the format and functionality of its two main components: the grammar and the PET parser. The chapter closes with the motivation for the research that will be presented in the following chapters.

Chapter 3: Core grammar The first component of the grammar that is built is a hand-written *core grammar*. The goal of the core grammar, which includes a small core lexicon with syntactically idiosyncratic lexemes, is to model those parts of a language that are hard to acquire from a treebank and are largely dependent on the grammar writer's view on the language. The chapter is divided in three parts: analysis, design and implementation. First, the reader is presented with a number of basic properties of the German language, such as its (moderately free) word order and the topological fields. Then, literature regarding the analysis of German within HPSG is discussed, and a number of analyses for common linguistic phenomena, such as constituent permutation and extraposition, are proposed. Last, some details of the implementation of the grammar are given, including a novel method to model word order constraints and topological fields in the DELPH-IN formalism.

Chapter 4: Lexical acquisition In this chapter, the core grammar is extended with a lexicon in a deep lexical acquisition (DLA) step. The chapter commences with an introduction of methods with similar goals in mind. Then, an overview of the Tiger treebank (Brants *et al.* 2002), which will be used as the source for the DLA algorithm, will be given. This is followed by a description of the algorithm that is used to extract the lexemes from the treebank. Some descriptive statistics are given in order to be able to gauge the size of the lexicon and compare it to existing lexical resources.

Chapter 5: Leverage of the gold standard The Tiger treebank is not only used as a resource from which lexical information can be extracted. The experiments in chapter 5 employ the treebank as a gold standard. First, it is explained how the output of the grammar that is created in chapters 3 and 4 are translated into a form that is compatible with the Tiger treebank. This translation makes a number of additional applications of the gold standard possible. A new grammar testing technique, *unit testing*, is introduced, which offers the grammar writer the possibility to identify where in the chain of core grammar, deep lexical acquisition and output conversion mistakes and omissions are located. The second application is the automatic disambiguation of parsed linguistic data, which can be useful to automatically create an HPSG treebank from the source treebank. Naturally, a held-out set can be used to test the performance of the parser on unseen data, an experiment with which the chapter is closed.

Interlude The Interlude is an unnumbered chapter, which forms the bridge between experiments on the grammar and experiments on the parser. The grammar engineering paradigm of the previous chapters is evaluated as a whole, and compared to other grammar engineering paradigms. The comparison is with respect to a number of dimensions: linguistic relevance, level of abstraction, ability to generate, efficiency, coverage, development time and clean evaluation.

Chapter 6: Efficiency In the Interlude, it was concluded that, although several useful optimisation techniques have been implemented, the PET parser was a relatively slow parser, because it wades through the search space exhaustively, in contrast to most state-of-the-art competitors. The chapter starts with a discussion of existing techniques, and it is explained why these methods are not directly applicable to the PET parser. Instead, a new priority model on parser tasks is defined, which consists of a simple generative model of HPSG rule applications. Tasks with a lower priority are executed later, or might even be pruned away entirely. In the experiments, variants of the priority model and the different pruning strategies are tried, and an analysis is made of which tasks are actually being pruned. An evaluation on a test set reveal 10- to 20-fold speedups, without loss of parser accuracy (measured in dependency f-score).

Chapter 7: Robustness Another conclusion from the Interlude was the relatively high proportion of sentences for which no analysis at all is returned by the precision-oriented grammar. Two approaches are applied to the PET parser. The first is a variant of the well-known fragment parsing approach. The second approach is the definition of *robustness rules* in the grammar, which work as ordinary rules, except that they are penalised firmly by both the disambiguation model and the priority model. The robustness rules are meant to induce massive over-generation, which is tamed by the pruning algorithm from chapter 6. Also, a combination of both approaches is tried, which yield competitive results (an f-score of 73.1%), compared to hand-written and DGE parsers. The chapter concludes with a qualitative analysis of how the robustness rules behave in practice.

Chapter 8: Conclusions This chapter repeats the most prominent results of this thesis. A number of extensions for future work are also presented.

2 Background

The topic of this thesis is *parsing*: the automatic assignment of linguistic structure to an utterance. Often, a sort of grammar plays an important role in this process. Different types of formalisms for grammars (e.g. context-free grammar, Head-driven Phrase Structure Grammar) will be introduced in the first section, as well as a basic parsing algorithm (Cocke-Younger-Kasami: CYK), which makes use of these grammars. Also, the distinction between shallow and deep parsers will be discussed. A number of deep parsers based on hand-written grammars will be presented in section 2.2, along with a characterisation of their performance and the complexity of their development. An important resource in in the field of computational linguistics are collections of linguistically annotated linguistic data (so-called treebanks). They are introduced in section 2.3. How deep grammars can be derived from treebanks is explained in section 2.4. The grammar plays a pivotal role in the parsing process, but the success of a parser is also determined by the algorithms that employ the grammars. Attention to the optimisation of these algorithms in terms of efficiency and robustness is given in section 2.5. Because a grammar embedded in the DELPH-IN framework is created in this thesis, a separate section is devoted to the specific design of a DELPH-IN grammar and some of the inner workings of the PET parser. The chapter is closed by a motivation for the research that is carried out in this thesis. All in all, this chapter should give the reader enough background to get a good understanding of the research program that is presented in this thesis. However, it might still be necessary to follow references to get a thorough understanding of certain specifics.

2.1 Parsing and grammars

To start understanding what parsing is, let's consider the following sentence:

(1) A girl gave her cat food.

After a bit of thought, two different meanings for the same sentence come to mind, both equally plausible:

1. A girl was giving some food to her cat.
2. A girl was giving cat food to some female.

The phenomenon that one sentence can have different readings is called *ambiguity*. Syntacticians (researchers who investigate syntax/grammars) often annotate sentences with linguistic structures. It is common that two different readings for a sentence also lead to distinct annotations. One of the basic forms of

annotation is called constituent trees (or: phrase structure trees), and distinct trees for both readings are given below:

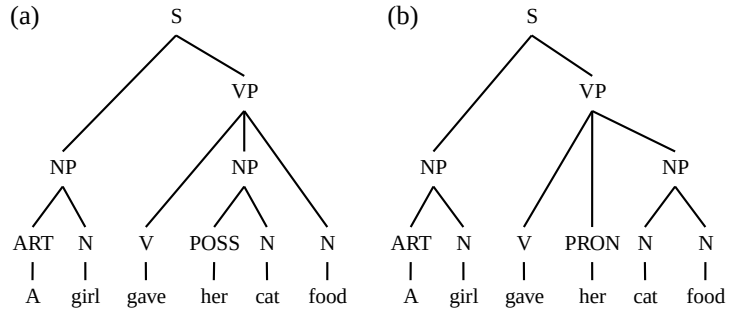


Figure 2.1: Two different constituent trees for an ambiguous sentence. The non-terminal node labels S, VP, NP and PP stand for sentence, verb phrase, noun phrase and prepositional phrase, respectively. ART, PRON, V and N stand for article, pronoun, verb and noun.

Each time two nodes are merged into one node, this upper node is called a constituent. For instance, in the left tree, 'her cat' is a constituent, but 'gave her' is not. For each language, there are clearly some regularities in how words combine to phrases, and how phrases combine to even larger phrases. One way to describe such regularities are grammar rules, and a set of such rules is called a *grammar*. An example of a certain type of formal grammars (context-free grammars) is given in figure 2.2. It is said that a grammar *generates* a sentence if at least one analysis can be formed that spans the entire sentence, and that only uses rule productions from the grammar. For instance, the example grammar generates 'A girl gave her cat food', because at least one tree can be formed that spans the whole sentence (see figure 2.1). On the other hand, it does not generate 'dog food', because dog is not in the dictionary. It also does not generate 'A girl gave a cat', because no proper VP can be formed. Most types of grammars define root conditions. In our example, the set of possible root symbols could be a singleton set with S, and in that case, the grammar does not generate 'a cat', because NP is not part of the set with start symbols.

It is straightforward to see how the grammar in figure 2.2 can be derived from the trees in figure 2.1, and this has actually been done (Magerman 1995;

S	→	NP VP	V	→	gave
VP	→	V NP N	N	→	girl
VP	→	V PRON NP	N	→	cat
NP	→	ART N	N	→	food
NP	→	POSS N	POSS	→	her
NP	→	N N	PRON	→	her
			ART	→	a

Figure 2.2: A small context-free grammar for English.

Charniak 1996). If a large set of annotated linguistic data is given (a *treebank*), it is easy to create a comprehensive grammar for a certain language automatically: one large pass over all trees is sufficient to derive all grammar rules that occur in the treebank. If the treebank is sufficiently large, one may assume that it would cover all possible (morpho-)syntactic phenomena of the language. We will see later that, although parsers based on such grammars perform reasonably well, a treebank is never large enough for convergence, due to the creative nature of language.

2.1.1 Deeper analysis of language

The grammar we have seen so far is fairly superficial, as many aspects of language are not included in this grammar. For instance, the English language knows the rule that the subject and the predicate must have equal number: the sentences 'The dogs barks' and 'The dog bark' are clearly ungrammatical. If our grammar is meant to be an acceptor of language, we certainly want to exclude such ungrammatical sentences and hence, the grammar needs to be constrained. A common way to achieve this is to specialise the grammar rules by adding *features*:

$$\begin{array}{lcl} \text{S}\,[\text{NUMBER sg}] & \rightarrow & \text{NP}\,[\text{NUMBER sg}]\ \text{VP}\,[\text{NUMBER sg}] \\ \text{S}\,[\text{NUMBER pl}] & \rightarrow & \text{NP}\,[\text{NUMBER pl}]\ \text{VP}\,[\text{NUMBER pl}] \end{array}$$

where 'sg' and 'pl' are abbreviations of 'singular' and 'plural', respectively. Initially, this looks like a good idea, but is not practical if more features are added, and rules have to be added for each possible feature value combination. A better and more compact approach is to under-specify the features, and use co-indexation to enforce equality:

$$\text{S}\,[\text{NUMBER } \boxed{1}] \rightarrow \text{NP}\,[\text{NUMBER } \boxed{1}]\ \text{VP}\,[\text{NUMBER } \boxed{1}]$$

When 'dogs' and 'barks' are combined with this grammar rule, they will turn out to be incompatible with each other, because they have 'pl' and 'sg' as their values for NUMBER. The process of trying to merge existing feature structures is called *unification*. This is an operator that takes two feature structures as input, and gives a new feature structure as output. A unification will *fail* when there is a conflict between the constraints, as will happen in the case with 'dogs' and 'barks'.

In order to take the abstraction one step further, we need to understand the concepts of *complementation* and *modification*. Consider the following examples:

(2) John runs.

(3) * John runs the window.

(4) John closes the window.

(5) * John closes.

It is obvious that the second and fourth sentence are ungrammatical (marked with a star). This is because the verb 'to run' is intransitive (it does not need an object such as 'the window' to be complete), whereas the verb 'to close' is transitive (it needs an object in order to be complete). Words or phrases that are needed before a word can be used by others are called *arguments*. Different types of arguments can be distinguished: subjects, specifiers and complements. The union of all arguments a word has is called a word's *valency*.

The following sentence exemplifies *modification*:

(6) John closed the large window yesterday.

This example shows two modifiers (or: adjuncts). The first is 'large', and it modifies 'window'; the second is 'yesterday', which modifies 'closed'. Modifiers tend to give more information on the word/phrase that it modifies: it's not just a random window, it's a *large* window. In terms of grammaticality, modifiers can be left out easily: 'John closed the window' is equally grammatical.

2.1.2 Head-driven Phrase Structure Grammar

The feature for NUMBER we introduced above can be represented in a lexical item. However, properties about modification and complementation can be represented in the lexicon as well. Consider the lexical entries for the transitive verb 'eat' in figure 2.3. Although terms as features, complementation and modification are widely accepted in the linguistic community at large, the structures in figure 2.3 are embedded in a specific linguistic theory called Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag 1994), of which a comprehensive introduction has been written by Bender *et al.* (2003).

The lexical entry in figure 2.3 showcases a number of HPSG's basic properties. The first is the concept of *typed feature structure* (TFS: Carpenter 1992), defined by a *type* (such as *synsem* or *index*), and a (possibly empty) mapping of attributes to values (an attribute-value matrix: AVM). The values in the AVM are TFSs as well, and causes the structures to be nested. Angle brackets indicate (ordered) lists. Apart from the TFS formalism, HPSG also dictates that a number of standard features ought to be used, such as SYNSEM (syntax-semantics), LOCAL, NONLOCAL, CAT (category) and HEAD. The HEAD feature indicates what the type of the word/phrase is, such as verb, noun, preposition etc. Often, the HEAD feature has subfeatures indicating gender, number, countability, etc.

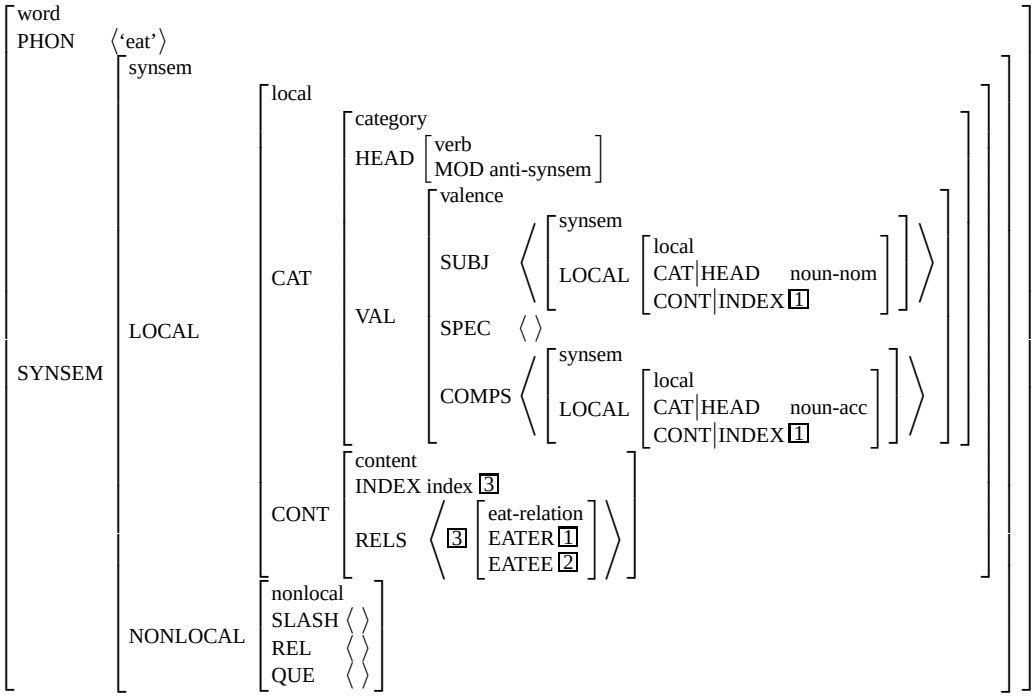


Figure 2.3: Depicted is a full HPSG-style lexical entry for the verb 'eat'.

The features in VAL reveal the valence of the word. In this case, 'eat' is a transitive verb, because a subject (in nominative case) and an object (in accusative case) are needed.

Because so much information is in the lexicon, the grammar rules turn out to be fairly simple and abstract, and not many of them are defined. Hence, HPSG is considered a strongly *lexicalised* framework. A construction (or *schema*, plural: schemata[1]) does not impose constraints on the HEAD of its daughters, but on other grounds. For instance, the HEAD-COMPLEMENT schema requires the head daughter to have a COMPS list with at least one element, and the first element should unify with the non-head daughter's SYNSEM feature. The mother takes over all complements of the head daughter, except the first; that is cancelled out against the complement itself. A TFS for the HEAD-COMPLEMENT rule is given in figure 2.4.

Other schemata that are commonly used in HPSG are HEAD-ADJUNCT, HEAD-SUBJECT, HEAD-SPECIFIER and HEAD-FILLER. The commonality between all these rules is that they are *headed*: the HEAD feature of the head daughter is co-

[1]I will use the words 'construction', 'rule' and 'schema' interchangeably, although one might argue that the terms refer to slightly distinct concepts. For instance, a schema can be seen as a rather abstract entity, whereas several constructions might be based on the same schema.

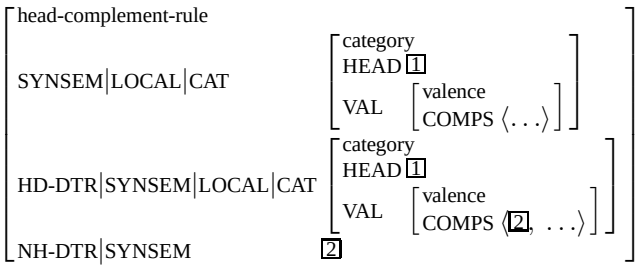


Figure 2.4: Shown is the basic structure of one of HPSG's immediate dominance schemata: HEAD-COMPLEMENT.

indexed with the mother's HEAD feature. Hence, HPSG is called a *head-driven* formalism. Figure 2.5 shows how a complete sentence can be analysed using these schemata. All words that are in the subtree with a certain HEAD are called the *projection* of the word that introduced that particular HEAD. In the example, the projection of 'eat' is 'eat cheese tomorrow', and the projection of 'cheese' is 'cheese'.

In the CONT(ENT) feature, the semantics of the projection of the feature structure is represented. It consists of a list/set of *relations* or *predicates*. Each predicate has a number of features showing how the relation is connected to other relations. The INDEX feature indicates which predicate is visible to other phrases to complete the semantic composition. In the lexical entry in figure 2.3, the co-indexations show that the EATER and EATEE roles are equated with the subject's and complement's INDEX feature. Rule applications merge the relations together into a large set, and the INDEX feature of the head daughter is usually taken over[2].

This way to represent the meaning of an utterance has been advocated by Pollard and Sag (1994), but more advanced formalisms have been proposed as well. One common extension is Minimal Recursion Semantics (Copestake *et al.* 2005), which can informally be characterised as predicate-argument structures with the ability to represent semantic scope of adjuncts as well. Also, the arguments are not named, but numbered: ARG0, . . ., ARG*n*.

One of HPSG's specialities is its treatment of unbounded dependencies. Let's consider the following sentences:

(7) That is the cheese Antje ate.

(8) That is the cheese he said Antje ate.

(9) That is the cheese we think he said Antje ate.

[2]In the HEAD-ADJUNCT schema, the modifier is the head. In this schema, the INDEX value is taken from the non-head daughter.

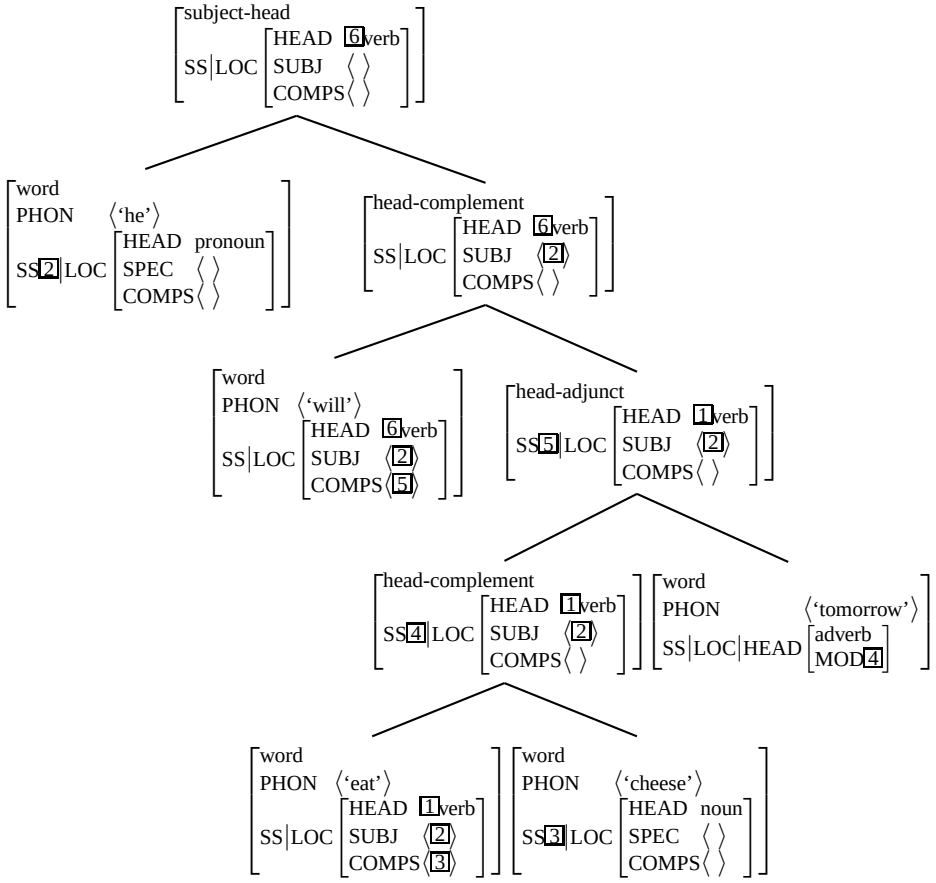


Figure 2.5: An HPSG analysis for the sentence 'He will eat cheese tomorrow'. For space reasons, some feature paths have been omitted. Also some features have been abbreviated.

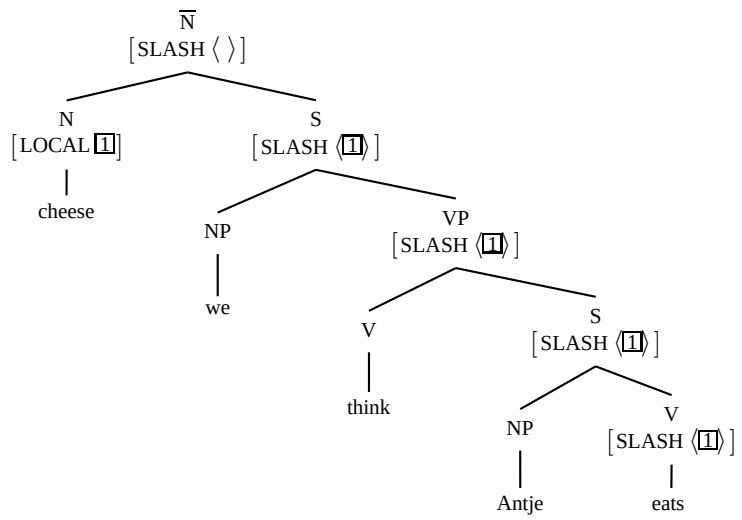


Figure 2.6: The figure shows how a SLASH value can be used to capture unbounded dependencies.

In some linguistic formalisms, it is said that in English relative phrases, there is a *gap* after the verb 'ate'. A gap is a simple feature structure that is not connected to any material in the sentence. In this case, it indicates that there should normally have been an object at the gap's location. Because the gap has almost no constraints, it will unify with the complement of 'ate'. The problem that has to be solved is that the verb 'said' is defined to take a verb phrase as its complement, but this verb phrase should be saturated (have empty valence lists), such as in the sentence 'He said (that) Antje ate the cheese'. This is not true for the phrase 'Antje ate', because the complement of 'ate' is still missing. An HPSG solution is to indicate the presence of a gap in the NONLOCAL | SLASH feature, and discharge the SLASH feature when the missing complement 'the cheese' is observed, high up in the tree. The rule to do the latter operation is called HEAD-FILLER. See figure 2.6 for an example. Example sentences 7-9 also show why this dependency is called unbounded: the number of times one can put a phrase between 'the cheese' and 'Antje ate' is sort of infinite (although it can become very hard to understand the utterance in the limit). Generally, HPSG uses the SLASH feature to facilitate the binding of arguments that are outside the projection of its head word.

I have now shown a glimpse of the properties of HPSG, which should help the reader understand the HPSG core grammar for German, whose design will be presented in chapter 3. However, there are plenty of other grammar formalisms for which good grammars and parsers have been constructed, such as Lexical-Functional Grammar (Bresnan 1982), Combinatory Categorial Grammar (Steedman 2000) and Lexicalised Tree-Adjoining Grammars (Joshi and Schabes 1997). I will not present the formalisms here, but the reader can

S	→	NP V	V	→	barks
N	→	N N	N	→	barks
NP	→	ART N	N	→	dog
			ART	→	the

Figure 2.7: A small context-free grammar for English.

assume that these formalisms have defined their own solutions for the phenomena discussed in this section (complementation, modification and long-distance/unbounded dependencies).

2.1.3 A basic parsing algorithm

Now we have a better understanding of what a grammar is, and how analyses of sentences look like, it is a good time to see how computer programs try to find that analysis automatically. One of the algorithms is the Cocke-Younger-Kasami (CYK) algorithm (Younger 1967; Kasami 1965), which I will briefly introduce here. In order to make the algorithm easier to understand, a small toy grammar is given in figure 2.7. The sentence that will be used for parsing is 'The dog barks', which can be read in two ways: there is a dog that barks; there are dog barks.

CYK parsing is a purely bottom-up chart parsing algorithm. The first step is to initialise the *chart*. The chart has generally $\frac{n(n+1)}{2}$ chart cells, where n is the number of words. In our example, the are 6 cells (one for each span): (0,1), (1,2) and (2,3) are lexical cells, and (0,2), (1,3) and (0,3) are phrasal cells, of which the last one should contain the root node in the end. The cells can be filled with chart items, if rule applications succeed. The chart initialisation phase consists of populating the lexical cells by a dictionary look-up. For instance, the span (0,1) gets one chart item: ART. The span (2,3) receives two items, because the word 'barks' is considered ambiguous by the grammar: one N node and one V node are inserted.

When the lexical phase is finished, the phrasal items need to be computed. In pseudo-code (assuming that the grammar is in Chomsky Normal Form, with only binary rules):

```
for l = 2...length(input)
  for i = 0...length(input)-l
    j = i + l
    foreach decomposition ((i,k)(k,j))
      foreach rule r
        foreach chart_item ci_left in chart[i,k]
          foreach chart_item ci_right in chart[k,j]
            c = match (r, ci_left, ci_right)
            if c != NULL
              chart[i,j].add(c)
```

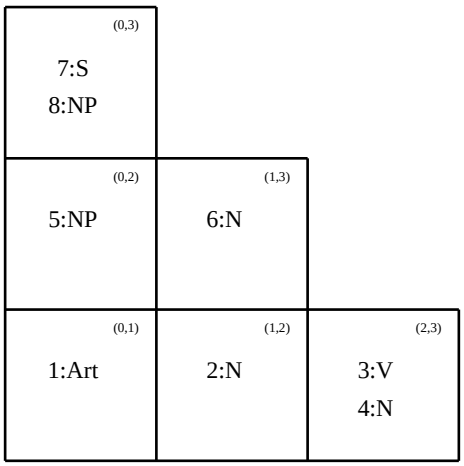


Figure 2.8: Depicted is a full parse chart for the sentence 'The dog barks', given the grammar in figure 2.7. The numbers between brackets indicate what the span of that particular cell is. For instance, (0,2) spans the words 'The dog'. The numbers before the chart items indicates the order in which the items are created and put on the chart.

The chart cells (0,2), (1,3) and (0,3) will be visited in this phase, in this order. For span (1,3), two items need to be combined: 'dog' and 'barks'. Only the noun compounding rule matches with these two items, and therefore one chart item is put in chart cell (1,3). When all chart items with span length 2 are computed (spans (0,2) and (1,3) here), the algorithm moves on to span length 3, which spans the complete sentence in our case. There are two decompositions of (0,3) possible: (0,2)(2,3) and (0,1)(1,3). The first will yield the S node (combining NP/'the dog' and V/barks); the second decomposition will result in an NP node (combining the ART/'the' and the noun compound N/'dog barks').

When the phrasal phase ends, it can be determined whether the grammar generates the input sentence, by looking if one of the nodes in the topmost chart cell is one of the start symbols. However, we are more interested in finding how the analyses look like. These can be reconstructed when backlinks are kept to the daughters of each node. Also, we want to know what the best analysis is. Choosing the best candidate parse among all possible analyses is usually achieved by a statistical model. The process of choosing the best parse according to some scoring model is called the *disambiguation* phase.

2.1.4 A taxonomy of parsing research

The task of parsing is perhaps one of the most widely studied subfields of computational linguistics, and a large variety of parsers have emerged over time. To understand what a parser does, let's first consider a definition of the task that a parser is subjected to: given a sentence, assign the correct linguistic structure to it. This is a fairly uncontroversial definition, but when one tries to fill in

```
h1 e2 {prop past indicative}
{ h3:pron_rel(x4 {3 sg m std_pron})
  h5:pronoun_q_rel(x4, h6, h7)
  h8:_play_v_1_rel(e2, x4, x9 {3 sg})
  h10:time_n_rel(x9)
  h11:def_implicit_q_rel(x9, h12, h13)
  h10:_yesterday_a_1_rel(e14 {prop untensed indicative, x9) }
{ h6 =q h3
  h12 =q h10 }
```

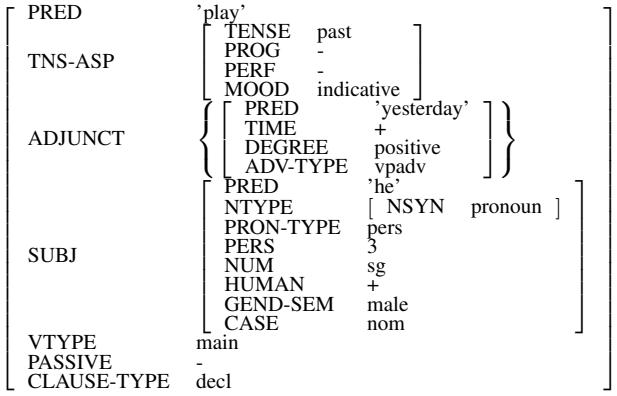


Figure 2.9: Depicted are (a) an MRS structure and (b) an LFG f-structure for the sentence 'He played yesterday'.

the details, there is not much agreement upon what the correct linguistic analysis is. I would like to refer to figure 2.9, where the preferred output of the simple sentence 'He played yesterday' is given for two parsers. The top figure is an account of the semantics of the sentence, in the form of Minimal Recursion Semantics (MRS: Copestake *et al.* 2005) roughly comparable to predicate logic, with the additional possibility to underspecify scope. It is composed of elementary predications (EPs), where each predicate/relation has a set of arguments, relating to other predicates. More on MRS will explained in section 3.3.4. The lower output is an f-structure, which is a syntactic representation. It is in the form of an untyped AVM. The figure functions as an illustration of the divergence between the different parsers. Also note that the differences tend to become larger when the sentences contain more complex phenomena. Not only the output is different across parsers, there exists also a wide array of distinct designs for parsers. An important building block of one method (probabilistic context-free parsing) has already been introduced before (a context-free grammar), but there are many more methods and formalisms. Some parsers are even entirely statistical, not making use of any sort of grammar.

Unfortunately, it is beyond the scope of this thesis to give an all-encompassing overview of all parsers. However, a crude division of all current parsers is given in figure 2.10. The most principal component in describing the variance between all different parsers[3] is *linguistic depth*, and hence I will differentiate between

[3]In this thesis, I will use the term *parser* for both the algorithm and the combination of algorithm

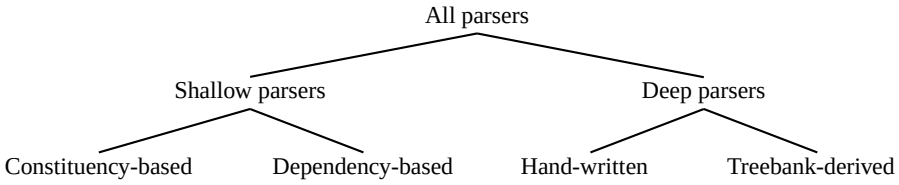


Figure 2.10: A taxonomy of active fields of parsing research.

shallow and deep parsers.

The first really successful parsers were *shallow parsers* or *statistical parsers*. For instance, the seminal work of Magerman (1995) created an entirely probabilistic version of a context-free grammar, whose probabilities were learnt from a treebank, the Penn Treebank (Marcus *et al.* 1994). Although the method is very simple, it works reasonably well. Later systems (Collins 1997; Charniak 2000; Charniak and Johnson 2005) were improved variants of these systems, for instance by adding an adjunct/complement distinction or a better parse disambiguation component based on a discriminative model. Whereas these systems were all based on phrase structure trees, other parsers have put their focus on syntactic dependencies (Nivre 2007; McDonald *et al.* 2005), also learnt from treebanks. The main advantage of dependency-based output is that it provides a very natural way to capture long-distance dependencies or non-contiguous trees.

The natural opponent of shallow parsers are *deep parsers*. 'Deep' can mean a number of things here, and this term is interpreted in different ways by different researchers. However, the following properties correlate fairly well across deep parsers:

The output is more informative Shallow parsers tend to give simple output only, such as constituent trees or simple dependencies. Deep parsers are capable to give a more detailed analysis of a sentence (as we have seen in figure 2.9 already). For instance, it can give morpho-syntactic information (such as tense, number, etc.). Also passivisation of a sentence can be detected, such that the agent and the patient roles of a verb can be assigned correctly[4]. In the MRS (Minimal Recursion Semantics) formalism (Copestake *et al.* 2005), the scope of adjuncts can be modelled as well.

The grammar only accepts grammatical sentences A number of deep grammars are also *acceptors*, meaning that the grammar was written to reject (i.e. not give any analysis) for sentences that were not licensed by the grammar.

and grammar. The meaning of the word should be clear from the context.

[4]Consider the sentences 'Antje eats the cheese.' and its passivised variant 'The cheese is eaten by Antje.'. The subjects of these sentences are 'Antje' and 'The cheese', respectively. However, the agent ('Antje') and patient ('the cheese') roles are filled by the same actor in both sentences.

This type of grammar can for instance be useful for an application judging the grammaticality of text produced by humans (Bender *et al.* 2004). Also, these grammars can form the basis for large-scale hypothesis testing in studies on syntax (Bender 2008b). An alternative name for such grammars is *precision(-oriented) grammars*.

The grammar can be used for generation Some deep parsers are bi-directional: the output of the parser can be fed into a generator, which will then generate grammatical sentences from it. This feature is highly dependent on whether the grammar is also meant as an acceptor: if the grammar licenses ungrammatical sentences, it will also generate ungrammatical output.

Apart from the individual merits listed above, there are also downsides to deep parsers. If the grammar was written to be an acceptor, it is hard to reach wide coverage on open domains, as there can always be constructions that do not fit the expectations of the grammar writer, resulting in a false negative. A second disadvantage of this approach is that the detailed symbolic models of language require a lot of human attention to be created. Generally, the development time is at least 10 person years, if not more. To be able to train statistical models (to allow for the automatic disambiguation between different readings of a sentence), treebanks have to be created as well, adding to the total cost of producing a hand-written grammar.

Many have sought ways to increase the scalability of deep parsers (both in terms of development speed and coverage), and a large family of parsers have arisen that were automatically derived from existing treebanks. Although deep parsers were long conceived to yield inferior results to state-of-the-art statistical parsers, these so-called *treebank-derived parsers* were able to fill the gap, not the least because sophisticated statistical models could be trained from readily-available treebanks. Treebank-derived parsers will be discussed in section 2.4, but first we'll delve into hand-crafted deep grammars.

2.1.5 Evaluation metrics

Before a number of parsers are introduced in more detail, it is important to understand which metrics are used to evaluate the performance of a parser. The first set of metrics is on the level of individual sentences:

coverage Indicates for how many sentences at least one analysis is returned.

annotation rate Indicates for how many sentences the exactly correct analysis is among the n-best list of analyses (as decided by the disambiguation model).

exact match Indicates for how many sentences the exactly correct analysis is ranked as number one by the disambiguation model.

Given the definitions of these types of evaluations, it can be concluded that the set of covered sentences is a subset of the set of annotated sentences and that the set of annotated sentences is a subset of the set of sentences for which the entirely correct reading is returned. All three types of evaluation can be criticised. The 'coverage' metric would in principle be fairly easy to crank up, as it does not assess the quality of the parses. As an extreme example, the grammar $w*$ has full coverage, but the resulting analyses are not informative. 'Annotated' and 'exact match' bear the risk of being too optimistic, as the person who is doing the evaluation might be tempted to be too permissive in accepting readings.

Presenting per-sentence numbers is not the standard in the parsing field at large. Evaluations based on smaller units (e.g. words) offer the advantage to be able to give credit to almost-correct parses as well. Such granular evaluations are usually presented using the following formulas, for precision, recall and f-score, respectively:

$$P = \frac{\textit{retrieved} \cap \textit{relevant}}{\textit{retrieved}} \tag{2.1}$$

$$R = \frac{\textit{retrieved} \cap \textit{relevant}}{\textit{relevant}} \tag{2.2}$$

$$F = 2 \cdot \frac{P \cdot R}{P + R} \tag{2.3}$$

In these formulas, *retrieved* is the set of predictions made by the algorithm, and *relevant* are the facts as stated by the gold standard. Hence, *retrieved* $\cap$ *relevant* equals the set of correct predictions. Precision is defined as the proportion of all predictions that is correct, and recall is defined as the proportion of all facts that are predicted. F-score is the harmonic mean of recall and precision, which penalises large difference between precision and recall.

2.2 Hand-crafted deep grammars

Over the decades, a wide variety of hand-written deep grammars have evolved. This section will go through the properties and performance of four of them. Three of the discussed parsers are for English (the English Resource Grammar (ERG), the XLE LFG parser and the RASP parser); one is for Dutch (the Alpino parser).

2.2.1 The English Resource Grammar

The English Resource Grammar (ERG) (Flickinger 2000) is embedded within the DELPH-IN framework and is therefore based on the HPSG linguistic framework and uses the MRS formalism (Copestake *et al.* 2005) for its output. The time to develop the ERG can be estimated to be at least twenty person years. Its main data structure, the type hierarchy, contains around 4500 types, of which 950 are leaf lexical types. These lexical types are in a many-to-many relation with the lexicon, which contains over 35,000 items. 33 lexical, 43 morphological and more than 200 phrasal rules have been devised. Large-scale parsing is done using the PET parser (Callmeier 2000), an agenda-driven parser for HPSG grammars, of which more details are given in sections 2.5 and 2.6.

Several papers evaluating the ERG's performance have been published. The first one, in which the Redwoods treebank is introduced (Oepen *et al.* 2004) only evaluates the parser on a per-sentence basis (we will come back to this specific type of treebanks, *dynamic treebanks*, in section 2.3.1). Because the focus of the grammar writer might be on improving the results of this treebank, an overly rosy picture of the grammar's performance can be created. It would be better, therefore, to have a truly independent treebank, against which the grammar writer does not evaluate the grammar too often.

The first attempt to test the ERG on independent data has been carried out by Baldwin *et al.* (2004), who applied the ERG to the British National Corpus (Burnard 2000). It was found that 32% of the sentences have full lexical span[5] of which 57% of the sentences receive a parse, of which 83% show the correct parse in all parses. At first sight, this indicates poor performance: $0.32 \cdot 0.57 \cdot 0.83 \approx 0.15$. This is due to a number of issues, lexical insufficiencies being the largest one: either a lexical item was not present in the lexicon at all (68% of all sentences), or the word was known, but was not assigned the correct lexical type by the lexicon (40% of the non-covered sentences). Other important shortcomings were identified as well: missing or ill-devised constructions (40% of the non-covered sentences), and the remaining 20% of the non-covered sentences were due to fragments, garbage input, incorrect preprocessing or resource exhaustion. Although the numbers are interesting, one can question how telling these percentages are, given that the percentages are only extracted from 32% of the corpus (those sentences with lexical span). One might think that longer sentences have a larger chance of having a lexical gap, and the division between the different categories in the error analysis might just as well be different for longer sentences. It is also not clear how cases are counted in which

[5]This means that for each word, a lexical entry is found in the lexicon. The procedure takes multi-word lexical entries into account, meaning that there must be a path from the start to the end of the sentence using only lexical items.

the grammar falls short on several points in the sentences.

Since this paper, much effort has been geared specifically towards cross-domain coverage, and this is demonstrated by the results presented by Ytrestøl *et al.* (2009), in which articles from Wikipedia are preprocessed and parsed, after which the correct reading of each sentence is picked by an independent annotator. The resulting treebank is codenamed 'WeScience'. The grammar's coverage varied between 81% and 91%, depending on the section, which is much higher than in the previous study on the BNC, even though the mean sentence length is comparable: 17.9 words per sentence (w/s). The proportion of all sentences for which the ERG was able to deliver the perfect analysis (the annotation rate) was estimated to be around 60%. The improvement of the results on unseen text can be attributed both to a better grammar (both its constructions and the lexicon) and better robustness facilities (unknown word handling, sophisticated pre-processing). The latest publication (Flickinger *et al.* 2010) showed even better numbers on unseen text from Wikipedia: around 85% of all sentences received at least one parse, and roughly two-third of the parsed sentences was assigned the entirely correct parse by the disambiguation model.

The authors note that the annotation rate is quite a bit lower than the one reported for the original Redwoods treebank (Oepen *et al.* 2004): 60% vs 82%. A plausible explanation for this difference is twofold. First, the linguistic data in the Redwoods treebank is a lot simpler, exemplified by the fact that the average sentence length is somewhere between 7 and 8 w/s. Second, the grammar and the Redwoods treebank have been developed concurrently, meaning that the ERG's developers had the chance to add constructions and lexical entries. This was not the case for the WeScience (Ytrestøl *et al.* 2009) and WikiWoods (Flickinger *et al.* 2010) treebanks, for which 'out-of-the-box' experiments were conducted. Therefore, it is unfair to make a straight comparison between the annotation rates for the Redwoods and WeScience treebanks.

So far, only sentence-level results have been shown. However, it is also interesting to give the parser credit for sentences that receive an almost entirely correct analysis. Before one can do an evaluation in a more granular manner for MRS, each MRS should be broken up into smaller, atomic pieces. How these pieces should look like is not a straightforward problem, and one specific way to do this is presented by Dridan (2009), called Elementary Dependency Match (EDM). For realistic, independent text (the *ws02* and *cb* sets), the ERG reaches f-scores between 63% and 70% if only NAMES and ARGS are counted, and f-scores between 65% and 73% are reported if NAMES, ARGS and PROPS are taken into account. For all these results, precision scores consistently higher than recall, because non-covered sentences severely harm the recall metric, but not the precision metric[6].

[6]There is ongoing (but unpublished) work on creating a bi-directional conversion of the grammar's

2.2.2 The ParGram parser for English

The Lexical-Functional Grammar (LFG) (Bresnan 2001) counterpart of the ERG is the ParGram parser (Maxwell and Kaplan 1993; Riezler *et al.* 2001) for English. The grammar features both a c-structure (in the form of phrase structure trees) and an f-structure (in the form of attribute-value matrices (AVMs)), as defined by the LFG formalism. From an algorithmic perspective, this can be very beneficial, as the phrase structure can be determined in polynomial time, whereas AVM unifications can take exponential time in theory (see (Maxwell and Kaplan 1993) for more details). In total, the grammar consists of more than 300 rules, and has a lexicon that is comparable in size to the ERG: almost 10,000 verb stems are identified. However, there are not many words in the lexicon for the other categories, as these are automatically recognised by a morphological analyser.

The parser's most important evaluation paper (Riezler *et al.* 2001) focuses on derivations of two resources: the Penn Treebank (Marcus *et al.* 1994) and the Brown corpus (Francis *et al.* 1982). From the first, a subset of 700 sentences (from section 23) has been annotated manually (King *et al.* 2003) with f-structures (in the form of predicate-argument structures), which indicate dependency relations between words (such as one word being the subject of another), but which also gives morpho-syntactic information about the words themselves (such as number and tense). On average, this set has 19.8 words and 31.2 predicate-argument structures per sentence. This resource is called the PARC700 corpus[7]. The second data set that was used for evaluation is the Susanne corpus, of which 500 sentences were annotated with grammatical relations by Carroll *et al.* (1999), to which I will refer as SUSANNE500[8]. The evaluation was done by converting the grammar's output (f-structures) to the grammatical relations scheme by a (possibly error-prone) hand-written procedure.

The coverage of the ParGram grammar is reported to be 74.7% (PARC700) and 79.6% (SUSANNE500) on a per-sentence basis, which is comparable to the ERG. However, when robustness measures are taken (*fragment* and *skimming*, to be explained in section 2.5.1), full coverage is obtained. The authors report f-scores of 78.6% (PARC700) and 73.0% (PARC700, only dependency relations), and 74.0% on the SUSANNE500 set (also only dependency relations). A later evaluation of the parser (Kaplan *et al.* 2004) yielded a slightly improved f-score on the PARC700 set: 79.6%.

MRS to a format called Dependency MRS.

[7]The names Dependency Bank, or DepBank, are also found in the literature.

[8]The Susanne corpus, in turn, is a subset of the Brown corpus (Francis *et al.* 1982).

2.2.3 The RASP parser

The RASP parser (Briscoe and Carroll 2002; Briscoe *et al.* 2006) is another example of a hand-written grammar for English. It is a tag sequence grammar (Briscoe 2006), a formalism that uses phrase structure trees augmented with features. Almost 700 grammatical rules have been devised. The grammar's development time is estimated to be 2 years only. The output is usually formatted as grammatical relations, as proposed by Carroll *et al.* (1998).

The authors report a coverage rate of 85% on SUSANNE500 (Carroll *et al.* 1999), their development set. For independent evaluation, the RASP system was evaluated on a re-annotated version of the PARC700 set (Briscoe and Carroll 2006) with grammatical relations (as opposed to f-structures). On this set, 84% of the sentences yielded an analysis that fulfilled the root condition, and an f-score of 79.7% is reported. However, due to the differences in annotation (f-structures vs grammatical relations; more details are given in the paper), this does not directly compare to the 79.6% performance of the ParGram grammar as reported by Kaplan *et al.* (2004).

2.2.4 The Alpino parser

The last parser I will discuss is the Alpino parser (Bouma *et al.* 2000; Van Noord 2006) for Dutch. This is a left-corner parser (Van Noord 1997), based on a hand-written HPSG grammar. Instead of using a purely unification-based formalism, the grammar is based on Definite Clause Grammars (Pereira and Warren 1980), implemented in PROLOG, allowing the grammar to use a richer set of operations, such as set membership. As we will see later, this can make the linguistic description of certain phenomena significantly easier. The grammar contains about 600 rules, indicating a fairly "constructional approach", of which the author claims that it is more efficient than a very lexicalised grammar (a grammar that encodes the bulk of the information of the grammar in the lexicon, and relies on a few very general schemata only). From the start, much effort has been invested into providing coverage for a wide range of domains[9]: the grammar's lexicon contains approximately 200,000 named entities; lexical rules to identify dates and other special named entities; a number of error-correcting heuristics; unknown word guessing based on the word's orthographemic properties. The grammar outputs syntactic dependencies in the form of a directed acyclic graph, allowing for node re-entrancies for raising, control and ellipsis in

[9]Nowadays, this also holds for the ERG, although this focus on wide coverage has only come in a later stage.

coordination[10].

An interesting choice of the authors is to prefer the *concept accuracy* metric rather than f-score as measure of success:

$$CA^i = 1 - \frac{D_f^i}{\max(D_g^i, D_p^i)} \tag{2.4}$$

where D_f^i is the number of missing or incorrect relations in sentence i, and D_g^i and D_p^i are the number of relations in the gold standard and the produced parse, respectively. Intuitively, one can see it as the inverse normalised distance to the gold standard parse, hence a slightly harsher metric than dependency f-score. The authors present results for three different data sets, of which only one (Trouw) is sufficiently challenging (newspaper text) and independent (not used as the grammar's development set). For this data set, a concept accuracy score of over 90% is reported.

2.2.5 Other hand-written grammars

Although I have introduced mostly grammars/parsers for English in this chapter (and actually most work in in the parsing community has been geared towards the English language), there have been considerable efforts to create grammars for other languages as well. Two large initiatives are aiming to facilitate the conscientious process of crafting grammars in a multi-lingual environment. The first is the ParGram project (Butt *et al.* 2002). Apart from the grammar for English discussed before, there are a few other grammars attached to that project, for German (Dipper 2003; Rohrer and Forst 2006), Japanese, Norwegian and Urdu.

A number of HPSG grammars are embedded in the DELPH-IN community, for French, German (Müller and Kasper 2000), Japanese (Siegel and Bender 2002), Modern Greek (Kordoni and Neu 2005), Norwegian (Hellan and Haugereid 2003), Portuguese (Branco and Costa 2008), Spanish (Marimon *et al.* 2007) and Wambaya (Bender 2008a).

DELPH-IN is also the home of the Matrix project (Bender *et al.* 2002), which provides grammar engineers with a starters kit for writing HPSG grammars, ensuring a basic level of consistency in the design of the feature geometry. Also, many of the technical practicalities have been taken care of. The natural extension of the Matrix is the Matrix customisation system (Bender and Flickinger 2005; Drellishak 2009). In this setting, a linguist can fill out a form, answering

[10]The reader might notice that the Alpino dependency scheme is very similar to the Tiger treebank's scheme. The Tiger treebank has indeed been an inspiration for the Alpino annotation scheme, but the re-entrancies in the Tiger treebank are only used for coordinations, not for raising and control.

rudimentary questions about the language at hand (about word order, existence of determiners, gender etc.). The system will return a customised grammar, offering a grammar with a number of words and lexical types, which can actually parse (and generate) a few basic sentences in that language. This can speed up the initial steps in the grammar writing process greatly, because the wheel need not be re-invented every time a certain phenomenon is implemented.

2.3 Treebanks

Treebanks and other annotated resources are an indispensable resource for current language technologies. They are used in many different circumstances and for different purposes. However, they face the same problem as hand-written deep grammars: the large investments that are needed in order to create a valuable resource (although the investments that are needed for treebanks are an order of magnitude smaller than for hand-written grammars). Any sizable treebank needs several person years by educated annotators to construct, which is time-consuming and expensive. This section will start out by giving an overview of a number of treebanks, and will explain how previous studies have made attempts to reduce the development time of treebanks, for instance by using existing grammars/parsers.

The value of a treebank depends on a number of factors. The first is the format that the treebank uses. Is the atomic unit a constituency or a dependency? Are crossing edges allowed? Are links between the nodes labelled? The second factor is the level of detail that is used in the annotation guidelines. More labels generally means that finer distinctions are being made. The third factor is the level of consistency that the resource keeps. If the same linguistic phenomenon is annotated differently in different places, this has a negative effect on the value of using this resource for either training or evaluation purposes. Usually, the workflow enforces some sort of checking or double annotation to reduce inconsistencies. The last factor determining the usability of a treebank is its size. For evaluation purposes, it can be relatively small ($<$1000 sentences). However, if the treebank is going to be used for training a certain symbolic or statistical model, it generally needs to be at least one order of magnitude larger to be useful.

One of the first large-scale treebanks, the Penn Treebank (Marcus *et al.* 1994), uses a constituent tree format, with a small number of additions. First, traces were put in the trees, to indicate long-distance dependencies. Second, functional tags were added to some nodes, to indicate whether a certain node is the subject of the clause, for instance. Although this resource has been immensely useful for the evolution of the field of parsing, there was also criticism on the treebank's

format, the most important being that constituent-based evaluation is not very useful for languages with less configurational word order than English.

Therefore, dependency-based representations have been proposed, for instance by Lin (1998), yielding a more suitable representation in cross-linguistic settings. The most basic form is where the sentence is transformed into a directed acyclic graph (dag), with the edges being labelled. Small variations exist in whether the edges of the graph can cross (non-projectiveness), and whether a node can have more than one incoming edge (introducing a re-entrancy). Some schemes also convey grammatical features, such as number or case, or even pronoun type. Also, no consensus exists on what a word is. Is 'vice versa' one word, or two? And what about 'to walk'? In some schemes, 'to' is just seen as a marker of 'walk', and is not represented as a separate node in the eventual graph. Some annotation schemes, such as Minimal Recursion Semantics (Copestake *et al.* 2005) even go one step further, aiming for a semantic representation rather than a syntactic one. See section 3.3.4 for more details.

2.3.1 Strategies for efficient annotation

The creation of a treebank is a costly and complicated task. Well-trained annotators have to spend time on the annotation of the text, and many problems can arise. For instance, a sentence may have more than one plausible analysis, or none, or the annotators might not agree with each other. The annotation scheme is therefore never undisputed, which might lead to inconsistencies over time, or between annotators. Hence, the process of creating treebanks is of interest: can their development be sped up? And what influence does that have on the accuracy? Can one treebank be converted to another format without losing information and/or consistency?

The procedure to create the Penn Treebank (Marcus *et al.* 1994) was simple, but cumbersome. First, the words were given part-of-speech tags using a tagger, and correct by the annotators. Then, the output of a parser (the Hindle parser: Hindle 1989) was hand-corrected as well, but because the parser left many found constituents as fragments, it required considerable additional effort from the annotators attaching these constituents to the main tree. Some other widely-used examples of manual annotation are actually re-annotations of existing resources. A part of section 23 of the Penn Treebank (the *de facto* evaluation standard for many studies on statistical parsing) has been manually annotated with f-structures (in the form of predicate-argument structures) (King *et al.* 2003) and grammatical relations (Carroll *et al.* 1999). A subset of 500 sentences from the Brown corpus has also been re-annotated with grammatical relations. Given the size of the latter sets, these are mainly used for evaluation purposes.

The Tiger treebank (Brants *et al.* 2002) is an interesting example, because two very different methods have been used to create the treebank. The first method makes heavy use of statistical models. Initially, all terminal nodes, of which the TnT part-of-speech tagger (Brants 2000) is confident that the tag can be computed correctly (in about 84% of the cases), it is automatically assigned, whereas the others are tagged by the annotator. When the sentence is tagged, an iterative annotation process is started, in which a statistical model (a Cascaded Markov Model) proposes new phrases one by one, and the annotator can intervene after each new proposed phrase. Around 71% of the model's proposals were readily accepted by the annotators, whereas the others needed some kind of intervention. The most interesting part of this approach was that annotation speed could be improved without a previously-existing resource: each time a few sentences were annotated, the statistical model could be re-trained such that the following predictions would be of higher quality.

The second method for the creation of the Tiger treebank was to use the output of an existing LFG parser for German (Dipper 2000). Because the majority of the sentences yielded too many parses, an Optimality Theory (Prince and Smolensky 2004) scheme was used to partially disambiguate between the analyses, reducing the average number of parses to 16.5 (but with a median of 2). The annotator could then choose between those parses, which was then converted to the Tiger treebank annotation scheme using the transfer procedure described by Zinsmeister *et al.* (2002). Promising as this technique looks, the grammar was not mature enough to give good coverage, returning no analysis for 50% of the presented sentences[11]. Furthermore, only 70% of the covered sentences (hence 35% of the complete set) contained the correct parse among the ones presented to the annotator. The workflow used in this study is not unique: the PARC700 (King *et al.* 2003), covering a part of section 23 of the Penn Treebank, and the Alpino treebank (van der Beek *et al.* 2002) have been developed in a similar fashion, although the disambiguation is aided in different ways and a solution was presented in a majority of the cases.

A special case of grammar-aided treebanks are *dynamic treebanks* (Oepen *et al.* 2004; Rosén *et al.* 2009), which is of specific interest for large hand-written grammars. Instead of recording the output of the parsing process (MRSs or f-structures, for instance), the process of getting to these outputs is retained. In the case of the Redwoods treebank (Oepen *et al.* 2004), the typical workflow consists of: parsing the sentence; extracting the 500 most probable readings from the packed forest; manually disambiguating which parses are deemed correct (which might turn out to be none, or more than one). The disambiguation step is discriminant-based (Carter 1997), speeding up the annotation process

[11] These figures improved in a later version of the grammar.

considerably. The relation between a chosen reading and the chosen discriminants is bi-directional: from the set of resolved discriminants (or: decisions) the resulting tree[12] can be computed; on the other hand, all decisions can be reconstructed if the correct tree is chosen. The use of dynamic treebanks has the advantage that when the grammar is updated in the future, the treebanking software can infer which choices were made by the annotator, so that updating the treebank can be done with minimal effort, as these choices can be reapplied: only new discriminants will be presented to the annotator. Speeding up the annotation can be done by resolving certain discriminants from an existing resource, for instance part-of-speech tags, as carried out by Tanaka *et al.* (2005).

Another method for creating a new treebank is by converting a treebank in one formalism to a new one in another formalism. Usually, this is achieved by using an automatic procedure, using heuristics written by a human. The heuristics can be in the form of scripts/macros, or a sort of transfer rules. The latter is done by Forst *et al.* (2004), who created TigerDB by a double conversion: first, the trees from the Tiger treebank are converted into an LFG gold standard for German, after which a DepBank-like format is extracted from the f-structures. Apart from the possibility to train statistical models from this resource, the main goal of the authors was to create a resource with which a sensible cross-parser evaluation can be carried out. We will see more examples of such treebank conversion heuristics in the next section, where treebanks are used to aid the construction of deep grammars.

2.4 Using treebanks to create grammars

Magerman (1995) and Charniak (1996) were among the first to show that it is straightforward and effective to learn a statistical parser (based on a PCFG) from an existing treebank. This work has been extended by many, and parsing section 23 of the Penn Treebank has been the *de facto* standard evaluation set for statistical parsing for more than a decade. Later, statistical parsers using dependency-style evaluations (Lin 1998) came into existence, with very good results as well (Nivre 2007; McDonald *et al.* 2005). In order to create training sets for these parsers, the Penn Treebank was automatically converted to dependency graphs. With the success of these conversions, the belief came that parsers using more sophisticated linguistic formalisms could also be learnt from a treebank. In this section, we will look at a number of these studies, converting the Penn Treebank to the LTAG, CCG, LFG and HPSG formalisms. Afterwards, two studies on creating German grammars will be introduced. For the range of studies in this section, I coin the name *deep grammar extraction* (DGE).

[12] If not all discriminants have been resolved, this is a set of trees. Also, all readings can be rejected if none of the readings is correct.

2.4.1 Deep grammar extraction for English

The algorithm that Xia *et al.* (2000) describe, reads off a Lexicalised Tree-Adjoining Grammar (LTAG: Joshi and Schabes 1997) from the Penn Treebank. First, the treebank is binarised, and heads, complements and adjuncts are explicitly labelled. The main instrument to do this is a head-percolation table in the spirit of Magerman (1995). It seems that traces are discarded in this step, indicating that long-distance dependencies can not be resolved. Second, the elementary trees (tree fragments to which the dictionary maps) are extracted, each being either a predicate-argument tree (a head with its arguments), a modification tree (a word modifying something else) or a coordination tree. Notice that this scheme leaves out the possibility to have an elementary tree that needs a complement before it can function as an adjunct. Hence, a preposition is represented as a tree that needs an NP as its complement, but it is not equipped to modify anything afterwards, for instance a verb or a noun, so it must be selected for by another word. In the example they give (Figure 6), some other unexpected trees come out: 'by then' is analysed as 'then' modifying 'by'. These are taken out using an extra filter, indicating which elements can modify other elements. After this filtering, the number of elementary tree templates is just over 6000. When certain categories (e.g. all verbal categories that only differ with respect to their morpho-syntactic information) are merged in the treebank, this number drops to below 3000.

The authors evaluate their method not in terms of parsing accuracy (which would depend on a lot of other factors, such as the quality of the disambiguation model and robustness measures), but on the proportion of words in section 23 that have the correct word-tag pairs in the learnt dictionary (column three in table 2.1). The other columns indicates how the word-tag pairs that are not learnt can be broken down. $2 \cdot 2$ categories can be identified: having seen the word or not, and having seen the template or not. So (sw,st) means: this word/tag combination is not in the dictionary the algorithm learnt, but the word was seen in the text, but was connected to another template, and the template has been observed as well, but has been assigned to other words. A later study (Chiang 2000) evaluates the parsing success using this grammar, which was competitive by that time's standards: labelled recall was 86.9%, labelled precision 86.6%. Current top algorithms have f-scores of around 91%.

(Hockenmaier and Steedman 2002; Hockenmaier and Steedman 2007) present a very similar algorithm for Combinatory Categorial Grammar (CCG) (Steedman 2000). The lexical types of CCG are not unlike the ones in LTAG, although there is one major difference: the elementary trees in the study by Xia et al. can only specify adjuncts modifying a certain phrasal category (S, VP, etc.), whereas the adjunct in CCG can also specify whether the modified category

Method	♯tags	Learnt	Not learnt			
			(sw,st)	(uw,st)	(sw,ut)	(uw,ut)
LTAG, normal	6099	92.6%	4.7%	2.5%	0.25%	0.02%
LTAG, compressed	3014	94.0%	3.4%	2.5%	0.08%	0.01%
CCG	1129	94.0%	2.2%	n.a.	0.02%	3.83%
HPSG	2345	95.0%	2.2%	2.7%	0.10%	0.00%

Table 2.1: Per tag evaluation of some of the treebank conversion methods.

needs complements. A second major difference is the explicit utilisation of the traces in the treebank. For instance, in the relative phrase 'what IBM bought', 'bought' receives a transitive verb entry in the dictionary. When the sentence is parsed, special operators (type-raising and composition) are used to correctly identify 'what' as the object of 'bought'. The attractiveness of this approach lies in the higher degree of generalisation of the lexicon: only one lexical entry is needed for the verb 'bought'. A third enhancement is the treatment of argument clusters, as in 'bought 2 books yesterday and 1 book today'. Here, a coordination of two non-constituents ('2 books yesterday' and '1 book today') is necessary, for which the CCG formalism provides a nice way solution. A few type-changing operators were added to facilitate a compact analysis of topicalisation ('The other half, we may have before long') and NP extraposition ('Factories booked $236.74 billion in orders in September, $\lceil_{\text{NP}}$ nearly the same as the $236.79 billion in August $\rceil$').

This grammar has been evaluated in the spirit of Xia et al.'s study (table 2.1). As was expected, there was a large improvement in the compactness of the lexicon (second column), with equal performance (third column), indicating for how many words in section 23 of the Penn Treebank the correct lexical type was assigned to that particular word in the learnt lexicon. Clark and Curran (2004a) present parsing results of this parser on the basis of labelled syntactic dependencies that are extracted from section 23 of the Penn Treebank. The parser shows an f-score of 86% on labelled dependencies, but one remark must be made with respect to these results. The conversion procedure of Penn Treebank-style trees to CCG is bound to contain errors, and because the syntactic dependencies used for evaluation are created using the same conversion procedure as the training set, the parser might be biased towards errors introduced by the conversion procedure.

Cahill *et al.* (2002) showed how a grammar in the framework of Lexical-Functional Grammar (LFG: Bresnan 2001) can be extracted from the Penn Treebank as well. In contrast to the monostratal nature of CCG, this study needs less work to be converted, as the LFG's c-structures are phrase structure trees. First, all nodes in the phrase structure tree are augmented with *functional annotations*, or *f-descriptions*, based on a set of hand-written regular expressions. From these augmented trees, rules can be learnt. An example of an extracted rule, including

its functional annotations, is:

$$\begin{array}{lccc} \text{S} \rightarrow & \text{NP} & \text{VP} & \left(\begin{array}{c} \text{ADV} \\ \downarrow \in \uparrow \text{ADJN} \end{array} \right) \\ & \uparrow \text{SUBJ} = \downarrow & \uparrow = \downarrow & \end{array}$$

In short, this means that the NP f-structure co-indexes with the SUBJ attribute in the f-structure of the S node, and that an optional adverb can modify the VP. Hence, the functional annotations fully determine the f-structure of the whole sentence. The importance of long-distance dependency (LDD) resolution, as shown in the studies on CCG and HPSG, is also underlined by a subsequent study (Cahill *et al.* 2004). Now, they call the previously derived f-structures *proto f-structures*, because certain parts of the derivations are non-deterministic, due to rules such as the following:

$$\begin{array}{lccc} \text{S} \rightarrow & \text{S} & & \text{NP} \quad \text{VP} \\ & \uparrow \text{TOPIC} = \downarrow & & \uparrow \text{SUBJ} = \downarrow \quad \uparrow = \downarrow \\ & \uparrow \text{TOPIC} = \uparrow \text{COMP}^* \text{COMP} & & \end{array}$$

where the * here is to be read as the Kleene star, requiring zero or more instantiations (LFG functional uncertainty). That means that when this rule matches, the NP might match the VP's complement, or the VP's complement's complement etc. A probabilistic model is created, choosing the candidate that maximises the product of the subcategorisation conditioned on the verb's lemma, and the joint distribution of the type of LDD and path that is taken.

The authors present two parsing architectures: *pipelined* and *integrated*. In the first architecture, the text is parsed using a normal PCFG model, and the annotation heuristics are used to create the f-structures. The second approach creates specific rules with different functional annotations, and parse using that (more sparse) annotated PCFG (A-PCFG), after which the constraint resolver will compute the f-structures. It turns out that the second, integrated strategy outperforms the pipeline by a few percentage points (depending on which test set is taken). More details on the methods are given by Cahill *et al.* (2005) and Cahill *et al.* (2008a).

The methods described above illustrate how the Penn Treebank can be converted to a deeper formalism using a heuristic converter or, in the case of LFG, how the trees can be annotated automatically. Although this is a sensible approach, there are limitations in scalability when extending this workflow to other approaches, because there is only a limited amount of information in the treebank, on the basis of which the heuristics have to operate. This can be seen in the CCG study already: apart from the standard CCG operators, the type-changing schemata were added to accomplish some generalisations in order to get a more compact grammar.

Raising the bar towards deeper instantiations of grammars that are embedded in constraint-based formalisms is not straightforward. One proposal to develop

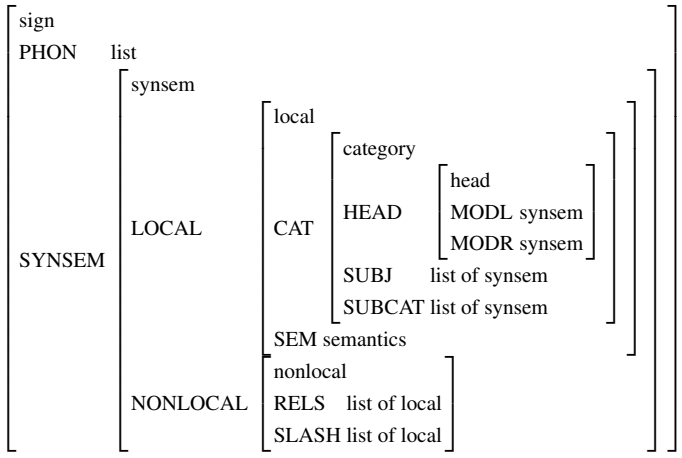


Figure 2.11: The feature geometry of the typed feature structures in the Enju grammar.

a deeper grammar (the Enju parser) is described by Miyao *et al.* (2004). As opposed to the previously discussed studies, Enju does not only inject deep properties to the grammar by the conversion procedure, but also by defining certain linguistic properties in the core grammar. In a sense, this was also done when a CCG for English was created (Hockenmaier and Steedman 2002) by defining an extra type-changing operator, but the addition of linguistic information was done more explicitly and elaborately when the workflow for Enju's grammar was designed. This becomes clear when the geometry of the typed feature structures within the Enju grammar is observed (figure 2.11). Features such as MODL, MODR (left and right modification, respectively), SUBJ and SUBCAT can be equally expressed in the other formalisms, but the possibility to use the NONLOCAL features stands out.

The geometry in figure 2.11 forms the basis for the construction of seven HPSG schemata: SUBJECT-HEAD, HEAD-COMPLEMENT, HEAD-MODIFIER, MODIFIER-HEAD, FILLER-HEAD, HEAD-RELATIVE and FILLER-INSERTION. These rules define constraints on how words/phrases can be combined, as explained in section 2.1.1.

The basic tree structure in HPSG does not contain labels for phrase categories, but applications of the above-mentioned rules (in fact, the schemata abstract away from these categories). In order to convert the Penn Treebank-style trees to HPSG-style trees with rule applications, the authors introduce a new technique. First, the trees are binarised, and each subnode is being annotated with a label that indicates whether it is the head in that particular rule application or not. Also, the information about the traces is used to augment the head/non-head annotations with information about the NONLOCAL properties of the nodes (similar to (Hockenmaier and Steedman 2002)). Then, a feature structure is built at the root node, which should be known: the head is a verb, no SUBJ and SUBCAT information is needed, and RELS and SLASH are empty lists.

The annotated rule schemata are re-played downwards. Due to the head/non-head and NONLOCAL annotations, the correct rule application tree can be constructed top-down in a deterministic fashion. The last step is to read off the lexical types (and the lexical entries in the dictionary) directly from the tree.

The authors report that the Enju grammar acquired 42,000 words, of which 25,000 are nouns and 10,000 are verbs. They arrive at 2,345 templates, of which 1,596 for verbs, and they claim this is significantly less than Xia's LTAG extraction method (although a closer inspection of the results seem to suggest that this is only before applying the filter for very unlikely templates). Lexical coverage on section 23 of the Penn Treebank was reported to be 95.0%, 1 percent point higher than in previous studies. The sentential coverage, measuring the proportion of sentences for which for all words the required lexical template was observed in the training set, was reported to be 43.0%. In a later version of the parser (Miyao 2006), lexical and sentential coverage rates of 99.1% and 84.1% were reported. Parsing results (Ninomiya *et al.* 2006) show labelled precision and recall of about 87% on predicate-argument structures, induced using the same procedure.

2.4.2 Deep grammar extraction for German

So far, I only discussed deep grammar extraction methods for English. Similar approaches have been taken to create valuable resources for German. This is interesting mostly because one would like to prove that the methods work well independent of the language or the resources that are chosen. Specifically, it is sometimes argued that German is harder to parse than English (Kübler *et al.* 2006), due to the lower degree of configurationality in the German language and a richer morphology (an introduction to some German syntactic properties is given in section 3.1). Both properties are reflected in the Tiger treebank for German (Brants *et al.* 2002), on which the two studies that I report on are based. First, the terminal nodes (the words) are annotated with more extensive morphological information than the Penn Treebank. Second, the data structure in the Tiger treebank is quite different from the Penn Treebank. It allows discontinuous (or: non-projective) trees, which happen in about one third of the trees. Also, secondary edges 6 used to represent ellipsis in coordinations. The last main difference between the Tiger treebank and the Penn Treebank is the relatively high arity of the nodes in the former (> 5 in some cases). A more elaborate introduction on the Tiger treebank can be found in section 4.2.

Cahill *et al.* (2005) were the first to adapt their DGE technique to create an LFG grammar for German. In order to make their algorithm work properly, the Tiger trees are converted to Penn Treebank-style trees. Discontinuities are

removed by adding traces. They report a 74.6% f-score on syntactic dependencies from a 2000-sentence held-out set from the Tiger treebank, which is significantly lower than for English. This can be attributed to the flat nature of the Tiger treebank, which poses sparseness problems when the c-structures are created using an external parser. A subsequent study (Rehbein 2009), devoting more attention to the type of PCFG parser and morphology, reaches a maximum f-score of 77.3%.

Hockenmaier (2006) also did a study converting the Tiger treebank to a CCG treebank. Analogous to the previous study, an additional preprocessing step was carried out to render the trees in the source treebank projective. This process is rather crude: constituents that cause non-projectiveness are attached to higher nodes, until the discontinuity has disappeared. The paper presents quite a few details on how certain constructions are handled, such as cluster coordination and extraposed relative clauses. No parsing experiments are reported, only lexical coverage numbers on unseen text (similar to the scores in table 2.1). The lexical coverage turns out to be quite a bit lower (86.7% vs 94.0%) and the number of lexical types (2506 vs 1129) was higher than the English CCG counterpart. This indicates a lack of generalisation in the learning process. For instance, German allows different permutations in saturating its complements. Because CCG is such a strongly lexicalised framework, different entries have to be created in the lexicon to account for these permutations. This is unwanted, because it is known that this type of permutation applies to all verbal lexical types in German, so learning that the arguments of all verbs can be permuted for each individual verb separately seems to cause the inferior results.

2.4.3 Evaluation issues

I referred earlier to the claim that German is harder to parse than English, which is contradicted by Kübler *et al.* (2006). The results in this section seem to support this hypothesis, as the grammars for German are performing significantly worse than their English counterparts. It must be noted, however, that state-of-the-art results of statistical dependency parsers for German (on the same data sets) actually compare favourably to dependency parsers for English: $> 90\%$ (Hall and Nivre 2008) vs 88 % (Hall *et al.* 2007). Therefore, it seems reasonable to propose the alternative hypothesis that current DGE techniques have a bias towards languages with a higher degree of configurationality. The method I will introduce in the following chapters is intended to overcome this variance with such language-dependent properties.

Another evaluation issue in DGE studies in general is related to the way the gold standards are created. All studies in this section, both for English and

German, have resorted to a sort of conversion technique to create a treebank in the target formalism (HPSG, LFG, etc). The created treebank is then divided in a training set and a test set, meaning that the grammar is evaluated against an automatically converted gold standard. There are two (related) objections against this mode of evaluation. First, there might be errors in the conversion procedure. The procedure might be designed to create finer distinctions that are not made in the source treebank, and the discovery of these distinctions is error-prone. The second objection is that the conversion procedure can discard fine distinctions that were available in the source treebank. To understand how this can happen, one has to realise that the distribution of linguistic phenomena in a corpus probably approximates a Zipfian distribution: there are a few phenomena that are frequent, and many that are infrequent. However, there is a fair chance to observe one of these infrequent phenomena in a sentence, because the tail of infrequent phenomena is long. The writer of the conversion script will most likely devote more attention to the frequent phenomena, and the fine distinctions for infrequent phenomena might disappear in the conversion. It is reasonable to assume that the infrequent phenomena are harder to model than the regular ones, and therefore one can conclude that it is easier to reproduce the facts in the new treebank than it was for the source treebank.

2.5 The interplay between grammar and parser

Two challenges are often associated with parsing with deep grammars: robustness and efficiency. In the first case, a deep grammar does not find a solution for a sentence. This can be due to a variety of reasons, as we will see later. In the second case, the parser does find an analysis, but the computation takes too long to be practical. In both cases, there is an interaction between the grammar and the algorithm that employs the grammar. If the grammar is over-generating, the computational bounds can be met (time or memory requirements), or the parser can be too slow. On the other hand, if the grammar is constructed to be fairly precise, a sentence might not receive any analysis at all, creating the need to have a fall-back option in case this happens.

2.5.1 Robustness methods

A large subset of existing hand-crafted deep grammars are also *precision parsers*, meaning that they aim to provide a model of grammaticality. An advantage is that less load is put on the disambiguation model, as the strict modelling of the language at hand already restricts the solution space. A disadvantage is that it is hard to reach full coverage with such grammars. The following causes of a

non-covered sentence can be identified:

Preprocessing errors A wide variety of errors fall into this category: sentences or words might be split erroneously, the sentence contains foreign language, or entities such as dates or addresses (with a very distinct syntax) are not recognised properly.

Insufficient lexicon The word is not in the lexicon, or the lexical entries in the dictionary for that word are insufficient (or even incorrect).

Insufficient constructions The grammar writer might not have considered a certain linguistic phenomenon, or the set of rules does not support the phenomenon[13].

Sentence is deemed 'ungrammatical' There can be spelling/transcript errors in the text, or a different dialect or spelling convention is used.

For most applications, returning no analysis is undesirable, and that is what robustness methods are aiming to counter. Preprocessing is a task with many 'dirty details', and can be surprisingly difficult when a parser is applied to text from a previously unused source. A recently developed technique for the PET parser is *chart mapping* (Adolphs *et al.* 2008), which facilitates the cleaning and preprocessing of text, such as lightweight named entity recognition, by using a rewrite system.

Unknown word handling is also an important issue in deep parsing, as we have seen before (Baldwin *et al.* 2004). As it turns out, many unknown words are proper nouns, so assuming that all unknown words are proper nouns is not a bad first guess. However, one can find reliable cues to determine the type of a word. One such cue is the word's orthography. For instance, an English word that ends with 'ion' is likely to be a noun, and capital letters indicate proper nouns[14].

Also, the parts-of-speech of the word's immediate context can be a useful indicator of the unknown word's lexical type. The *posmapping* strategy that is commonly used when the PET parser is applied to unseen text makes indirect use of this clue. First, the text is tagged with broad lexical categories (such as verb, noun etc). The grammar defines a mapping of each tag to one or more lexical types, and each time an unknown word is encountered, the tag is looked up in the mapping, and the appropriate lexical types are put on the chart. This turns out to work reasonably well, because the distribution of lexical types follows a Zipfian distribution: there are a few lexical types that cover a large share of the

[13]For certain phenomena, it is hard to constrain a rule that supports the phenomenon. In such cases, the grammar writer can decide to discard the rule, in order to shrink the size of the solution space.

[14]The ParGram parser for English makes heavy use of these indicators. Most nouns, adjectives and adverbs are considered to be so predictable that the lexicons for these categories are very small.

lexical instances, and hence unknown words are likely to be part of this small set. For example, it is likely that an unknown verb is either an intransitive or transitive verb, but not an auxiliary.

But even when full lexical span is achieved, the parser might still not find a solution for the complete utterance. As we have seen before, this can have a number of causes, but it is hard for the parser itself to identify the cause of the non-covered sentence. Therefore, fall-back strategies are defined, which usually give suboptimal solutions. However, it is still considered to be better than giving no solutions at all.

The most straightforward approach is a technique called *fragment parsing*. This algorithm harvests individual fragments from the chart, and combines them in order to get an analysis that spans the complete sentence. Which pieces are taken is usually based on a minimum-cost algorithm, where cost is minimised when the number of fragments is lowest, or where the combined probabilities of the fragments is highest. This baseline method has been used by many precision parsers. More details are explained by Kasper *et al.* (1999), Bouma *et al.* (2001), Riezler *et al.* (2001) and Zhang *et al.* (2007a), although the algorithms will be elaborated on in chapter 7 as well, in which I will propose a new method to achieve a more robust parser.

More innovative is the hybrid approach taken by Zhang and Kordoni (2008). If the grammar does not license any structure to a particular utterance, a second parsing stage is invoked, in which a PCFG model elaborates on the chart that had been formed until then. It will create a so-called *pseudo-derivation tree*, of which the unification may not be successful, but a proper semantic representation for the whole sentence can still be inferred. Because there is no gold standard for sentences that can not be parsed, it is hard to evaluate the outcome of an experiment. However, the authors claim that promising results were obtained in a preliminary experiment. They did a manual inspection of the resulting trees for a subset of the PARC700 set (King *et al.* 2003), and for the 54 sentences that invoked the second stage, the ERG created created pseudo-derivation trees for 41 of them. Of these, 13 were completely correct and 18 had no more than two crossing brackets.

And what should happen when computational resources are exhausted, i.e. when a time or memory limit is hit? The usual setting for the PET parser is to stop parsing, but the XLE parser will fall back to a technique called SKIMMING (Riezler *et al.* 2001): instead of going through the search space exhaustively, there is only a limited amount of work per subtree, making the rest of the parsing process polynomial instead of exponential.

2.5.2 Search space restriction

Quite a few papers have addressed the problem of computational complexity of deep parsing. Initially, the focus was on creating algorithms that guarantee that the parser will return identical results to a parser without the modifications. A popular, non-intrusive method to bring down computational requirements for parsing is called *packing*. In context-free grammar packing, if there are two subtrees creating the same category for the same span, only one of them (the representative) will be used in the rest of the parsing process. Using packing in context-free parsing assures that the computational resources that are needed to complete the parse is quadratic with respect to the sentence length, and linear with respect to the number of categories. Variants of packing for deeper formalisms than CFG are introduced as well (Maxwell and Kaplan 1993; Miyao 1999; Oepen and Carroll 2000a). I will introduce PET's non-intrusive efficiency techniques in section 2.6.2.

Another way to speed up a parser is by restricting the search space. These methods do not guarantee that the optimal solution will be found. However, we will see that the currently developed methods perform surprisingly well in terms of speed-up rates, without loss of accuracy. Depending on the exact metric, such methods can even yield improved results. I will discuss some of these techniques in this section.

Most deep formalisms are lexicalised at least to a certain degree, and most work on restricting the search space has aimed at doing so on a lexical level. A deep parser usually has a dictionary of *supertags*, which can be characterised as specialisations of part-of-speech tags. Supertags contain more syntactic and/or semantic information than a part-of-speech tag, for instance which kinds of complements a word has, gender, scope information, a noun's countability etc. The process of restricting the placement of supertags on the initial chart is called *supertagging*. Because this restriction turns out to be very effective, this has also been called "almost parsing" (Bangalore and Joshi 1999). These techniques have been applied to a number of deep parsers, for example by Matsuzaki *et al.* (2007), Clark and Curran (2004b), and Dridan *et al.* (2008). Later studies on the Enju parser did not do the supertagging on the basis of the information on the words alone, but incorporated a statistical parser into the process. The result of this stage could then provide an indication of whether the supertag would be needed in a later stage in the deep parsing process. The shallow parser can be either a parser based on a CFG (CFG filtering: Zhang and Matsuzaki 2009) or a statistical dependency parser (Zhang *et al.* 2010b).

There are also a number of studies on restricting the search space of the parser on a phrasal level. Cahill *et al.* (2007) describe how, instead of using the unifications immediately, they first apply a PCFG parser to the raw text. Subsequent

unifications are only allowed if the resulting c-structure would not cross brackets with the PCFG tree. The first results they present are poor, showing a large decrease in parsing time, but also a severe drop in coverage. To counter this, they propose a hybrid system, in which first the fast but low-coverage parser is tried, and the full parser is used as a fall-back strategy. This final system reduces parsing time by 18% without losing accuracy.

However, a more recent study of the same authors shows better results (Cahill *et al.* 2008b). Now, the c-structure pruning is not based on isomorphism with existing trees (modulo unary rule applications), but on the relative probabilities given by the PCFG within one chart cell. If a subtree receives a score lower than the highest score in that cell divided by a constant, that subtree is pruned. The final result they present is a parsing time decrease of around 67%, with a slight *increase* of f-score. This is attributed to the fact that less sentences go into SKIMMING mode under this setting.

Ninomiya *et al.* (2005) describe two techniques that I will discuss: beam thresholding and iterative parsing. *Beam thresholding* prunes away items locally (within one parse cell), if the item's figure of merit (FOM) is lower than the highest FOM in that cell, minus a certain (log) bandwidth. Hence, this approach is very similar to the one taken by Cahill *et al.* (2008b), except that the figures of merit are based on a discriminative model, and not on a generative model. This restriction strategy does not prevent the creation of very improbable chart items, however prunes them away *after* they are created. In parsing with the PET parser, this could be costly, which it is not the case in Cahill *et al.*'s method, because most of the parsing time is in the unification, which is only to follow after the c-structure pruning.

The second method the authors introduce is *iterative parsing*. The algorithm describes how the parser starts with a very small beam initially, pruning away relatively large parts of the search space. If no parse is found, the beam is widened until there is an analysis covering the complete span. The implementation they use makes sure that chart items that have been created (but which are located outside the beam) are not destroyed, but only temporarily marked as invalid, so that re-creation of chart items is unnecessary when the beam is enlarged.

An innovative approach to search space reduction is taken by van Noord (2009). Instead of relying on a large gold standard treebank, which might not be available, their system assumes that the output of the parser is good enough to learn which parser tasks are more promising than others. Hence, the algorithm parses large amounts of text (40 million words), disambiguates the best solution according to an already existing disambiguation model, and learn which left-corner splines are more promising than others. Interestingly, taking the mere existence of a particular spline in the training set as the criterion already shows

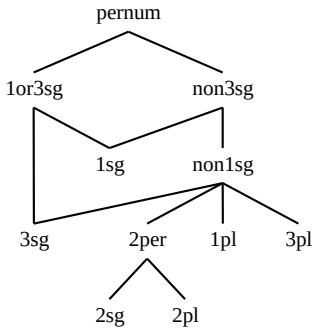


Figure 2.12: Depicted is a type hierarchy for person and number (adapted from (Flickinger 2000)).

good results: a speedup of 75% (the average parsing time reduced from 25 to 6 seconds), without loss of accuracy.

2.6 Anatomy of a DELPH-IN parser

A DELPH-IN parser can be divided into two components: the language-independent PET parser (Callmeier 2000) and the grammar, which is specifically constructed for one particular language. This thesis encompasses the creation of such a grammar (in the first part of the thesis) and modifications to the PET parser (in the second part), and therefore it is necessary to understand the details of both components to a certain degree. First, I will discuss the formalism of the grammar, followed by a discussion on some specifics of the PET parser.

2.6.1 The grammar

A central data structure in a DELPH-IN grammar is the *type hierarchy*. Each node in the hierarchy is a typed feature structure (Carpenter 1992), with each of the values in the feature structure being a type of the same hierarchy. Every node is a specialisation of its parent node(s), because each subnode automatically inherits all constraints from its parents, and (potentially) adds extra constraints. A typical DELPH-IN grammar has a type hierarchy of a few thousand nodes. Building up a large type hierarchy, creating lexical types and a lexicon is a complicated endeavour. Hence, effort has been put into providing starting grammar writers with a good starting point. A good candidate for such a starting point is the Matrix (Bender *et al.* 2002), as discussed in section 2.2.5.

The type hierarchy defines all sorts of linguistic entities: data types (*number*, *case*, *boolean*, etc), composite types (*sign*, *synsem*, etc), rules and lexical types. The type hierarchy can be used to create smart generalisations (using underspecification), so that tight representations can be built (see figure 2.12). This

is useful, as the word ‘walk’ can get the type ‘non3sg’. This yields more compact representations than when the person and number attributes are separated, because in that case, two separate items are needed: one representing first and second person singular, and one for plural.

The second component of a DELPH-IN grammar is the lexicon. This is basically a mapping from stems to lexical types, and includes a small bit of extra information (all examples are taken from the ERG):

```
consumption_n1 := n_-_m_le &
 [ ORTH < "consumption" >,
   SYNSEM [ LKEYS.KEYREL.PRED "_consumption_n_1_rel",
            PHON.ONSET con ] ].
```

Here we see the first example of a $\mathcal{TDL}$ expression (Krieger and Schäfer 1994)[15]. This syntax means that the type `consumption_n1` is an instantiation of the lexical type `n_-_m_le`[16] and the remaining lines are meant to further specialise the AVM that was introduced by the lexical type. The angle brackets are used to indicate lists. Text between quotation marks indicate strings. Everything else is considered to represent a type from the hierarchy. `con` is used here as an indication that the onset is a consonant, which is important for the ‘a’/‘an’ distinction in English. The second line indicates what the key relation is, in order to attribute a form of semantics to the lexical entry in the lexicon.

The next component is the rule set. Two types of rules are defined: lexical and phrasal rules. Phrasal rules are mostly simple instantiations of their respective types in the type hierarchy, as well as lexical rules that do not change the orthography of a word. Morphological rules (lexical rules that do have an effect on the orthography) are represented with an additional type of $\mathcal{TDL}$ syntax, in which a transformation mechanism based on regular expressions is used to represent the orthography changes. Consider the following definition of plural inflection of English nouns:

```
n_pl_olr :=
%suffix (!s !ss) (!ss !ssses) (es eses) (ss sses)
lex_rule_infl_affixed &
[ ND-AFF +,
  SYNSEM mass_or_count_synsem &
         [ LOCAL plur_noun ],
  RNAME "LNPL" ].
```

Of course, not all noun inflections are regular, and irregular morphological rule applications can be represented using the following format, with one (word form, inflection rule, lexeme) triple per line:

```
corpora N_PL_OLR corpus
```

[15] Actually, current DELPH-IN grammars only use a subset of the expression power of $\mathcal{TDL}$, as disjunction turned out to be computationally too expensive while parsing.

[16] When the type hierarchy is defined, the same syntax is used to indicate inheritance. Multiple inheritance is allowed in this case.

The last major component of a DELPH-IN grammar is the list of *root conditions* (or: start symbols). This indicates how the root of the parse tree should look like in order for the parse tree to be considered a valid utterance. Usually, root conditions require the top node's complement list to be an empty list. Distinct root conditions can for instance be defined for declarative, interrogative and wh-sentences, and separate ones for stand-alone NPs, PPs etc. An example of a root condition from the ERG is:

```
root_question := phrase &
  [ INFLECTD +,
    SYNSEM root_synsem &
             [ LOCAL.CONT.HOOK.INDEX.SF ques ],
    DIALECT us ].
```

2.6.2 The PET parser

A parser for the data structures described before is the Linguistic Knowledge Base (LKB) (Copestake 2002). This is a grammar development platform for HPSG, with some additions to facilitate the treatment of MRS structures. It comes with a graphical interface, with which typed features structures from the type hierarchy can be inspected. Apart from being able to parse a sentence (and being able to give detailed information on the parsing process, including unification failures), the LKB can also generate utterances that are licensed by the grammar, using an MRS structure as input. However, the implementation has its computational limitations, and that was the reason to develop a replacement: the PET parser (Callmeier 2000). It is implemented in C/C++, and special attention has been paid to optimising the computationally expensive TFS operators (unification and subsumption). However, there is no graphical interface, and neither can it generate from a semantic input, and both systems thus co-exist.

Parsing with the PET parser consists of the following stages:

Pre-processing The raw text is input into the chart mapping machinery (Adolphs *et al.* 2008), which can transform all sorts of different input into the tokenisation that the grammar expects. For instance, the Penn Treebank format can be converted this way, including repairing tokenisation mismatches, for instance by a conversion from 'do n't' to 'don't'.

Morphological analysis All words are subjected to regular expression-based rules for morphological analysis, in order to arrive at lexemes. As we have just seen, this analysis is done using both regular transformations and an exhaustive list for irregular inflections.

Lexical look-up The possibly inflected words are then matched with lexical entries. The features that were added during the previous stage should match

with the lexical entry. For instance, a hypothetical grammar for English would analyse 'dogs' as both a third person, singular verb and as a plural noun, but only the latter analysis will survive in this stage, because 'dog' is not a valid verb. Notice that, although morphological analysis is done before lexical look-up, the morphological rule applications are put *higher* in the derivation tree than the lexical entries.

Parsing In this stage, an agenda-based chart parsing algorithm is used (Kay 1980). Under normal circumstances, this stage takes up the lion's share of the computation time. Several techniques are used (discussed immediately hereafter) to reduce the computational requirements.

Solution collection After the parsing stage is finished, the chart needs to be inspected for correct readings, where 'correct' is defined by whether the top node complies to one of the root conditions specified by the grammar writer. Also, the different analyses need to ranked with respect to the reading's plausibility.

Packing, unpacking and other efficiency techniques

The part in the pipeline that requires most computational resources is the parsing stage. In order to reduce the computational complexity of parsing, a type of *packing* is used. This mechanism, used in most modern parsing algorithms, treats all chart items, residing within the same chart cell and belonging to the same category, as one node. In the example in figure 2.1 ('A girl gave her cat food'), this is achieved by merging the two VP nodes for 'gave her cat food' into one node, and only combining that merged chart item with 'a girl', the subject of the sentence. Only one chart item is thus created in the topmost chart cell. However, there are still two solutions, and they are obtained by a procedure referred to as *unpacking*, which requires one modification to the packing algorithm: backlinks to all different decompositions have to be created. In the example, that means that the merged VP node contains two decompositions: V/NP/N and V/PRON/NP. The solutions are easily retrieved by following the backlinks, and retrieving the best solution only (which is practical when the solution space is large) is done by only unpacking the most likely decomposition for each node. Because there can be more than one solution, the data structure effectively represents a collection of trees, and is therefore called a *parse forest*.

This method of packing, *equivalence-based packing*, works well for PCFGs, but will be less effective when applied to unification-based grammars, because equivalence of two chart items is bound to be more rare when the descriptions of the chart items are more detailed (although equivalence-based packing is still employed in the Enju parser (Miyao 1999)). Therefore, the PET parser uses a

technique called *subsumption-based packing* (Oepen and Carroll 2000a). In this case, if there are two chart items X and Y, and X subsumes Y, X will function as the representative for the rest of the parsing process (although Y is possibly part of the eventual best solution), because whatever unifies with Y will also unify with X. Y will not be used in the consecutive parts of the parsing process, and is therefore called 'frosted'. Because there are parts of the feature structures that are very unlikely to be subsumed by other structures (most notably semantic features), these can be discarded from the subsumption test (using *packing restrictors*), yielding higher packing rates.

However, this means that retrieving the solutions from the parse forest (unpacking) is not guaranteed to give valid solutions only, and that re-playing the unifications is needed when listing the eventual solutions. For instance, if Y is frosted, and X functions as its representative, and X is used by another rule, unification of that same rule with Y might lead to a unification failure (because Y has more constraints than X, by definition). Therefore, whichever procedure is used to extract the solution from the packed forest, it must be checked whether the resulting derivation tree indeed unifies as expected.

The simplest method for finding the n-best solutions from a packed forest, where 'best' is defined with respect to a certain statistical model, is to enumerate all possible solutions, and consequently score and rank them. The number of solutions can be very high for realistically-sized sentences, though, and extracting all possible solutions is not feasible in that case. To prevent this scenario, Van Noord (2007) presents an unpacking algorithm that can approximate the n best solutions, based on a search beam, without enumerating the (possibly exponentially large) number of solutions. However, a more recent technique called *selective unpacking* has been devised (Zhang *et al.* 2007b), which can guarantee that the best n solutions can be found. Although the algorithm is expected to find the requested solutions within a short time span, it does have an exponential worst-case time complexity.

Apart from the implementations of packing and unpacking, the PET parser employs a number of other techniques to improve the efficiency of the parser. I will discuss three of these techniques here: the rule filter and the quickcheck filter (Kiefer *et al.* 1999), and hyper-active parsing (Oepen and Carroll 2000b). Although all three methods achieved significant speed-ups, they guarantee that the parser will yield identical results to the basic parsing algorithm. The *rule filter* is a filter of tasks on the agenda, based on the combinations of mother and daughter(s). It is possible that a certain combination of rules will *always* fail because one of the daughters introduces a constraint that is incompatible with a constraint of the mother. These incompatibilities can be computed off-line (during grammar compilation), and are put in a hash table. Parsing tasks that introduce an incompatibility can be easily filtered out by one simple table

look-up. *Quickcheck* is a mechanism that employs the fact that certain parts in the feature geometry of a particular grammar cause a unification failure more often than other parts. These parts can be identified when a small piece of text is parsed, and the n feature paths that fail most often are retained. In a parsing session, these paths are (quick-)checked before every unification. If one of the paths fail, the entire unification is guaranteed to fail as well, and the algorithm will return that result as well, without doing the actual unification.

Oepen and Carroll (2000b) describe a technique, called *hyper-active parsing*, based on the following intuition. Consider a COMPLEMENT-HEAD rule, with the complement on the left side. Normally, the constraints of the complement side are fairly loose, because HPSG dictates that the head defines these constraints. The consequence is that relatively many chart items will match the complement if the head is not filled. In regular, left-to-right active parsing, the complement will be filled first, followed by the head, yielding many active chart items that will fail when the head side of the rule is unified with another chart item. Therefore, control over the order in which the arguments of the rule are filled can be beneficial. Hyper-active parsing defines a specific ordering for all rules, so that certain rules can also be filled right-to-left, which can significantly reduce the number of parsing tasks that the parser has to execute.

Agenda-based chart parsing

Chart parsing based on an agenda of parsing tasks was first introduced by Kay (1980). The PET parser is based on this algorithm. One of the algorithm's main data structures is the *chart*. The chart contains items, which are feature structures associated with a start and end index (i, j). Two types of chart items (or: *edges*) are defined: passive and active. A passive edge is a complete edge: a grammar rule where the daughters (either one or two) have been unified with other passive edges. An active edge, on the other hand, still needs (at least) one of its daughters to be filled before it is considered to be complete. Notice that the algorithm leaves space for rules with a higher arity than 2.

The second large data structure within PET is the *agenda*. This is implemented as a priority queue, meaning that each task in the queue is associated with a certain priority. The agenda is initially filled with tasks during lexical look-up: all lexical items are combined with all rules (regardless of the rule's arity). When the lexical stage has finished, the phrasal stage starts. The parsing stage is a loop, in which the task with the highest priority is popped from the agenda, and carried out. The loop is terminated when either a resource limit is hit (in terms of time, memory or number of passive edges) or the agenda is empty.

Two types of tasks are identified: RULE+PASSIVE tasks and ACTIVE+PASSIVE

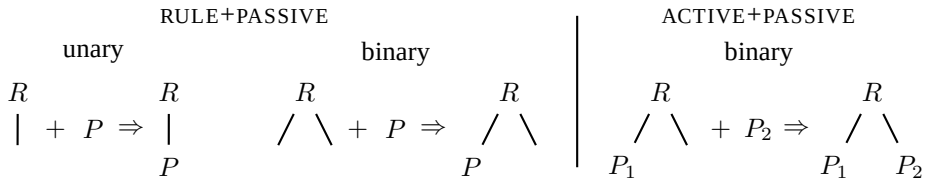


Figure 2.13: Depicted are the different types of tasks in the PET parser. Not shown are the features structures imposed by the rules and the chart items. This figure suggests that the daughters are always filled from left to right, but we have seen before that the other direction is also possible.

tasks. A schematic representation of these tasks is shown in figure 2.13. Due to the precision-oriented nature of the grammars that are usually used with this parser, the execution of a task often fails. If this happens, no subsequent action is taken, and the loop will go to its next iteration. However, if the task succeeds, the resulting feature structure will be put on the chart. This results in new tasks to be inserted into the agenda:

- For each inserted passive item, add RULE+PASSIVE tasks that combine the passive item with each of the rules, and add ACTIVE+PASSIVE tasks that combine with each of the neighbouring active items.
- For each inserted active item, add ACTIVE+PASSIVE tasks that combine the remaining gaps in the active item with existing neighbouring passive items in the chart.

In the standard version of PET, the priority scores only have a heuristic meaning, intended to create a (depth-first) right-corner parser (formulas taken from the actual code):

$$Pr_{passive}(i, j, n) = j - \frac{i}{n} \tag{2.5}$$

$$Pr_{active}(i, j, n) = j - \frac{2i}{n} \tag{2.6}$$

where i and j are the start and end indices of the task's span respectively, and n is the length of the sentence. The heuristics have the effect of a right-corner parser. Given a sentence with length 3, (passive) task spans are ordered as follows: (0,3), (1,3), (2,3), (0,2), (1,2), (0,1).

2.7 Motivation

Overseeing the landscape of parsing using hand-written grammars, the largest issue of this paradigm is *scalability*. It takes many person years to create a

grammar with reasonable coverage, and the person years themselves are costly, as very skilled persons are needed. Also, skilled as the grammar engineer might be, it is hard to convey all intricacies of the grammar in documentation. The consequence is that, if the maintainer of the grammar decides to quit, a possible successor will need relatively much time to get himself acquainted with the grammar.

Treebank-based deep grammars are supposed to circumvent the issues hand-written models of language suffer from. By using simpler grammar models, coverage can be reached quickly, and the workflow is more insightful and reproducible than for parsers based on hand-written grammars. DGE parsers have produced state-of-the-art results, also because advanced statistical models could be trained from the converted treebanks with relative ease. However, they have a number of disadvantages in common, when compared with their hand-written siblings:

Generation from these models is not straightforward

Unlike parsing, which can be computed in polynomial time, generation/realisation from a semantic form to an utterance can only be done in exponential time. Therefore, a precise description of the language by the grammar is necessary in order to keep the computational requirements within reasonable bounds, when an exhaustive algorithm is used[17]. Also, the definition of a precise model of the language prevents the generator to yield ungrammatical utterances.

The derivation algorithms are biased towards configurational languages

The algorithms presented in section 2.4.2 rely on the fact that the underlying representation is a projective tree. However, treebanks for less configurational languages, such as the Tiger treebank for German (Brants *et al.* 2002), are often not in the form of projective trees, and the authors of DGE studies for German (Hockenmaier 2006; Cahill *et al.* 2005) have coerced the annotations in this treebank into projective trees, potentially losing information[18].

Adding linguistic knowledge to the models is not principled

The level of depth or precision of the resulting parsers is limited by the level of depth in the treebank. Creating a grammar that is deeper than what the treebank supports natively always involves making finer distinctions than the ones that the treebank makes. Currently, these distinctions are made

[17] Actually, studies on realisation from treebank-derived deep grammars, using non-exhaustive algorithms, have been performed by Nakanishi *et al.* (2005) and Cahill and Van Genabith (2006).

[18] One might attribute the worse results to the language or treebank being harder to analyse (Kübler *et al.* 2006). However, results from experiments in statistical dependency parsing suggest that this is not true (Hall and Nivre 2008).

by heuristic algorithms, but an approach that has a better grounding in a linguistic theory would be desirable.

In the methodology I propose in the first half of this thesis (chapters 3, 4 and 5), I aim to combine the merits from both grammar engineering paradigms. Roughly, the methodology can be summarised as: automate when possible, engineer manually when necessary.

First, the construction of a core grammar will be discussed (chapter 3). It gives a detailed account of the language at hand (German), including the relatively free word order (known as *non-configurationality*) German is known for. Also, several kinds of long-distance dependencies are covered by the grammar. Over-generation is prevented as much as possible. The core grammar also includes a core lexicon, containing syntactically and semantically idiosyncratic lexemes, which are hard to learn. Because the core lexicon only contains function words, the core grammar has virtually no coverage. The open class lexicon is obtained in a symbolic (*i.e.* non-probabilistic) deep lexical acquisition (DLA) step (chapter 4), derived from the annotations in the Tiger treebank. Learning curves will be presented for different parts-of-speech, so one can estimate whether the lexicon has converged to a stable state. Because the algorithm is simple and transparent, it is easy to reproduce, and wide coverage is reached within little development time, therefore improving on the poor scalability of hand-written grammars.

Statistical models are indispensable for any state-of-the-art parser, and the presence of a treebank is a prerequisite for the creation of these models. Creating treebanks manually (or even computer-assisted) is a costly enterprise, adding up to the total cost of hand-written parsers. In the approach I am taking (chapter 5), the text from the Tiger corpus is parsed using the grammar from the previous chapters, and the readings are automatically disambiguated using the information in the original treebank. Again, this is a large reduction of the needed development time, compared to treebanks that are constructed manually. The rest of the chapter is devoted to the performance of the grammar/parser on unseen text, when a statistical disambiguation model is learnt from the treebank using existing technology.

Chapters 2-5 constitute one coherent piece: it explains how the combination of grammar and treebank can be constructed with the help of the Tiger treebank. Although the individual parts of this process have been discussed, the Interlude will discuss how the entire workflow compares to other grammar engineering paradigms (parsers based on hand-crafted grammars, DGE parsers) in a number of respects.

In the second part of the thesis, the grammar is used as it has been developed in the first part. It turns out that the obtained grammar suffers from some of the

weak points as other deep grammars: low efficiency and lack of robustness. In chapter 6, efficiency will be improved using a novel pruning algorithm. Instead of focusing on pruning chart items, the algorithm prunes tasks from the agenda. A PCFG model of HPSG rule applications is used to compute a new priority model for the agenda in PET, such that promising tasks are carried out before tasks that are less likely to contribute to the best analysis. Different counting and pruning strategies will be explored, as well as a combination of simple supertagging and phrasal restriction.

Chapter 7 will be devoted to improvements of the parser on the issue of robustness. This is achieved by the introduction of hand-written, over-generating robustness rules. The pruning algorithm developed in chapter 6 will be used to restrict the search space appropriately, in order to minimise the negative consequences in terms of efficiency. This method will be compared to the widely employed strategy of combining existing fragments of the chart.

The last chapter will be used to wrap up the research that has been carried out for this thesis. A summary of the conclusions will be presented, as well as directions for future research.

3 Core grammar construction

As mentioned at the end of the background chapter, chapters 3, 4 and 5 form one coherent part of this thesis, and covers the construction of a wide-coverage HPSG grammar for German, including a compatible treebank. This chapter provides more details about the *core grammar*, containing all the elements of a grammar, but without a lexicon for open word classes (which will be created in chapter 4). I will start this chapter by outlining a number of basic properties of the German language. This is followed by a short summary of existing HPSG accounts of German. However, I will also explain why some of these accounts can not be implemented in the DELPH-IN formalism. Next, the basic structure of the grammar will be outlined: its type hierarchy, its rules, its lexical types and its core lexicon.

3.1 The German language

This section is meant to give the reader an impression of the German language. There is no space for an exhaustive discussion about all of the language's intricacies. The decision to include a phenomenon/construction or not in this section depends mostly on whether the core grammar in section 3.3 makes an attempt to correctly model a certain aspect of German.

One of the most widely used classification schemes for languages is the canonical order of subjects (S), verbs (V) and objects (O). Six different permutations of these constituents can be identified: SVO, SOV, VSO, VOS, OSV and OVS. English, for instance, is generally considered to be an SVO language:

(10) John eats cheese.
S V O

This is only a very crude characterisation of a language and such a label is more of a word order preference than a strict rule. Even English, with a fairly rigid word order, has counterexamples to the SVO order:

(11) Cheese, John eats.
O S V

German behaves as a V2 language in declarative sentences, meaning that the word order can be either SVO or OVS. Adjuncts can be placed before the verb as well, pushing both subject and object after the verb (with a fairly strong preference of placing the subject before the object).

(12) Antje isst gerade den Käse.
Antje eats currently the cheese.

(13) Den Käse isst Antje gerade.
The cheese eats Antje currently.

(14) Gerade isst Antje den Käse.
Currently eats Antje the cheese.

The word order is different when the utterance is a question or a command. In these cases, the verb is put in first position:

(15) Isst Antje gerade den Käse?
Eats Antje currently the cheese?

(16) Iss den Käse!
Eat the cheese!

On the other hand, the verb ends up last in a clause, when that clause is a subordinate of another verb:

(17) Ich denke, dass Antje den Käse isst.
I think, that Antje the cheese eats.
I think that Antje is eating the cheese.

All the examples above assume that there is only one verb in a clause. This is not the case for many sentences, due the presence of auxiliary verbs. If an auxiliary verb is present in a clause, the position of the main verb (the one that has its person and number matched with the subject of the clause) follows the rules given above, but the other verbs move into a *verb cluster*, at the end of the clause:

(18) Antje hat den Käse gegessen.
Antje has the cheese eaten.
Antje ate the cheese.

(19) Antje würde den Käse gegessen haben.
Antje would the cheese eaten have.
Antje would have eaten the cheese.

(20) Ich denke, dass Antje den Käse gegessen haben würde
I think, that Antje the cheese eaten have would
I think, that Antje would have eaten the cheese.

In the large majority of cases, the word order within the verb cluster is determined by the following rule: the dependent verb ('gegessen' in the last example) is on the left of its immediate dominator ('haben'). However, there are some cases where this does not hold, usually observed in subordinate clauses. This is called the 'auxiliary flip' or *Oberfeldumstellung*:

Vorfeld	Left bracket	Mittelfeld	Right bracket	Nachfeld
1	1	∞	∞	∞
Subject	Conjugated verb	Subject	Verb cluster	Extraposed material
Object	Complementiser	Object	Verb particle	'zu'-clauses
Adjuncts		Adjuncts		Subordinate clauses

Table 3.1: This table gives a summary of how constituents can be placed in different topological fields. The labels 1 and ∞ indicate how many constituents can be placed in that particular field.

(21) Ich denke, dass Antje den Käse wird essen können.
I think, that Antje the cheese will eat can.
I think that Antje will be able to eat the cheese.

The order of verbs and objects in German is commonly explained using a model of *topological fields*[1]. The field of the first main verb is called the *left bracket* ('die linke Klammer'), and the verb cluster is placed in the *right bracket* ('die rechte Klammer'). The position before the left bracket is called the *Vorfeld* ('forefield'), the positions between the left and right bracket the *Mittelfeld* ('middle field'), and the positions after the right bracket the *Nachfeld* ('afterfield')[2]. Verb-final clauses are considered to have an empty *Vorfeld*, and a possible complementiser (such as 'dass'/'that' or 'ob'/'whether') is located in the left bracket. Table 3.1 summarises how different kinds of constituents can be distributed over the different topological fields in each clause[3].

One interesting property of German is that subject, objects and adjuncts can move relatively freely across the *Vorfeld*, the *Mittelfeld* and sometimes even the *Nachfeld*, as in examples 23-25, with a strikingly small effect on the meaning of the sentence (permutation of constituents in the *Mittelfeld* is sometimes referred to as *Mittelfeld scrambling*[4]). There is, however, a preference to place the subject before the object, so 'den Käse' in example (26) should be read to have more focus.

[1] Actually, these topological fields are explicitly annotated in one of the treebanks of German, NEGRA (Skut *et al.* 1997).

[2] As the German names *Vorfeld*, *Mittelfeld* and *Nachfeld* are so habituated, I will use these instead of the respective English translations throughout this thesis.

[3] More fields are identified by some authors. See for instance (Kathol 1995) for an overview.

[4] There may be confusion of terminology here. I use the phrase '*Mittelfeld* scrambling' for different permutations of constituents within the *Mittelfeld*. This is different from the phenomenon that is sometimes referred to as 'extraction into the *Mittelfeld*', in which a constituent is extracted out of a VP into the *Mittelfeld*, as in the following example:

(22) Ich plante über Maria gestern ein Vortrag zu halten.
I planned about Maria yesterday a presentation to give.
Yesterday, I planned to give a presentation about Maria.

The interesting thing about this example is that 'über Maria' modifies the noun 'Vortrag' but is extracted out of the verb phrase. See for instance (Müller 1997) for more details on this phenomenon. The core grammar that is developed in this chapter will not be able to cover this type of extraction.

(23) Antje hat gestern den Käse gegessen.
Antje has yesterday the cheese eaten.

(24) Antje hat den Käse gestern gegessen.
Antje has the cheese yesterday eaten.

(25) Gestern hat Antje den Käse gegessen.
Yesterday has Antje the cheese eaten.

(26) Den Käse hat Antje gestern gegessen.
The cheese has Antje yesterday eaten.

The *Nachfeld* is used for a wide variety of clauses. Among those, the most important are extraposed relative clauses, subordinate clauses, sentential complements and verb phrase complements, as displayed respectively in the next examples:

(27) Antje hat den Käse gegessen, den Bernd empfohlen hat.
Antje has the cheese eaten, that Bernd recommended has.
Antje ate the cheese that Bernd recommended.

(28) Antje hat gemeint, dass Bernd den Käse empfohlen hat.
Antje has thought, that Bernd the cheese recommended has.
Antje thought that Bernd recommended the cheese.

(29) Antje hat gemeint, Bernd hätte den Käse empfohlen.
Antje has thought, Bernd has the cheese recommended.
Antje thought that Bernd recommended the cheese.

(30) Antje hat versucht den Käse zu essen.
Antje has tried the cheese to eat.
Antje tried to eat the cheese.

Except for sentential complements, constituents in the *Nachfeld* can be placed towards the *Mittelfeld* (32). Also, non-finite clauses can be fronted to the *Vorfeld*, and can take all (33), none (34) or some (35) of its arguments with it:

(31) Antje hat versucht ihm den Käse zu schenken.
Antje has tried him the cheese to give.
Antje tried to give him the cheese.

(32) Antje hat ihm den Käse zu schenken versucht.
Antje has him the cheese to give tried.
Antje tried to give him the cheese.

(33) Ihm den Käse zu schenken hat Antje versucht.
Him the cheese to give has Antje tried.
Antje tried to give him the cheese.

(34) Zu schenken hat Antje ihm den Käse versucht.
To give has Antje him the cheese tried.
Antje tried to give him the cheese.

(35) Den Käse schenken wird er seiner Tochter.
The cheese give will he his daughter.
He will give the cheese to his daughter.

German has a moderately complex morphology. Verbs are inflected for tense, number, person and mood[5]. Like English, both regular and irregular verbs are observed. Nouns exhibit gender, case and number. Most word forms of a lexeme are equal to the lexeme's base form. The inflection of a noun depends on the noun's declension class. The subject and the main verb have to agree with respect to person and number. In contrast to English, noun compounding is done by concatenating the two nouns: 'information theory' versus 'Informationstheorie'. Adjectives and nouns have to agree with respect to gender and number and in 'strongness', the latter being determined by the (non-)presence and type of determiner used in the noun phrase. Roughly, this means that case information needs to be provided by the determiner, but if there is none, the adjective will exhibit this information.

A peculiarity is the use of separable particles. For example, the verb 'auswählen' consists of the particle 'aus' and the verb 'wählen', and means 'to pick'. In non-finite forms (see example 36), the particle prefixes the verb, as well for finite verbs that are located in the right bracket (in subordinate and relative clauses; see example 37). When a finite verb resides in the left bracket, however, the particle remains in the right bracket (example 38). In case the verb is in the *zu*-form (to-form in English), 'zu' is placed as an infix between the particle and the main verb (example 39).

(36) Antje will den Käse auswählen.
Antje wants the cheese out-choose.
Antje wants to pick the cheese.

(37) ..., dass Antje den Käse auswählt.
..., that Antje the cheese out-chooses.
..., that Antje picks the cheese.

(38) Antje wählt den Käse aus.
Antje chooses the cheese out.
Antje picks the cheese.

[5] *Mood* is divided into three categories in the Tiger treebank (Brants *et al.* 2002): indicative, subjunctive (German: *Konjunktiv*) and imperative. The latter is used when the verb is a command ('Sit!'). In English, the indicative mood is mostly used, and the subjunctive form is hardly used, except in sentences such as 'Had I a hammer, I would hit the nail.'. However, the subjunctive mood is more widely used in German.

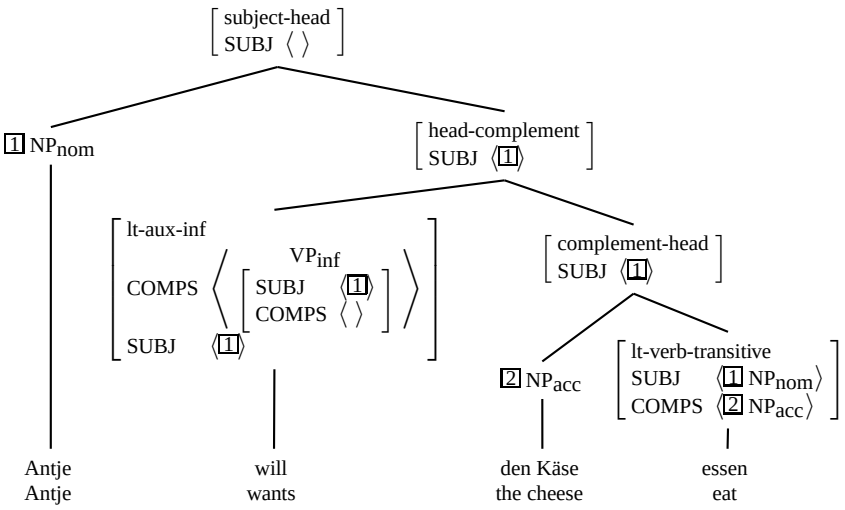


Figure 3.1: A naïve HPSG analysis of the German sentence 'Antje will den Käse essen'.

(39) Antje versucht den Käse auszuwählen.
Antje tries the cheese out-to-choose.
Antje tries to pick the cheese.

3.2 HPSG analyses of German

There is a large literature on HPSG analyses for German (Nerbonne *et al.* 1994). Many of them focus on the language's word order, as it turns out to be hard to create a formal model of this aspect of the German language. We are looking for an analysis that not only models the correct word order (and rejects incorrect ones), but also performs the semantic composition properly. Let's first take a look at the naïve analysis of a simple German sentence in figure 3.1, assuming the HPSG schemata and principles, as introduced in section 2.1.2.

Although this analysis is satisfactory, it does not extend to other word orders, for instance when a complement or modifier is in the *Vorfeld*. Consider the sentence 'Den Käse will Antje essen'. First, 'Antje' and 'essen' will connect correctly using the SUBJECT-HEAD rule. The following step would be to let 'Antje essen' be the complement of 'will'. However, the complements list in the feature structure of 'Antje essen' is not empty, and hence, the unification of the HEAD-COMPLEMENT rule will fail.

3.2.1 The HEAD-CLUSTER schema and argument attraction

One of the most widely adopted ideas to address this issue has been proposed by Hinrichs and Nakazawa (1994). The solution consists of two components:

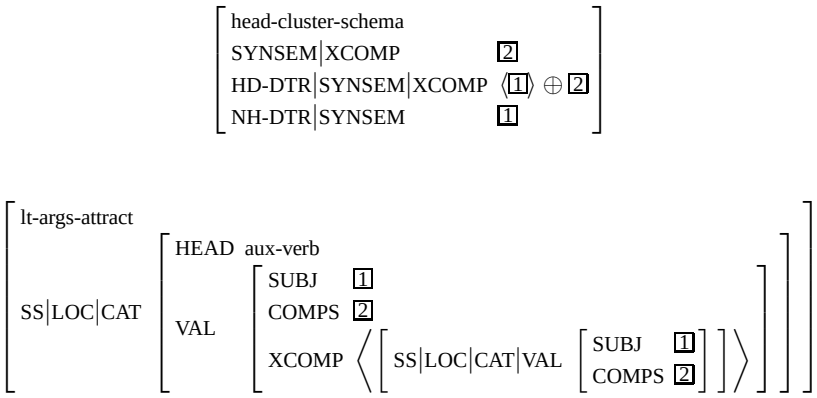


Figure 3.2: Depicted are (a) a representation of the proposed HEAD-CLUSTER schema and (b) the type in the hierarchy that introduces argument attraction. The operator $\oplus$ signifies list concatenation.

- First, the HEAD-CLUSTER schema is introduced. This schema allows an auxiliary to take a VP as a verbal complement, without requiring the VP to have saturated valence lists. The HEAD-CLUSTER schema does not discharge the COMPS list feature, but applies to a newly introduced feature called XCOMP.
- Lexemes that can act as the head of the HEAD-CLUSTER schema (*i.e.* having a non-empty XCOMP list) do not require the argument to have empty subject and complement lists. Instead, it identifies its own valence lists with the remaining subjects and complements, a technique called *argument attraction*. In principle, the auxiliary verb can have a semantic connection to the attracted arguments (for *raising* verbs[6]), but no connection is formed for other auxiliaries.

The definitions of both the HEAD-CLUSTER schema and the LT-ARGS-ATTRACT type are shown in figure 3.2. This rule can be used for the verb cluster itself, but also for the connection of the left bracket with the material on its right side (as for the previous example).

3.2.2 A fronting analysis

It is generally assumed that the first element in the sentence (the *Vorfeld*) is *fronted*, *i.e.* the dependency between the first constituent in the declarative sentence and the deepest verb is considered to be a long-distance dependency. An argument supporting this view is that the verb's arguments are not introduced by

[6]A typical example of raising is: 'Antje tried to eat the cheese', where 'Antje' is both the subject/agent of 'tried' and 'eat'. The lexical type of 'tried' should then make a connection between its own semantic agent slot and the INDEX of the attracted argument.

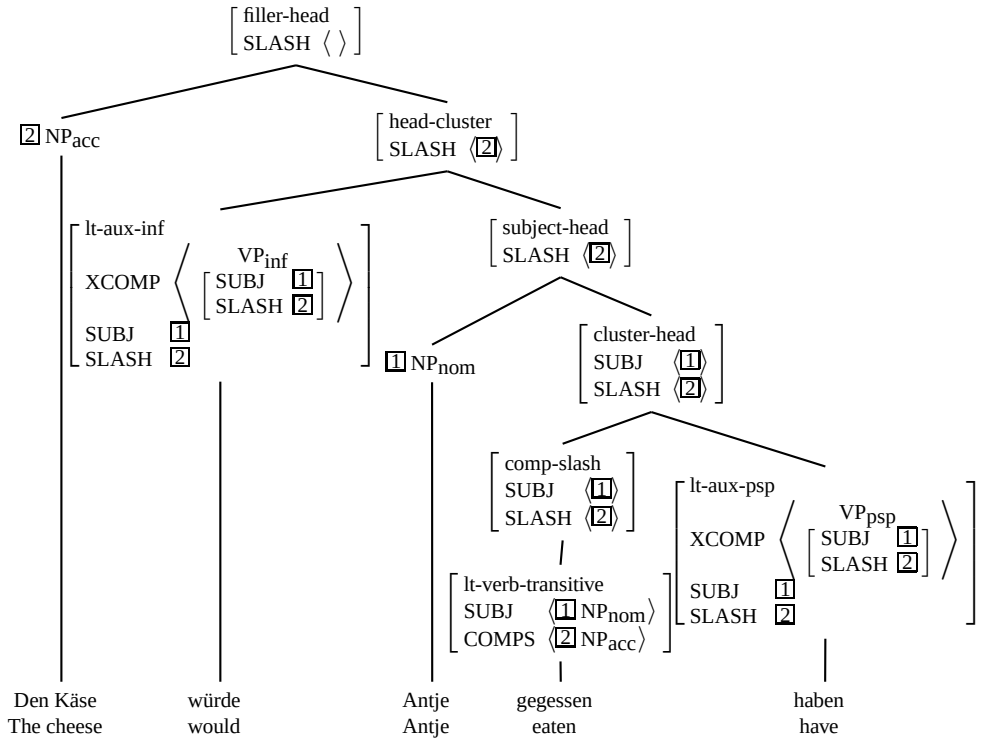


Figure 3.3: An HPSG derivation for the sentence 'Den Käse will Antje gegessen haben'. Notice the differences with the analysis in figure 3.1: argument attraction, the use of the HEAD-CLUSTER schema and the introduction of fronting using the SLASH feature. A translation for this sentence is 'Antje would have eaten the cheese', with the focus of the sentence on 'the cheese'. Notice that the argument attraction is not only carried out with respect to the SUBJ and COMPS features, but also for the non-local feature SLASH.

the auxiliary verb, but by the deepest verb[7]. The fronting analysis thus makes sure that the semantic roles are properly assigned: a fronted object is the argument of the deepest verb, not the auxiliary's. The fronting is achieved by moving the argument to the SLASH feature, and the most straightforward way to do this is by means of a lexical rule. In order to showcase the use of the fronting analysis and the use of argument attraction and the HEAD-CLUSTER rule, an analysis of the German sentence 'Den Käse würde Antje gegessen haben' is given in figure 3.3.

The extraction of a constituent to the *Vorfeld* can also be used for somewhat more complicated (and in principle unbounded) extraction, occurring very frequently in newspaper text. The structure of the sentence is composed of a direct speech construction (X says "Y"), in which one constituent of Y is fronted ("Y_1", X says, "Y_2"). The analysis we give is uses a very specialised rule (SLASH-ESCAPE). This rule combines two incomplete VP phrases: the first needs a sentence as its complement, the second needs the extracted constituent. No constraints are put on the extracted constituent. The SLASH-ESCAPE rule

[7]One can dispute which verb fronted adjuncts modify. This uncertainty is also expressed in the Tiger treebank, in which both levels of modification are found.

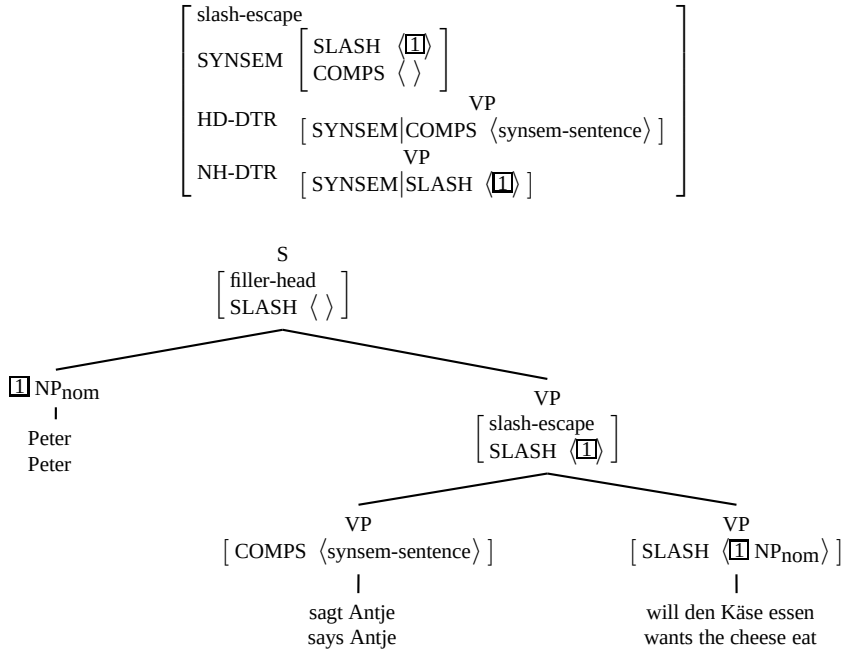


Figure 3.4: This figure gives the basic structure of the SLASH-ESCAPE rule, and a simple example of how this rule is used. A translation for the sentence in the example is ‘ “Peter”, Antje says, “wants to eat the cheese” ’.

and an example of its usage are given in figure 3.4.

3.2.3 Complement extraposition

Apart from leftward extraction, German also exhibits some forms of extraction in the opposite direction. The analyses used in the core grammar are different for the extraction of complements and modifiers. I will start with the former. Consider the following sentence:

(40) Antje hat die Neigung gehabt den Käse zu essen.
Antje has the urge had the cheese to eat.
Antje has had the urge to eat the cheese.

In this example, the noun ‘Neigung’ has a non-finite verbal complement, in this example realised as ‘den Käse zu essen’. However, there is a verb between the two constituents, and therefore the noun’s complement should be moved to a non-local feature by means of a lexical rule. The obvious location would be SLASH, but this could interfere with the leftward extraction. Therefore, a new non-local feature RSLASH is brought into existence. Discharging the RSLASH feature is done using the RFILLER-HEAD rule, analogous to the standard FILLER-HEAD rule. An analysis for example 40 is given in figure 3.5. This mechanism can also be used for the rightward extraction of other NP complements, such as wh-phrases and complementiser phrases:

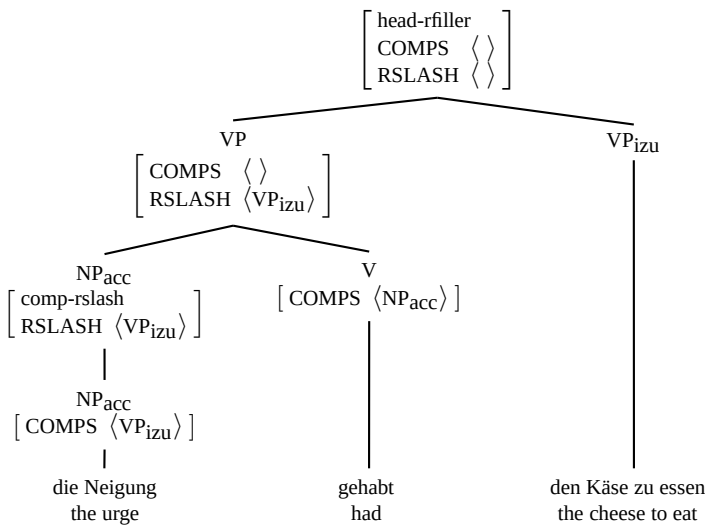


Figure 3.5: This figure depicts the relevant portion of the HPSG analysis for the sentence 'Antje hat die Neigung gehabt den Käse zu essen'.

(41) Antje wird keine Ahnung haben wo der Käse ist.
Antje will no idea have where the cheese is.
Antje will have no idea where the cheese is.

(42) Antje wird die Eindruck haben, dass Bernd den Käse isst.
Antje will the impression have that Bernd the cheese eats.
Antje will have the impression that Bernd eats the cheese.

3.2.4 Adjunct extraposition

Extraposition of adjuncts to the *Nachfeld* is done in a different way, and the analysis I propose is heavily indebted to the one proposed by Crysmann (2005). The most prominent form of adjunct extraposition is relative clause extraposition:

(43) Antje hat sich über den Käse gefreut, den Bernd gegessen hat.
Antje has REFL at the cheese rejoiced, that Bernd eaten has.
Antje looked forward to the cheese that Bernd ate.

The basic idea behind the analysis is to collect all possible *anchors* in the ANC feature when creating non-verbal constituents. The type of each anchor is *ref-index*. When a constituent is connected to the verb, the set of anchors is made visible to the extraposed material, so they can be combined. This combination is achieved using a newly introduced ANC-BIND rule[8]. In order to ensure the compatibility between the material in the *Nachfeld* and the anchor, the subfeatures

[8] As said earlier, the PET parser does not support sets. Therefore, the collection is encoded as a list, in such a way that the rightmost anchor is the head of the list. There are n BIND-ANC rules to connect the extraposed material to the i-th element in the list. Small experiments showed that no gains were

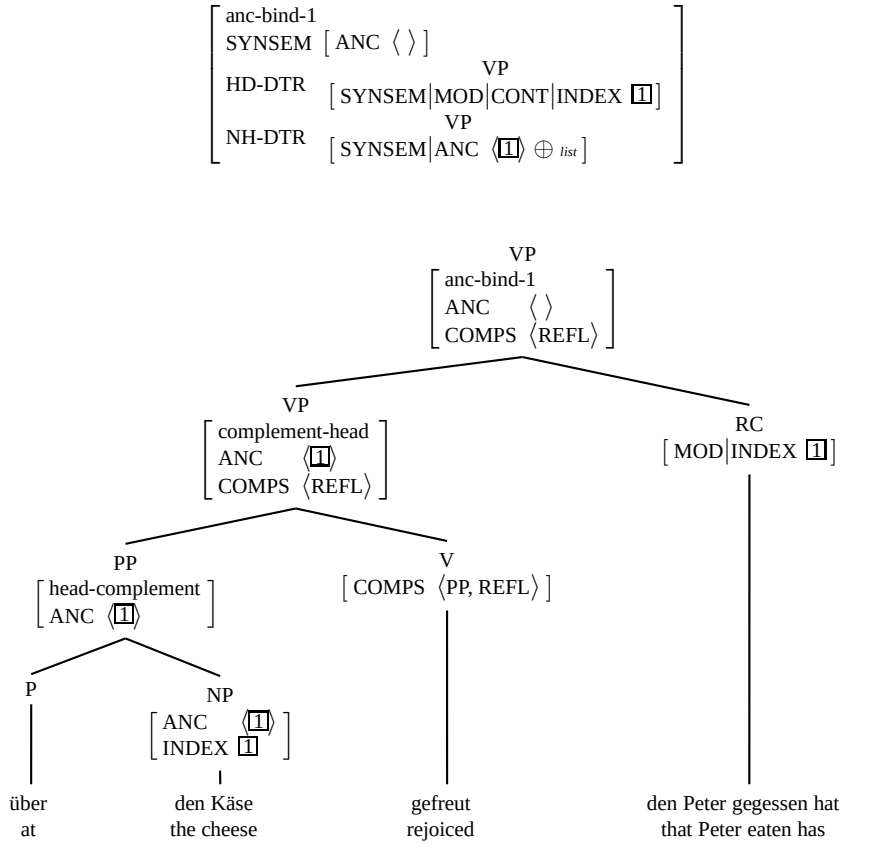


Figure 3.6: This figure illustrates how an extraposed adjunct (a relative phrase in this case) can be connected to an anchor in the *Mittelfeld*. At the top, a definition of the ANC-BIND rule is given. Below, a (partial) analysis is given for the sentence in example 43. The analysis shows how a collection of anchors is built up within the non-verbal constituent (in this case a PP, containing only one anchor), and how it is connected using the ANC-BIND rule.

for gender and number within the anchor are matched against the requirements in the MOD feature of the extraposed material. This is particularly important for relative phrases, where the relative pronoun's gender and number need to be equal to the gender and number of the noun that the relative phrase modifies. An analysis for example 43 is given in figure 3.6.

The original paper (Crysmann 2005) only focuses on the extraposition of relative clauses, but it became apparent that this analysis is applicable to the extraposition of comparative phrases as well (if one includes the adjectival INDEXes as well in the list of anchors):

(44) Antje denkt dass dein Käse herrlicher ist als mein Käse.
Antje thinks that your cheese more-delicious is than my cheese.

achieved with $n > 3$. Another implementation detail is that the PET parser can not maintain lists that are used for both collection and discharging. Therefore, the feature ANC-ALL is used for collection, and when the anchors are taken up by the verb phrase, they are moved to the ANC-ACTIVE feature, which can be discharged by the ANC-BIND rules.

Antje thinks that your cheese is more delicious than my cheese.

3.3 Implementing a core grammar for German

While the previous section shows how the analyses will eventually look like, this section goes into more detail on how a grammar can be written in the $\mathcal{TDL}$ framework (Krieger and Schäfer 1994) to reach these analyses. As has been discussed earlier, the grammar implementation should be able to cover a number of linguistic phenomena, while constraining local ambiguity and over-generation. Also, the grammar should be written in a compact and well-structured manner, in order to keep the grammar understandable, and thus maintainable.

3.3.1 Basic building blocks

Many grammars within the DELPH-IN framework are extensions of the language-agnostic Matrix grammar (Bender *et al.* 2002) or its customisation kit (Bender and Flickinger 2005). However, due to the history of the development of the project, the core grammar in this chapter does not use it as its starting point. The initial intention behind the project was to use Miyao *et al.*'s (2004) technique to create a grammar for German, using the Tiger treebank as the source treebank. In this stage, the Matrix grammar would introduce too much complexity, and for many phenomena, additional grammar engineering would be necessary. Hence, the additional value of using the Matrix grammar was not large enough. However, it quickly became apparent that a conversion of the source treebank into HPSG derivation trees, using heuristic conversion rules only, would yield poor results[9]: either the core grammar would remain rudimentary, and the grammar would suffer from unacceptable over-generation and would model the German language unsatisfactorily, or more effort would be put into grammar engineering, with a linguistically more appealing grammar as a result. As the project proceeded, the second solution received more attention, and more Matrix-compliance was achieved as a side effect, in terms of the naming of features, most notably for the semantics. However, the grammar never reached full compatibility.

As explained in section 2.6.1, the type hierarchy is one of the key structures of a grammar. Lexemes and rules are just instantiations of the types in the hierarchy. The top node of the type hierarchy is represented by the *top* type. In the core grammar, all variables, for person, number, mood, tense, verbal forms, inflection classes, agreement information, etc are directly attached to the *top* node. The large majority of these variables are featureless and make heavy use

[9] I will propose another methodology to achieve this conversion in section 5.3.

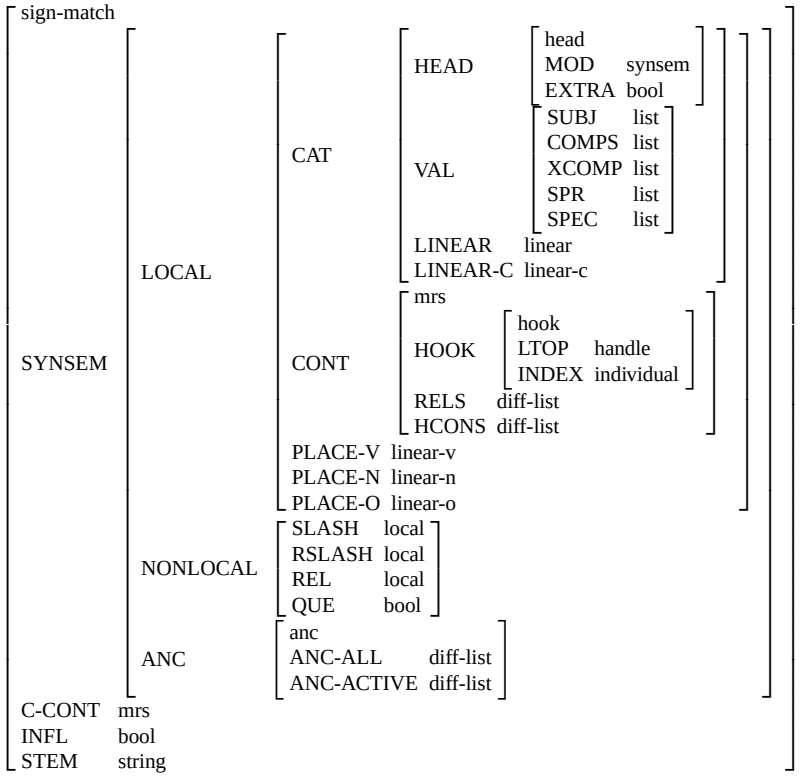


Figure 3.7: This figure shows the basic feature geometry of the core grammar. Most of the features are known to the reader familiar with HPSG, and the XCOMP and ANC features have been discussed in section 3.2. The PLACE-X and LINEAR features will be introduced later in this section.

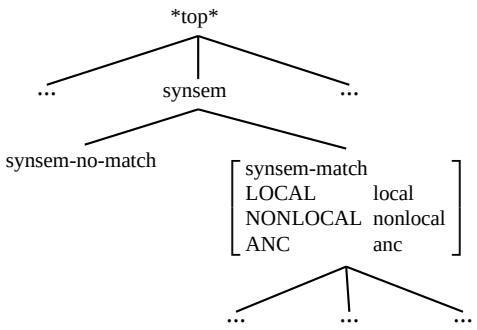


Figure 3.8: Three variants of *synsem* are defined. *synsem* is the top of this subhierarchy, but leaves undetermined whether it should be a *synsem-match* or *synsem-no-match*, and defines no subfeatures. *synsem-no-match* indicates that it should not match a *synsem-match*. The features that are normally associated with a *synsem* are introduced by the type *synsem-match*.

of multiple inheritance, as advocated by Flickinger (2000). Most other daughters of the top node are familiar categories from HPSG: *sign*, *synsem*, *local*, *nonlocal*, etc. Because one type (e.g. *local*) introduces the other (e.g. *cat*), a minimal feature structure can be read off from the type hierarchy. The feature geometry of the core grammar is shown in figure 3.7.

In a number of cases, introducing all subfeatures for a type is not necessary, and hiding these subfeatures will yield feature structures that are easier to read and, as a side effect, speed up computations. Also, sometimes one needs to express that a certain type should *not* be there. To accomplish this behaviour, some types are organised as *synsem* in figure 3.8.

Seven types of heads are identified in the core grammar: verbs, nouns, determiners, adjectives, adpositions, adverbs, and other. Some of the head types receive additional features. For example, verbs have a feature to indicate whether the verb is an infinitive, a finite verb, a participle, an imperative or a 'zu'+infinitive (comparable to the English 'to'+infinitive). Nominal heads display whether the NP contains a determiner, and whether this head can be followed by an apposition. The powerset of the heads is defined as well, allowing to select for any disjunction of heads. Some head features are defined for multiple heads. For instance, DEGREE is a useful feature for both adjectives and certain adverbs, and agreement features are added to verbs, nouns, adjectives and determiners.

A subhierarchy of the *sign-match* type is shown in figure 3.9. Roughly, this type divides into a few streams. For instance, all inflectional rules derive from *lex-infl-rule*, all lexical rules (following the inflection) stem from *lex-deriv-rule*, and all phrasal rules stem from *phrase*, of which some of the subtypes are depicted in the figure as well. All lexical items derive from *lex-type*. Not shown are the constraints that the types introduce, which take care of the creation of MRSs, introduce HD-DTR and NH-DTR for headed structures, and make sure

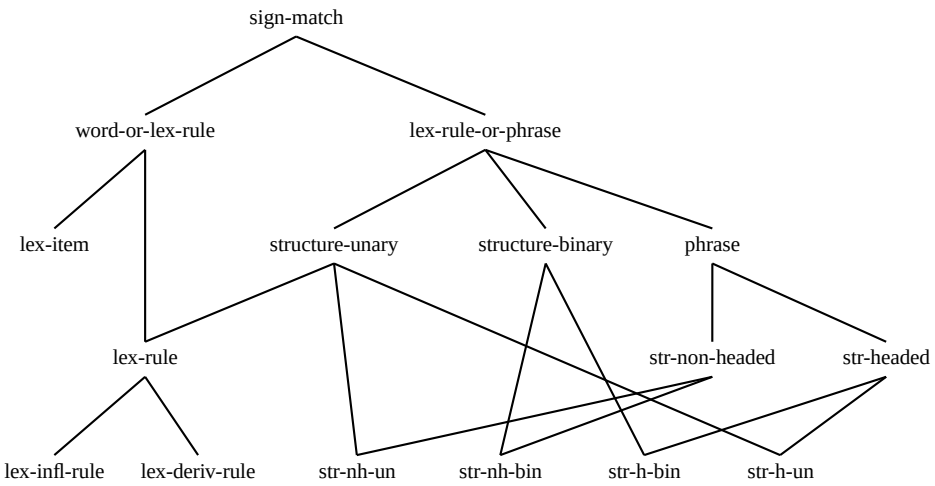


Figure 3.9: The subdivision of the sign type into lexical types and several sorts of rules. For space reasons, some abbreviations are used: 'str' for structure; 'h' and 'nh' for headed and non-headed, respectively; 'un' and 'bin' for unary and binary, respectively.

that inflected words can not be inflected again.

3.3.2 Lexical types

In earlier studies (Cahill *et al.* 2005; Miyao *et al.* 2004), the lexical types (or dictionary templates) were created dynamically. The advantage of this approach is the reduced development time, and there is the possibility that the algorithm finds lexical types of which the grammar writer might not have assumed that they existed[10]. Because this study exhibits a more constructionalist approach to some of German's non-configurational properties than Hockenmaier (2006), the permutation of constituents in the (*Mittelfeld* is taken care of by rules, not separate lexical entries), one can expect a more overseeable set of lexical types. Therefore, the set of lexical types is defined by the grammar writer, and not by the algorithm. Excluding non-auxiliary verbs, 94 lexical types are defined.

This does not entail that no form of automation can be used in defining the lexical types. Especially for verbs, there is time to gain: all non-auxiliary verbal lexical types are created automatically using a script, enumerating all different

[10]There are a number of objections against this argument. Often, these lexical types are very rare, and it is not likely that the resulting grammar will profit from having learnt this type, also because the disambiguation model will not favour them in the analyses. There is also a significant risk that the surprising lexical type is an error, introduced by the algorithm or the treebank. A related issue is that, exactly because the phenomenon is so infrequent and idiosyncratic, it is likely that the annotations in the treebank will be conflicting. This will yield an inconsistent grammar, which leads to unnecessary ambiguity.

combinations of subject types (nominative NP, expletive 'es', CP), and all possible sets of complements (NPs in different cases, CPs, PPs, wh-phrases, direct speech, etc) up to size three. Smart typing takes care of agreement of between subject and conjugated verb, between subject and the reflexive (if one is defined by the type), and the proper composition of arguments in the MRS. In total, 897 possible combinations of complements for non-auxiliary verbs created in this manner.

A special kind of lexical types is formed by *generic lexical types*, which is used by PET's unknown word handling mechanism, usually subtypes of existing lexical types. For instance, if an unknown word is seen, its (guessed) part-of-speech tag can be used to spawn generic lexical types in the parse chart. The core grammar contains generic lexical types for 12 different head types.

3.3.3 The core lexicon

Another difference to previous DGE methods is that a *core lexicon* is defined. All lexemes in the core lexicon share some of the following (correlating) characteristics:

- They have idiosyncratic syntactic or semantic properties. That means that it would be hard for a deep lexical acquisition (DLA) procedure to derive what the lexeme's properties are.
- There is no agreement on the linguistic properties for these lexemes. For instance, argument attraction as a way to describe German auxiliary verbs was not a part of the original HPSG proposal, but has been accepted as a good solution later. New advances can be incorporated with ease (and evaluated properly) when a core lexicon has been constructed.
- These lexemes are frequent in usual text. This has the consequence that small improvements to these lexemes will have a relatively large influence on the performance of the grammar at large. Also, errors in the DLA procedure are prevented this way.
- Not many lexemes are connected to one lexical type, indicating that these word classes are fairly closed. For instance, there is only one future auxiliary verb in German.

The core grammar contains approximately 550 lexemes, divided over the following categories:

- Auxiliary verbs (for future and past tense, modal verbs, raising and control verbs)

- Pronouns (demonstrative, personal, reflexive, relative, interrogative, indefinite)
- Determiners (definite, indefinite, demonstrative, possessive, interrogative)
- Adpositions (pre-, post- and circumpositions, preposition/article contractions ('im';'in the'), preposition/adverb contractions ('wobei'; 'where by'))
- Complementisers
- A small number of special adverbs for comparative constructions
- A few multi-word adverbs

3.3.4 Semantics vs syntactic dependencies

All types concerning the semantics (in the format of Minimal Recursion Semantics (Copestake *et al.* 2005)) are taken over verbatim from the Matrix grammar. Put very briefly, an MRS can be regarded as a set of semantic frames (relations), carrying a predicate, and defining zero or more roles (ARG1, ARG2, etc), each of them filled by other frames. The distinctive feature of the MRS formalism is the ability to underspecify semantic scope. For instance, regard the following sentence (a), with two distinct meanings (b) and (c):

(45) a. Every dog chases some cat.
b. All dogs chase the same cat.
c. All dogs chase a different cat.

The two meanings can be represented in predicate logic as follows:

$\exists$y $\forall$x dog(x), cat(y), chase(x, y)
$\forall$x $\exists$y dog(x), cat(y), chase(x, y)

However, it can be summarised in one MRS formula, leaving the scope of both quantifiers ('every', 'some') underspecified:

$h1$:every(x, $h3$, $h8$), $h3$:dog(x), $h4$:chase(x, y),
$h5$:some(y, $h7$, $h9$), $h7$:cat(y)

Equating $h8 = h4$ and $h9 = h1$ yields the first reading, where the existential quantifier outscopes the universal one. On the other hand, if $h8 = h5$ and $h9 = h4$, the second meaning is realised. By not equating any handles, both meanings (and no other meaning) can be embedded in one reading. See (Copestake *et al.* 2005) for more details.

The implementation of the formalism in the DELPH-IN framework reveals a number of other differences between MRS structures and standard syntactic representations. For instance, observe figure 3.10, which gives the ERG (Flickinger

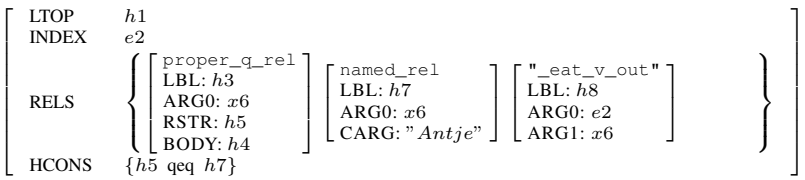


Figure 3.10: The MRS as output by the ERG on the sentence 'Antje is eating out'.

2000) output of the sentence 'Antje is eating out'. It is obvious that there is not a one-to-one relation between words and relations. First, some words are considered to be semantically void. For instance, the verb 'eat out' is made up of two words, but only introduces one semantic relation. The particle 'out' has a lexical entry, but does not introduce a semantic relation, as it is assumed that the verb that takes up the particle carries all semantic content. On the other hand, some words are directly related to more than one relation. In the example, the relations with predicate names `proper_q_rel` and `named_rel` are introduced by the name 'Antje' directly. Another difference is that two types of relations are defined: normal and special relations. The first are language-specific, and are usually connected to open word classes. In the format in the example, these are embraced in double quotes. Special relations introduce meaning that is not entirely lexical. The quantification relation in figure 3.10 is an instance of such a relation.

However, the extra assets of the MRS formalism compared to syntactic dependencies will not be used in this thesis (although the active/passive distinction is made). There are two reasons for this design choice. The first is that a large part of the extra information is not included in the Tiger treebank, the most important being the difference between scopal and intersective modifiers. The consequence is that the deep lexicon acquisition procedure can not properly derive its lexical entries, most notably the adverbs. Hence, the resulting full grammar would be incorrect. In practice, this means that the LTOP features in HOOK and the LBL features in the individual relations remain underspecified. HCONS will always be an empty list.

The second reason to not use the entire expressiveness of the MRS formalism is that there would be no way to compare the grammar's output against the Tiger treebank, and hence, there would be no gold treebank. Evaluation could then only be done by treebanking a small corpus by hand. On the other hand, the availability of a straight comparison against the Tiger treebank will create large advantages, which we will exploit in later chapters: the dependencies can be used for unit testing (section 5.2), and a large-scale dynamic HPSG treebank can be created automatically (section 5.3).

3.3.5 Morphology

As we have seen in section 3.1, German is considered to be a language with a moderately complex morphology, for three parts-of-speech: verbs, nouns and adjectives. German morphology is for a large part predictable from a word's stem and the desired morpho-syntactic properties, and the PET parser supports morphological rules based on regular expressions. However, to simplify the DLA process, the core lexicon (and the lexical entries from the DLA step) assume that all verbs are irregular, which can be represented as (word form, inflectional rule, stem) triples. In total, 99 inflectional rules are devised in such a way that maximal generality of the rules is achieved (i.e. that one rule can cover more than one cell in the inflection table). The following is an example of an inflectional rule:

```
ir-verb-npd-pr-13-pl-k :=
%suffix (xyx xyx)
lex-infl-rule-verb-npd-fin &
[ SYNSEM.LOCAL.CAT.HEAD.AGR agr-pr-13-pl-k ].
```

This inflectional rule will specialise the verb's AGR feature: after application of the rule, the lexical item has present tense, is either first or third person plural, and is in the subjunctive mood (German: *Konjunktiv*). The suffix part is present for technical reasons, otherwise the parser will not recognise the rule as an inflectional rule; the suffix is crafted such that it will never match. The 'npd' ('no particle deletion') infix in the parent type indicates that no particle needs to be deleted from the COMPS list, because it is already contained in the verb. Hence, there are two different entries in the inflections lists for the verb 'ablehnen' ('to reject') with the same AGR feature:

```
ablehnt IR-VERB-NPD-PR-3-SG-I ablehnen
lehnt IR-VERB-PD-PR-3-SG-I ablehnen
```

For the core lexicon, the lexeme's inflection triples were entered manually. The discovery of the inflection of words in the dynamic lexicon (from the DLA procedure) is slightly more complicated, and will be discussed in section 4.3.2.

3.3.6 HPSG schemata and topological fields

Eight generic immediate dominance schemata are defined in the core grammar:

- STRUCTURE-HEAD-ADJUNCT
- STRUCTURE-HEAD-SPECIFIER
- STRUCTURE-HEAD-SUBJECT

- STRUCTURE-HEAD-COMPLEMENT[11]
- STRUCTURE-HEAD-CLUSTER
- STRUCTURE-HEAD-FILLER
- STRUCTURE-HEAD-RFILLER
- STRUCTURE-HEAD-APPOSITION

They are organised according to the following scheme, inspired by Müller (2002): each HEAD-X derives from the other HEAD-NON-X schemata. If the HEAD-X schema affects a certain part of the feature structure (*e.g.* SUBJ), then the HEAD-NON-X schema defines a kind of standard behaviour for that feature (*e.g.* copy the SUBJ feature from the head daughter). For instance, these are definitions for STRUCTURE-HEAD-NON-SUBJECT and STRUCTURE-HEAD-SUBJECT[12]:

```
structure-head-non-subject :=
  structure-headed-binary &
[ SYNSEM        [ LOCAL.CAT.VAL.SUBJ #subj ],
  HD-DTR.SYNSEM [ LOCAL.CAT.VAL.SUBJ #subj ] ].

structure-head-subject :=
  structure-head-non-complement &
  structure-head-non-cluster &
  structure-head-non-adjunct &
  structure-head-non-filler &
  structure-head-non-apposition &
  structure-head-non-specifier &
[ SYNSEM        [ LOCAL.CAT.VAL.SUBJ < > ],
  HD-DTR.SYNSEM [ LOCAL.CAT.VAL.SUBJ < #subj > ],
  NH-DTR.SYNSEM #subj ].
```

However, STRUCTURE-HEAD-SUBJECT is still not finished. We have seen before that topological fields are a crucial tool for describing word order of German in the verbal domain, and a core grammar for German should model this aspect appropriately. What is proposed in this section is a flexible and powerful solution to define topological fields in terms of a finite state automaton (FSA). The FSA that models the topological fields in the verbal domain is given in figure 3.11. The states/vertices represent the topological fields themselves. The FSA's edges indicate how an analysis can proceed through different topological fields while creating an analysis bottom-up. Let's see which path in the FSA is followed for the following sentence (see figure 3.3 for an HPSG analysis):

(46) Antje würde den Käse gegessen haben.
Antje would the cheese eaten have.

[11] In fact, STRUCTURE-HEAD-COMPLEMENT has three variants, eliminating the first, second or third complement from the COMPS list. These are necessary for the potential different orderings of complements.

[12] Later, we will see why STRUCTURE-HEAD-NON-RFILLER is not in this list.

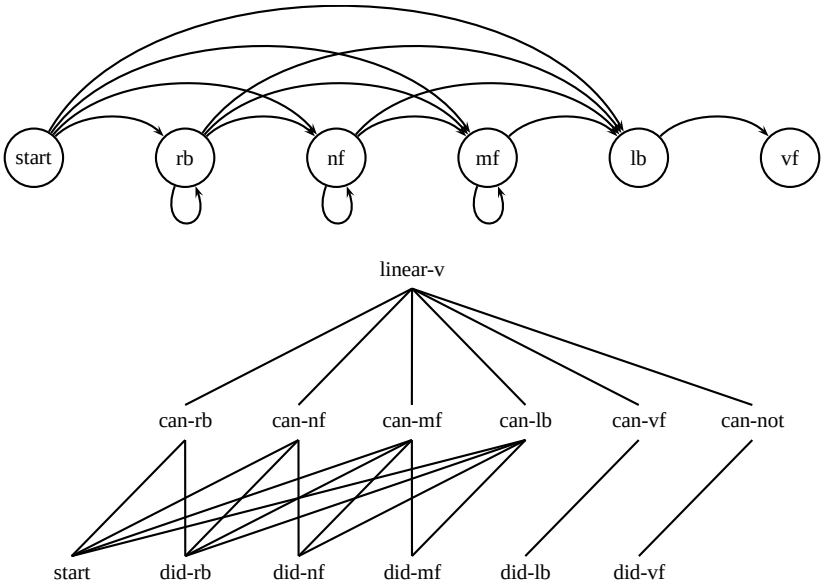


Figure 3.11: These figures show how the treatment of topological fields can be regarded as a finite state automaton, and how this mechanism can be implemented in a hierarchy of typed feature structures. The abbreviations 'vf', 'mf' and 'nf' stand for *Vorfeld*, *Mittelfeld* and *Nachfeld*, respectively; 'lb' and 'rb' stand for 'left bracket' and 'right bracket'.

Antje would have eaten the cheese.

The deepest verb is 'gegessen', which is in the 'start' state initially. 'gegessen' is combined with 'haben', which forms together the right bracket. There is no *Nachfeld*, so that phase is skipped. Instead, the next state is 'mf' (*Mittelfeld*), when the complement 'den Käse' is coupled with the right bracket. There is no path in the FSA from 'mf' to 'nf', which entails that this phrase excludes the possibility to combine with any constituent in the *Nachfeld* later in the process. The last two steps are also taken, after which the FSA ends in the *Vorfeld* state. From here, there are no outgoing edges in the FSA, meaning that the tree for the entire phrase can not combine with anything else from here.

This FSA needs to be implemented in the $\mathcal{TDL}$ framework (Krieger and Schäfer 1994). At the basis of the implementation is a subhierarchy of the *linear* type. This type has three daughters, one for each domain: *linear-v*, *linear-n* and *linear-o*. A subdivision of the *linear-v* type is shown in figure 3.11. Under *linear-v*, there is a *can* layer and below the *can* layer, there is the *did* layer. Both the *can* and *did* layer contain one type for each topological field. Additionally, there is a *start* node in the *did* layer. The *did* layer inherits from the *can* layer using a many-to-many relation, indicating which states can follow the current state: every line from the *did* layer to the *can* layer equals an arrow in the corresponding FSA. In the example in figure 3.11, if the current phase of the constituent is *Mittelfeld*, it can remain in the *Mittelfeld* stage or move on to the left bracket (but not to the *Vorfeld* or *Nachfeld*). A vertical line means that a

certain topological field can be repeated. Vertices with no outgoing connections are represented in the *did* layer via a *can-not* node.

Every subtree in an analysis has one of the *did* types in its LINEAR feature, which indicates in which state the subtree currently is. The content of the LINEAR feature can be used by the root conditions (determining whether a certain constituent spanning the complete utterance is indeed a correct sentence): the root condition for declarative sentences requires the *Vorfeld* to be completed (*i.e.* the LINEAR feature to be *did-vf*), while interrogative sentences should be have its LINEAR feature set to *did-lb*.

The rules in the grammar make use of the LINEAR feature. For each topological field, a structure type is created, in which the LINEAR features are checked and updated. For instance:

```
structure-mittelfeld := structure-verbal &
[ NH-DTR.SYNSEM.LOCAL.PLACE-V    did-mf,
  HD-DTR.SYNSEM.LOCAL.CAT.LINEAR can-mf,
  SYNSEM.LOCAL.CAT.LINEAR        did-mf ].
```

This type makes sure that the head daughter is allowed to engage in a *Mittelfeld* rule. This can be any *did*-subtype of *can-mf*. Also, the rule marks the newly created constituent as *did-mf*. The PLACE-V feature introduces extra possibilities. For instance, a verb can mark that one of its complements should only appear in certain topological fields. Furthermore, a modifier can define in which topological field it should appear[13], and with smart typing in the *can* and *did* layers, even sets of topological fields.

A tentative definition of STRUCTURE-MITTELFELD-SUBJECT-HEAD is as follows:

```
structure-mittelfeld-subject-head :=
  structure-mittelfeld &
  structure-head-subject &
  structure-head-last.
```

This is not a complete definition yet. The reader might have wondered what the interplay is between non-local constructions and the rule presented before. How does a certain rule know whether the ANC-ALL features from both daughters should be merged or not? And what to do with the RSLASH features from both daughters? It turned out that this behaviour does not coincide on the generic schema level (*e.g.* STRUCTURE-HEAD-SUBJECT), but it does fit comfortably on the level of individual rules (*e.g.* STRUCTURE-MITTELFELD-SUBJECT-HEAD). Therefore, the definition of STRUCTURE-MITTELFELD-SUBJECT-HEAD should define its behaviour in terms of how and whether the NONLOCAL and ANC features should percolate upwards:

[13] Seen from this perspective, the PLACE-X features share some characteristics with the POST-HEAD Boolean feature in the Matrix grammar.

```
structure-mittelfeld-subject-head :=
  structure-mittelfeld &
  structure-head-subject &
  structure-head-last &
  structure-rslash-from-nh &
  structure-rel-from-nh &
  structure-anc-add-none &
  structure-anc-active-nh.
```

Summarising, each rule that is finally instantiated by the parser is a combination of the following dimensions:

- An HPSG schema
- A topological field
- Head-first or head-last
- Non-local properties

So far, it has been assumed that the right bracket is not empty, with a right-branching analysis. In case the right bracket is empty, we follow Crysmann (2003), and use a left-branching analysis. This makes the FSA for verb slightly more complicated, with one hierarchy for either direction of branching. The FSA for the left-branching analyses is simpler, because only three fields are defined (the right bracket and Nachfeld cease to exist in this scenario). Separable verb particles are considered as normal complements of the verb, and are put in the *Mittelfeld* (and not in the right bracket). This is more efficient for the same reason as put forward by Crysmann (2003): in that case, the gap was highly underspecified, and the same holds here for the particle, leading to large local ambiguities. A disadvantage of this (compared to the situation where the particle is considered to be part of the right bracket) is that rightward extraposition to the *Nachfeld* can not be analysed by the core grammar, if the right bracket only consists of the separable particle:

(47) Antje isst den Käse auf den Bernd mitgebracht hat.
Antje eats the cheese up that Bernd brought has.
Antje eats up the cheese that Bernd brought.

We have seen examples for the verbal domain only so far, but two different domains are defined as well: nominal and other. This means that the entire notion of word order is implemented by using FSA to model topological fields. For the nominal domain, the core grammar defines five fields: head noun, determiner, pre-determiner, post-determiner/pre-noun and post-noun. For all other heads (*e.g.* adverbs, adjectives), the topological field structure is simpler: first, branch leftward (take elements on the right side of the head), and then branch rightward.

3.3.7 Coordinations

An important part of a deep grammar is the implementation of coordination structures. I will not delve into details about this part of the core grammar, however, but there are a few examples I would like to highlight.

Coordinations of verb phrases often display a form of *ellipsis*, which means that a certain part of the phrase is deleted (or: elided), because the hearer is able to infer what the deleted part is. Consider the following sentences:

(48) [Antje will den Käse essen] und [Peter will das Bier trinken.]
Antje wants the cheese eat and Peter wants the beer drink.
Antje wants to eat the cheese and Peter wants to drink the beer.

(49) Antje [isst den Käse] und [trinkt das Bier.]
Antje eats the cheese and drinks the beer.
Antje eats the cheese and drinks the beer.

(50) Morgen will Antje [den Käse essen] und [das Bier trinken.]
Tomorrow wants Antje the cheese eat and the beer drink.
Tomorrow, Antje wants to eat the cheese and drink the beer.

(51) Morgen will [Antje den Käse essen] und [Peter das Bier
Tomorrow wants Antje the cheese eat and Peter the beer
trinken.]
drink.
Tomorrow, Antje wants to eat the cheese and Peter wants to drink the beer.

(52) Morgen [will Antje den Käse essen] und [will Peter das Bier
Tomorrow wants Antje the cheese eat and wants Peter the beer
trinken.]
drink.
Tomorrow, Antje wants to eat the cheese and Peter wants to drink the beer.

In examples 48-50, the subject (‘Antje’) is elided, and it functions as a subject in the second conjunct as well. The adverb ‘morgen’ is elided in examples 51-52. The core grammar properly models that the elided constituent plays a role in both conjuncts. For instance, ‘Antje’ fulfills the subject role for both ‘essen’ and ‘trinken’ in the first three examples.

Another example where constituents play multiple roles in one phrase is when non-maximal noun phrases are coordinated:

(53) Von seinen Beschäftigten verlange er vor allem Arbeitsmoral und
From his employees demands he for all labour-ethics and

ordentliches Auftreten.
orderly demeanour.
He particularly demands labour ethics and orderly demeanour from his employees.

(54) Der französische Multi-Unternehmer und ehemalige Minister Bernard
The French multi-entrepreneur and former minister Bernard
Tapie...
Tapie...

In these examples, the modifier 'vor allem' and the apposition 'Bernard Tapie' should be connected to both elements of the coordinated phrase ('Arbeitsmoral' and 'Auftreten' in example 53; 'Multi-Unternehmer' and 'Minister' in example 54). Another peculiar coordination, at the interface of morphology and syntax, that the core grammar covers is the following:

(55) die blutigen Kasten- und Religionskriege
the bloody castes- and religion-wars

In this case, it seems as if there is a coordination between 'Kasten-' and 'Religions', both of them modifying 'kriege'[14].

The implementation of coordination structures can be a tedious part of writing a grammar. There are a few reasons for this. First, coordination of constituents can often be analysed using non-lexical solutions only. For instance, the coordination of two singular NPs containing count nouns yields a plural NP, in both English and German. A coordination of mass nouns, on the other hand, yields a singular NP. Another source of complexity is that rules for coordinations frequently introduce much ambiguity, which can only be constrained on the basis of the semantics of the conjuncts, and not on the syntactic properties. Some rules for coordinations have been implemented in the core grammar, but the balance between the amount of sentences that displayed this kind of coordinations was relatively small, and did not weigh out the computational consequences. A number of rules for asyndetic coordinations (coordinations without a coordinator, such as 'und'/'and') were therefore not used eventually.

3.4 Summary

In the remainder of the thesis, the core grammar will play a major role, being the base for the results in the forthcoming experiments. The main motivation for the workflow I present in chapter 3-5 was that hand-written formal grammars are

[14]This is not expressed in the annotation scheme of the Tiger treebank (Brants *et al.* 2002), however, in which a coordinated NP is created, with the left conjunct have a TRUNC part-of-speech tag (from 'truncated'). The core grammar mimics the Tiger annotation in this case.

not scalable: it takes much effort to create and it is hard to maintain. Regarding the first aspect, the development time for the core grammar was around 1 person year, which is much less than existing hand-written grammars for German (Crysmann 2005; Dipper 2003). The modularity of the grammar constructions and the compact size of the set of lexical types contribute to the maintainability of the grammar.

The second contribution of this chapter is a flexible and powerful way to model topological fields by means of finite state automata, using only the restricted variant of the $\mathcal{TDL}$ formalism that is used in the DELPH-IN tools. This analysis can easily be extended to other languages, more topological fields and more domains.

The grammar itself, as a resource, is a valuable addition in the landscape of (German) deep parsers. Although no novel linguistic analyses are presented, the core grammar is a synthesis of existing HPSG accounts of a non-trivial range of phenomena that occur in the German language. However, the development of a deep grammar is a continuing exercise, and there are a number of constructions that the core grammar is not able to analyse satisfactorily (yet), for instance because no analysis is known in the literature, or because readily available analyses would cause the grammar to overgenerate too much. The following is a non-exhaustive list of uncovered phenomena:

- Subjectless passive sentences;
- Partial VP fronting;
- Free relatives;
- Gapping;
- Auxiliary flip (or: *Oberfeldumstellung*);
- Extraposition crossing a particle in the right bracket;
- Extraction into the *Mittelfeld* (Müller 1997).

At this point, evaluation of the core grammar *per se* is hard to achieve, as it has virtually no coverage. One could use a test suite to this end: a carefully created collection of grammatical and ungrammatical utterances that tests whether certain linguistic phenomena are covered and whether ungrammatical utterances are properly rejected. I also used this approach (see appendix A), mostly for regression testing during development. However, the risk of test suites is that only phenomena that the grammar writer deems important are included, possibly obscuring shortcomings of the grammar. An alternative method of evaluation (unit testing) of the core grammar's constructional coverage will be given in section 5.2.

4 Creation of a deep lexicon

Because the lexicon in the core grammar from the previous chapter only contains function words, and no content words, it has practically no coverage. The lexicon plays a pivotal role in lexicalised grammars, and the construction of the lexicon is an immense task: lexical entries contain detailed descriptions of the words on a morphological, syntactic and semantic level. It can save the grammar writer a lot of time if there was a resource from which this information could be inferred. In this chapter, the Tiger treebank (Brants *et al.* 2002) is used for this purpose.

The chapter commences with a small introduction in which the approach followed in this chapter is compared to the methods in earlier deep grammar extraction studies. This is followed by an overview of the Tiger treebank. To facilitate the subsequent step, a preprocessing stage is proposed. Finally, a deep lexical acquisition (DLA) step is carried out, of which some descriptive statistics are given. Note that although much of the work in this chapter is described in (Cramer and Zhang 2009), there have been considerable improvements since, so the results presented in this chapter provide an up-to-date account of this work.

4.1 Introduction

Acquiring deep lexical resources is an important task for deep parsing. As has been noted by Baldwin *et al.* (2004), omissions in the lexicon are a frequent reason for parse failure for the ERG (Flickinger 2000), and there is no reason to assume that this is different for other lexicalised grammars. However, creating a deep lexicon can be a very time-consuming task, if done manually. With increasingly large and detailed linguistic resources (such as treebanks and dictionaries) available, researchers have sought ways to leverage these to shorten the development time for their systems.

In section 2.4, a number of deep grammar extraction methods have been discussed. The main advantage of these methods is that lexical entries can be deduced, or even read off from annotated training data, with high precision and recall. The algorithms that are presented pose two prerequisites:

1. A sufficiently informative treebank should be available.
2. There should be the possibility to convert this treebank into a format that conforms to the target formalism.

Preliminary efforts to convert the trees from the Tiger treebank into HPSG derivation trees (not reported on in this thesis) proved to be fruitless, mainly

caused by the fact that HPSG parse trees are not simple binarisations of the Tiger trees. The second condition can thus not be fulfilled. The Tiger treebank provides the algorithm enough information to couple lexemes to the appropriate lexical types that are defined in the core grammar. This simplifies the acquisition process considerably, but it leaves the responsibility to cover certain phenomena to the core grammar. Other than that, there are three other major differences to the workflows from the previous deep grammar extraction algorithms (see section 2.4).

The first is a consequence of including a small lexicon in the core grammar: not all lexical items need to be derived. The lexemes that are supposed to be contained in the open word lexicon are complementary to the lexemes in the core lexicon. For instance, no new prepositions or auxiliary verbs should be learnt in the DLA procedure. In section 3.3.3, it has been shown why lexemes with idiosyncratic properties are already contained in the core lexicon. This might render the DLA procedure more straightforward. The word classes that will be identified by the DLA algorithm are: (non-auxiliary) verbs, nouns, names, adjectives and adverbs.

The second difference with previous approaches is that the set of lexical types is defined by the grammar writer beforehand, and thus, there is no need for constructing them automatically. This difference has also been discussed in section 3.3.2. Sometimes, the inference of such lexical types is artificial. In the study by Miyao *et al.* (2004), for instance, the building blocks (*synsems* for NPs, PPs, etc) are defined by the grammar writer, and this already shapes the space of lexical types that the algorithm can find: it is unlikely that truly novel types are discovered.

In the approach that is explained in this chapter, the grammar writer can easily add details that are hard to find by a heuristic procedure. The lexical types are so detailed, that the algorithm can not be expected to discover all the very specific details of the lexical type it is supposed to find. For example, adverbs that can modify adjectives have fairly strong preferences on where they are located with respect to the adjective they modify, so this distinction needs to be made. However, this distinction is not useful for adverbs modifying verbs, as they can be placed in either the *Vorfeld*, *Mittelfeld* or *Nachfeld*. So, for some adverbs there is a generalisation over placements, and for some adverbs this generalisation is not there. Hence, given the level of descriptive detail for these lexical types, all lexical types are given beforehand by the grammar writer, and the lexical acquisition algorithm only needs to associate lemmas with the existing lexical types.

The third difference to previous approaches is that for some phenomena, a constructionalist solution has been chosen instead of an entirely lexical one. Therefore, one can expect that the number of lexical types is relatively re-

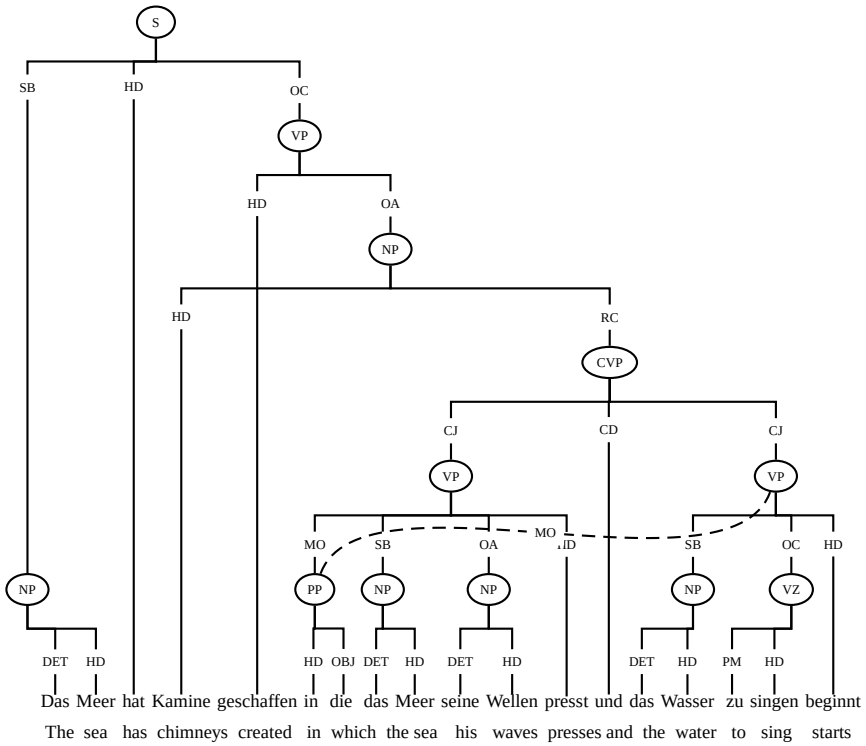


Figure 4.1: An example graph from the Tiger treebank, adapted from s16695. A verbatim translation of the (somewhat poetic) sentence is: 'the sea has chimneys created in which the sea the waves pushes and (in which) the water to sing starts'. The dashed line indicates a secondary edge.

stricted, compared to, for instance, the approach taking by Hockenmaier (2006). Strongly lexicalised analyses for these phenomena would lead to predictable variations of the same lexical type and hence, unnecessary local ambiguity. Permutation of arguments and adjuncts in the *Mittelfeld* is a good example of this: several lexical entries are needed in a true lexicalist approach (possibly leading to sparseness), whereas only one entry is needed by the more constructionalist approach in this study. See section 3.3.6 for further details on the constructions that achieve this behaviour.

Although the algorithm contains a small number of statistical components, it is mostly based on a hand-written heuristic algorithm, comparable to previous deep grammar extraction methods. The advantage is that the process can be tightly controlled, in order to maintain a high level of quality in the lexicon. On the other hand, this entails that extra work needs to be done, and that the workflow becomes highly language- and resource-dependent, and much of the work is not transferable to other projects (even though the process itself is more replicable than just hand-writing).

4.2 The Tiger treebank

Following Hockenmaier (2006), Cahill *et al.* (2005) and Rehbein (2009), I use the Tiger treebank (Brants *et al.* 2002)[1] to extend the core grammar. The text comes from a German newspaper (*Frankfurter Rundschau*). The corpus consists of more than 50,000 sentences, with an average sentence length of 17.6 tokens (including punctuation tokens). As explained in section 2.3.1, the treebank is constructed by having a model propose an analysis, which is then corrected by an annotator. This model has been either a statistical model (Brants 2000) or a linguistic model (Dipper 2000).

The Tiger treebank is a dependency treebank, meaning that the edges between the nodes have labels, in this case to denote grammatical functions. Even though the basic structures are trees, formally, the trees might by non-projective (*i.e.* have crossing edges). This allowed the annotators to easily represent German's relatively free word order. The importance of this is underlined by the fact that around one-third of the sentences receive a non-projective analysis. An example of a tree from the treebank is given in figure 4.1.

Secondary edges are used to represent ellipsis in coordinations, and turn the tree into a directed acyclic graph (DAG), as can be seen in figure 4.1. In this case, the constituent 'in die' ('into which') is not only connected to 'presst' ('presses'), but also has a secondary edge to 'beginnt' ('starts'), indicating its role of modifier in both conjuncts. Note that, because a secondary edge also bears a label, a constituent can play different roles in each conjunct, although it does not happen very often in practice.

The annotation scheme defines 26 phrase categories, 56 part-of-speech tags and 48 edge labels. The treebank also encodes number, case and gender at the noun terminal nodes, and additionally, degree at adjectival terminal nodes. Tense, person, number and mood is annotated at verb terminals. Whether a verb is finite, an infinitive or a participle are encoded in the part-of-speech tag. Inflection classes are not annotated.

The annotation guidelines indicate that the subject should be attached to the S-node, whereas the other arguments and adjuncts of the deepest verb should be attached to that verb. This means that if an argument or an adjunct appears in the *Vorfeld* (instead of the subject), combined with a sentence including an auxiliary, this results in a non-projective tree. Subject control can be identified by observing a verb with an OC (clausal object) dependent, which itself is has the VP category, with 'zu' ('to') + an infinitive as its head. There are two exceptions to the rule that only subjects can be sisters of auxiliary verbs. The first is object control, in which the object is a sister of the auxiliary. The second is that adverbs

[1] `http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERCorpus/`, freely available for non-commercial use.

in the *Vorfeld* are explicitly marked as modifiers of the auxiliary, not of the deepest verb. [2]

4.2.1 Preprocessing the treebank

The annotation format posed a number of problems for the DLA algorithm. Therefore, before the extraction process is started, a number of preprocessing steps are carried out. This preprocessed variant of the corpus is also used for all other experiments in this thesis.

The most important peculiarity in the treebank is that in many phrases, no head is explicitly indicated. This was done in order to retain theory-independence. For instance, there are disputes about whether the determiner or the noun is the head of a noun phrase/determiner phrase. Therefore, the designers of the format chose to annotate determiners, adjectival modifiers, nouns and names in NPs with the edge label NK (noun kernel). A head-finding procedure in the spirit of Magerman (1995) and his successors is used to re-annotate all NKs into either DETs, MOs, HDs and APPs. Similar arguments arise for PPs and CPs, for which it is not entirely clear whether the adpositions and complementisers are the heads of the respective phrases, or rather markers. In the preprocessing step, these are split into a HD and an OBJ, where the phrase category of the OBJ is identified automatically on the basis of the head. Particles in circumpositions are marked with the label AD.

The second large transformation involves verb-particle combinations. In German, the particle is sometimes attached to the verb, and sometimes separated, depending on whether the verb phrase is verb-final or not, and on the morphological properties of the verb (see examples 36-39 in section 3.1). The annotation format gives different lemmas for both cases; the particle is only included if the particle is attached to the verb in the source text. To improve on this, if the particle is not included in the lemma, it is prefixed. For instance, in the sentence 'Er wählt den Käse aus' ('He chooses the cheese out'), the lemma of the verb is changed from 'wählen' ('choose') into 'auswählen' (lit. 'out-choose').

Other than these two major changes, a number of smaller changes are carried out as well. The ADJD part-of-speech tag, for example, can mean either a predicative or an adverbial form of an adjective. This is disambiguated into ADJA or ADV, respectively.

[2]The Alpino annotations, whose format was influenced heavily by the Tiger treebank, solve this in a more elegant way, by not distinguishing between primary and secondary edges, and adding more dependencies between verbs and their dependents. Subject and object control and passive constructions are more easily identified, because links between all subjects/objects and all verbs are established. For instance, in a (Dutch translation of) the sentence 'The cheese is eaten.', a subject dependency is made between 'cheese' and 'is', and an object dependency is established between 'cheese' and 'eaten'.

4.3 Acquisition of the lexicon

With the preprocessed version of the Tiger treebank in place, the process to automatically derive the deep lexicon can be started. The algorithm's skeleton is straightforward: a simple top-down traversal of the tree. Depending on the phrasal category of each non-terminal node, different types of information are collected, and attributed to the head of the non-terminal node. Morphological features are extracted from terminal nodes. After all sentences have been processed this way, the accumulated information is used to determine the set of lexical types that the word connects to, and how each word is inflected (if applicable).

4.3.1 Syntactic properties

Which kinds of syntactic features are recorded and attributed to the phrase's head node, depends on the category of the phrase:

S and VP For each normal verb, the type of subject is determined, being either a nominative NP, the expletive 'es' (comparable to 'it' in 'it rains') or a CP. Also, the subcategorisation frame is discovered, for which the algorithm has the following components at its disposal: genitive, dative and accusative objects, PP objects, separable verb particle, reflexives, predicatives, CPs and three types of verbal complements (wh-phrases, full sentences, 'zu'-phrases). A maximum of three complements (excluding the subject) is enforced.

NP A number of complements are identified, but only one complement per lexical entry is allowed. Possible complements are: zu+infinitive phrases, wh-phrases, CPs, PP objects and full sentences. It is also determined whether the noun allows appositions (if seen once, this is concluded). Some nouns can also directly modify verbs, which is also picked up.

AP Comparable to NPs, only one complement is allowed. Possible complements are: PP objects and dative and accusative NPs. Modification constraints are encoded in the lexical type in the core grammar, so they do not need to be discovered in the DLA algorithm.

AVP Adverbs do not have any complements, but sometimes they allow for (possibly extraposed) comparative phrases, denoted by function label CC, even though the degree of the adverb is not set to comparative. An example of such an adverb is 'gleichermaßen' ('equally'), allowing for a phrase such as 'gleichermaßen drastische wie unpopuläre Maßnahmen' ('equally drastic as unpopular measures'), where the PP headed by 'wie' ('as') is connected

to the adverb. Some adverbs can be used as object in a PP ('bis morgen'; 'until tomorrow'). This information is collected as well.

All phrases For complementation, the head of the phrase selects for the dependent. The reverse is true for adjuncts: they select for the head. These modification constraints are also picked up, but only for adverbs, because these constraints are already defined for adjectives and adpositions in their respective lexical types. The procedure does not only discover which kind of head the adverb modifies, but also the position of the adverb with respect to the head. For instance, 'genug' ('enough') in the phrase 'schnell genug' ('quickly enough') can only be placed after the head in the adverbial phrase, not before.

All phrases above assume that the lexical type of the head can be determined correctly on the basis of the context of its mother. For instance, when a noun phrase is observed, the properties of the head noun only depend on the elements within the noun phrase, and not on elements outside. This assumption works well for most head types, but is not sufficient to correctly determine the lexical type of verbs, for which a somewhat wider context window is necessary. More specifically, the subject is attached to the conjugated verb, whereas the procedure above, discovering the subcategorisation frame of the deepest verb, is only at the level of the deepest verb, so the type of subject is not readily available to the algorithm. The situation is more complex when one of the deeper nodes is also a coordinated VP. Also, passive constructions need to be accounted for, meaning that the subcategorisation frame of the deepest verb must at least contain an accusative NP as object, and the agent does not need to be expressed altogether. The solution I propose is to do one run beforehand, identifying (subject, deepest verb, passive) tuples, in which the first two elements point to nodes in the tree, and the last is a boolean. The procedure for finding the subcategorisation frames is then carried out for each tuple. In the following (somewhat artificial) example, all three complications are exhibited:

(56) Antje wollte den Käse essen aber wurde abgelenkt.
Antje wanted the cheese eat but was distracted.

The first pass yields the following set of tuples:

Subject	Deepest verb	Passive
Antje	essen	false
Antje	abgelenkt	true

For each tuple, the subcategorisation frame will be determined correctly: both 'essen' and 'ablenken' are transitive verbs.

4.3.2 Morphology

As explained in section 3.1, German has a moderately complex morphology, and this aspect of the lexical acquisition process turned out to be particularly hairy. In the most straightforward scenario, for each word category for which morphology is involved (verbs, nouns and adjectives), inflection triples are recorded, which map the word form via a morphological rule to the word form's lemma. The morphological rules are hand-written (see section 3.3.5), but are only meant to specialise the features involved in describing the morphology. Let's consider the word 'essen' ('eat'), of which a number of word forms have been observed, at a certain point in the lexical acquisition process:

```
essen IR-VERB-INF essen
isst IR-VERB-3-SG-PRES-IND essen
esst IR-VERB-2-PL-PRES-SUBJ essen
aßen IR-VERB-3-PL-PAST-IND essen
aßen IR-VERB-1-PL-PAST-SUBJ essen
```

The problem is that over 70 of these forms have to be learnt (for verbs), and that many of the slots will not be filled using this naïve strategy. In other words, data sparseness poses a serious problem for the morphological component of the DLA algorithm. A (partial) solution lies in a phenomenon called *syncretism*: the knowledge that different inflections of a lemma have identical word forms. For instance, it is known that the first-person plural form of a German verb is always identical to the third-person plural form. Given this piece of knowledge, the table above can be automatically extended with the following rows:

```
aßen IR-VERB-1-PL-PAST-IND essen
aßen IR-VERB-3-PL-PAST-SUBJ essen
```

What we need is a systematic way to achieve this. The inflectional rules themselves are already designed to make maximal use of syncretism. There is no inflectional rule for '1.pl.past.ind' in the core grammar, but a more abstract rule 'pa-13-pl' does exists, covering all inflections that are in the present tense, are in either first of third person and are plural. The mood is left undetermined, meaning that the word forms are equal for both indicative and subjunctive mood. A mapping from MORPH values to the inflectional rules is necessary, for instance:

'1.pl.pres.ind' → 'pa-13-pl'
'3.pl.pres.ind' → 'pa-13-pl'
'1.pl.pres.subj' → 'pa-13-pl'
'3.pl.pres.subj' → 'pa-13-pl'

Now, when a word form is observed in the source treebank (*e.g.* '1.pl.past.ind'), this is mapped to the correct inflectional rule ('pa-13-pl'). The triple (word form, inflectional rule, lemma) is eventually recorded[3]:

[3]Here, we see a clear example of the interdependence between the core grammar and the DLA algorithm. Because syncretism is exploited, the inflection rules had to be changed in the core grammar

```
aßen IR-VERB-NPD-PA-13-PL ablehnen
```

This works well for verbs. For adjectives, however, there are usually multiple word forms for one MORPH, due to the fact that this form also depends on the 'strongness' of the NP (determined by presence and type of determiner). This strongness is not annotated in the Tiger treebank. This is solved by not looking at the MORPH value, but by identifying the suffix of the adjective. Given the rather regular nature of adjectival inflection, this seems to work well. Also, each suffix maps to a *set* of AGR values, rather than singletons:

$$\text{'er'} \rightarrow \{ \text{ sm-m-n-sg, s-f-gd-sg, s-*-g-pl } \}$$

where the AGRs have four features: inflection class (strong, mixed and/or weak), gender, case and number. This is a fairly good solution: even though there are as many as $3 \cdot 4 \cdot 2 \cdot 3 = 72$ cells in the inflection tables, there are only a small number of suffixes ('-e', '-er', '-en', '-es', '-em').

Discovering the morphology of German nouns poses another problem to this machinery: the declension class of nouns needs to be identified (which is also not annotated in the Tiger treebank). For feminine and neuter nouns, there is only one class, but for masculine nouns, there are three candidates: strong nouns, inanimate weak nouns and animate weak nouns. Apart from these categories, there are words like 'Prozent' ('percent'), of which number and case are left undetermined in the Tiger treebank, and nouns such as 'Anfang' ('start'), that have case undefined, and have the ability to modify verbs. The solution I put forward for this is to heuristically discover to which inflection class each noun belongs, and to create separate morphology mappings for each inflection class.

Although these algorithms have been successful in handling German's morphology in a more principled way, there is another source of confusion: errors and inconsistencies in the treebank. To counter these, multiple candidate inflection triples are recorded, but only the candidate with most votes is kept.

4.4 The resulting lexicon

Gauging the reliability of the DLA procedure is hard, just by looking at the entries themselves. To give the reader an impression of the results, a number of examples in the lexicon are given in table 4.1. A manual pass through a representative part of the lexicon, testing the validity of the entries, would be cumbersome, and the evaluation would be subjective. Neither would such a check say anything about the completeness of the lexicon (which is impossible anyway). Therefore, only descriptive statistics will be presented in this section, which can be roughly compared to other state-of-the-art grammars. I am aware

noun + adp	noun + cp	noun + izu
Zweifel (an)	Zweifel	Zögern
Zusammenhang (mit)	Zuversicht	Zwang
Zusammenarbeit (mit)	Zusatzfrage	Zustimmung
Zugriff (auf)	Zeitungsmeldung	Zusage
Wissen (um)	Wissen	Zeitsouveränität

noun + sent	noun + wh	adj + adp	adj + NPacc
Zeit	Wahl	zählend (zu)	zusammenfassend
Werbeaussage	Tip	zurückgehend (auf)	zierend
Weisung	Schätzung	verärgert (über)	wählend
Vorurteil	Schimmer	verurteilt (zu)	wiederholend
Vorstellung	Prognose	vertröstet (auf)	wert

adj + NPdat	verb + izu	verb + sent
zuwider	zögern	zitieren
zustehend	zwingen	zeigen
zugänglich	wollen	wissen
zugewiesen	wissen	winken
zugetan	werben	wiederholen

verb + part	verb + refl	verb + adp
zusichern	zusammenstellen	zählen (zu/auf)
zustandekommen	zurückverfolgen	zwingen (zu)
zustechen	zurechtfinden	zweifeln (an)
zustimmen	zufriedengeben	zusteuern (auf)
zutreffen	zieren	zittern (um)

Table 4.1: This table shows five lexemes for a number of lexical types.

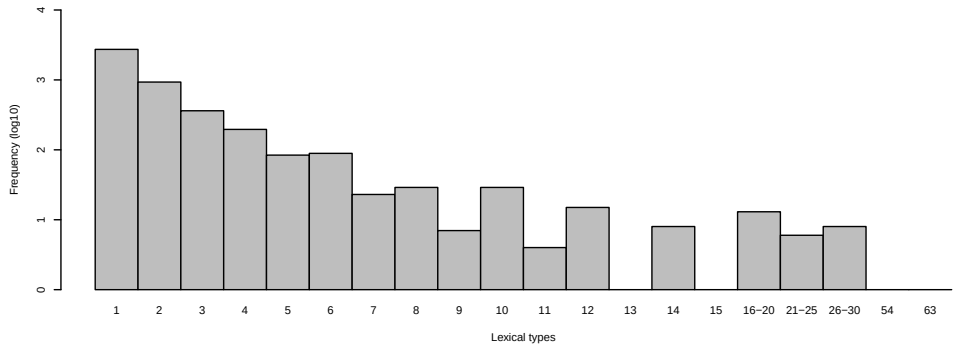


Figure 4.2: This graph shows the distribution of the number of lexical entries per verb. The lemmas having 54 and 63 lexical entries are ‘machen’ (‘to make’ or ‘to do’) and ‘sein’ (‘to be’), respectively. Note that the vertical scale is logarithmic; there are 2734 verbs with only one lexical type, and 932 with only two.

Part-of-speech	Lemmas	Lexical entries	Inflection triples
Verbs	4543	9235	18745
Nouns	33835	34821	51303
lt-noun-mod		228	
lt-noun-noapp		30086	
lt-noun-noapp-adp		255	
lt-noun-noapp-cp		100	
lt-noun-noapp-izu		150	
lt-noun-noapp-sent		81	
lt-noun-noapp-wh		12	
lt-noun-app		3749	
lt-noun-app-adp		47	
lt-noun-app-cp		30	
lt-noun-app-izu		55	
lt-noun-app-sent		20	
lt-noun-app-wh		8	
Names	12445	12783	na
Adjectives	7318	8018	50480
lt-adj		7722	
lt-adj-adp		69	
lt-adj-npacc		102	
lt-adj-npdat		125	
Adverbs	2654	4577	na
lt-adv-verb		2424	
lt-adv-adj_left		981	
lt-adv-adj_right		11	
lt-adv-adp_left		250	
lt-adv-adv_left		369	
lt-adv-adv_right		55	
lt-adv-nbar_right		170	
lt-adv-np_left		317	
lt-adv-none		303	

Table 4.2: This table gives a breakdown of the size of the acquired lexicon, excluding the entries in the core lexicon. The type lt-adv-none is for adverbs such as ‘morgen’, that can also be used as a noun phrase (‘bis morgen’; ‘until tomorrow’). The abbreviation ‘adp’ stands for adposition, ‘izu’ is for ‘zu’+infinitive clauses, comparable to the English ‘the ability to eat cheese’, and ‘sent’ stands for a sentential complement. A further breakdown of the lexical types of verbs is given in table 4.3.

Lexical type	Entries	% of entries	% of lemmas
lt-verb-reg-npnom-npacc	2259	24.46%	49.72%
lt-verb-reg-npnom	1574	17.04%	34.65%
lt-verb-reg-npnom-refl	593	6.42%	13.05%
lt-verb-reg-npnom-part	560	6.06%	12.33%
lt-verb-reg-npnom-npacc-part	545	5.90%	12.00%
lt-verb-reg-npnom-adp	288	3.12%	6.34%
lt-verb-reg-npnom-npacc-npdat	248	2.69%	5.46%
lt-verb-reg-npnom-npdat	203	2.20%	4.47%
lt-verb-reg-npnom-sent	200	2.17%	4.40%
lt-verb-reg-npnom-npacc-adp	192	2.08%	4.23%
lt-verb-reg-npnom-cp	162	1.75%	3.57%
lt-verb-reg-npnom-refl-part	157	1.70%	3.46%
lt-verb-reg-npnom-refl-adp	130	1.41%	2.86%
lt-verb-reg-npnom-izu	117	1.27%	2.58%
lt-verb-reg-npnom-npacc-npdat-part	113	1.22%	2.49%
others (182 lexical types)	1894	20.51%	na

Table 4.3: Breakdown of the occurrence of the lexical types of verbs in the lexicon. The third column indicates the percentage of all lexical entries having this lexical type, adding up to 100%. The fourth column shows the proportion of lemmas having this lexical type. For instance, half of the verbs can function at least as a transitive verb.

of the relative weakness of this type of evaluation, but an alternative method of evaluation (of the entire development chain) will be applied in section 5.2.

The results of the deep lexical acquisition procedure, based on the first 45k sentences in the Tiger treebank, are summarised in table 4.2 (the remaining 5k sentences will be used for evaluation purposes in the empirical experiments). Almost 60k lexical entries (lexemes) and their syntactic properties are discovered, many of which are nouns or names. This skewed distribution is perhaps even more pronounced than in other languages, because German can invent an innumerable number of nouns by compounding two or more atomic nouns together.

The large majority of the entries (86%) is filled with 'just nouns' ('lt-noun-noapp' in table 4.2), not having complements, and not allowed to have appositions. The proportion of is even higher for vanilla adjectives ('lt-adj': 96%). The question then arises: is it worthwhile to spend effort finding the remaining, more complex entries? That depends on the frequency of usage. One might argue that there is a positive correlation between frequency of the usage of a word and its syntactic and semantic complexity. From this, one might infer that the more complex lexical entries are probably frequent enough to have impact on the accuracy of the grammar on unseen text.

A somewhat worrying trend can be observed in the rightmost column in table 4.2, indicating how many inflection triples are found in total. As alluded to before, smart typing of the agreement features can lead to significant reductions in the number of inflection triples. However, even though special attention has been given to this aspect, no satisfactory results are obtained, as the DLA stage

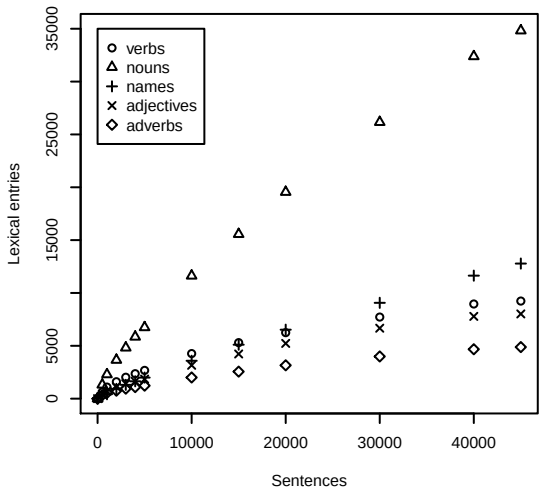


Figure 4.3: Learning curves of different parts-of-speech versus input size.

has not been able to cover the complete inflection tables. This is especially true for nouns, for which 1.5 inflection triples are found per lemma, but for which typically four slots are to be filled. The figures look better for verbs and adjectives, but it must be realised that there are far more word forms to be discovered for these parts-of-speech.

More details on the acquisition of verbs are presented in table 4.3 and figure 4.2. Table 4.3 shows the distribution of lexical types over the lexicon, ordered by class frequencies. Not unexpectedly, the intransitive and transitive types are most frequent, with almost half of the lemmas being identified as transitive. In total, 197 verbal lexical types are found. However, the first 15 already account for almost 80% of the lexical entries. Interestingly, none of the alternative subjects (CPs, expletive 'es') occur in the top-15. Figure 4.2 shows the skewedness of the distribution of the lexical types over the entries in the lexicon: most of the verb lemmas only associate with one (2734) or two (932) lexical types, and only a few lemmas (all of them frequent) have many more lexical entries, most notably 'machen' ('to make'/'to do') and 'sein' ('to be').

It is also interesting to see whether the learning process has converged. This is depicted in figure 4.3. One can clearly see that the nouns and names categories are still growing near-linearly, but that the other categories are getting closer to convergence. On the other hand, we can hypothesise that the marginal lexicon in a possible next chunk of training data mostly consists of easily-identifiable lexical types. As explained in section 3.3.2, the core grammar has a number of generic lexical types, that map parts-of-speech to deep lexical types. For verbs, these are the intransitive and transitive lexical types. Hence, it would be interesting to see how many lexical entries are added that are *not* either transitive

or intransitive. This gives a better indication of whether a parser based on this grammar would perform better on unseen text. It turns out that for the first 5000 sentences, the proportion of non-generic learnt lexical entries is around 51%, but *rises* to 68% in the last chunk of 5000 sentences. In other words, the later in the learning process, the more non-trivial verbs are discovered. The following table shows these statistics for a number of parts-of-speech (names are not included, as all entries are generic lexical types):

Part-of-speech	% in first 5000	% in last 5000
Verbs	50.8%	67.8%
Nouns	3.8%	2.1%
Adjectives	2.8%	2.2%
Adverbs	46.7%	54.7%

Table 4.4: This table indicates the proportion of learnt lexical entries that do not map to a generic lexical type. The second column shows this number in the first 5000 sentences, the third column of the last 5000.

But how does the resulting lexicon compare against other lexica within the DELPH-IN framework? This is illustrated in table 4.5. For each grammar (where Cheetah is the codename of the grammar that is developed in this thesis), the number of lexical types, the number of lexical entries and the number of 'standard' lexical entries are given for a number of parts-of-speech. What is considered a 'standard' lexical type is usually clear from the distribution of lexical types in the lexicon: usually, there is a limited set of types that take up the majority of the lexicon. Although the numbers are not entirely comparable, especially not across languages, they do give a crude idea of the sizes and levels of detail that the lexicons offer. The difference between the numbers of lexical types between the ERG and the grammars for German is striking: the ERG has a larger inventory of lexical types, allowing finer distinctions between individual lexemes. The lexicon of Cheetah is slightly larger than the lexica from the other grammars, especially when one realises that most adjectives in the GG are

	Verbs			Nouns		
	LTs	LEs	SLEs	LTs	LEs	SLEs
Cheetah	197	9235	3833	13	34821	33835
GG	105	5065	1775	23	21797	20327
ERG	324	8373	3576	252	18378	14064
	Adjectives			Adverbs		
	LTs	LEs	SLEs	LTs	LEs	SLEs
Cheetah	4	8018	7722	9	4577	2424
GG	17	10169	9936	12	4935	4615
ERG	96	5707	3459	83	2085	925

Table 4.5: This table gives the number of lexical types, the number of lexical entries in the lexicon and the number of 'standard' lexical entries in the lexicon for a number of grammar in the DELPH-IN framework. 'Cheetah' is the codename of the grammar that is developed in this thesis. The GG (German Grammar: Crysmann 2005) and the ERG (English Resource Grammar) are mostly hand-written grammars.

counted twice, because the predicative reading and the attributive reading are considered separate types.

4.5 Summary

In this chapter, a deep lexicon has been derived from the Tiger treebank, which connects seamlessly with the core grammar from chapter 2. Like previous studies, a hand-written heuristic procedure was developed for this purpose. However, no new lexical types needed to be created, and lexical entries that were already in the core lexicon were ignored in this process. This resulting procedure is straightforward, and easy to maintain. Extra attention has been given to the morphology of German, and specific tools have been crafted in order to counter data sparseness and to reduce the effect of annotation errors in the treebank. The resulting lexicon was obtained with relatively little effort, and is comparable in size to lexica of state-of-the-art, hand-written grammars.

The combination of core grammar/lexicon and the deep lexicon that was created in this chapter forms a coherent formal account of the German language. In the next chapter, attention will be paid to how this new resource can be used, for instance to create a treebank or to parse unseen text.

5 Leverage of the gold standard

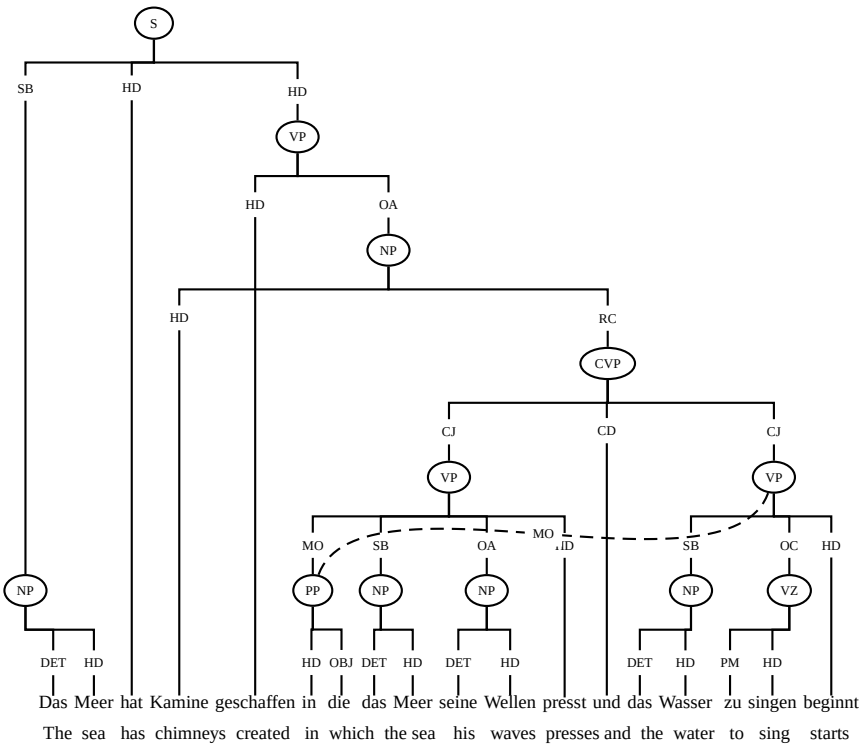
So far, the Tiger treebank (Brants *et al.* 2002) has been used as the source for a deep lexical acquisition step. However, it can be used as a gold standard as well. First, an explanation is given in section 5.1 on how gold syntactic dependencies are extracted from a held-out set from the Tiger treebank. These should be comparable to the output of the parser, which in our case is a (restricted) form of Minimal Recursion Semantics (Copestake *et al.* 2005). Therefore, I show how these MRSs can be converted to syntactic dependencies in Tiger's annotation format in the second section. The grammar's output can now be compared to the gold standard, which opens an array of possibilities. First, a new type of evaluation will be introduced in section 5.2: unit testing. The procedure checks whether the complete chain of core grammar, deep lexical acquisition and MRS-to-dependencies conversion works without flaws, independent of the quality of the used disambiguation model. Section 5.3 explains how a decent part of the static Tiger treebank can be turned into a dynamic treebank (Oepen *et al.* 2004) automatically, by parsing the text from the corpus first, and selecting the best reading by using the dependencies extracted from the Tiger treebank. Finally, a number of experiments will be presented in section 5.4, showing the adequateness of the grammar and its associated disambiguation model on parsing unseen text.

5.1 Comparing parsing output with the gold standard

5.1.1 Extracting the dependencies from the treebank

In order to be able to do an evaluation against the gold standard, we should define more precisely what that gold standard is. The definition in this study is as follows: a set of (head, label, dependent) dependency triples. This is computed by assigning a head/dominator to each phrase, and assigning one triple for each non-dominator child in the phrase to the dominator, using the label that the non-dominator child bore. Secondary edges are treated equally as normal edges. Although some forms of headedness were already made more explicit during the preprocessing phase in section 4.2.1, there are still some phrases where a form of dominator-dependent relation needs to be established:

PH/RE phrases Some CPs, PPs and NPs do not have a head, but are divided between a PH (placeholder) and RE (repeated element) chunk. In this case, the PH is seen as the head.



ROOT	ROOT	hat	presst	SB	Meer
hat	SB	Meer	Meer	DET	das
Meer	DET	Das	presst	OA	Wellen
hat	OC	geschaffen	Wellen	DET	seine
geschaffen	OA	Kamine	beginnt	MO	in
Kamine	RC	und	beginnt	OA	Wasser
und	CJ	presst	Wasser	DET	das
presst	MO	in	beginnt	OC	zu
in	OBJ	die	zu	OC	singen

Figure 5.1: Figure 4.1 repeated, but this time, the gold standard syntactic dependencies are given as well.

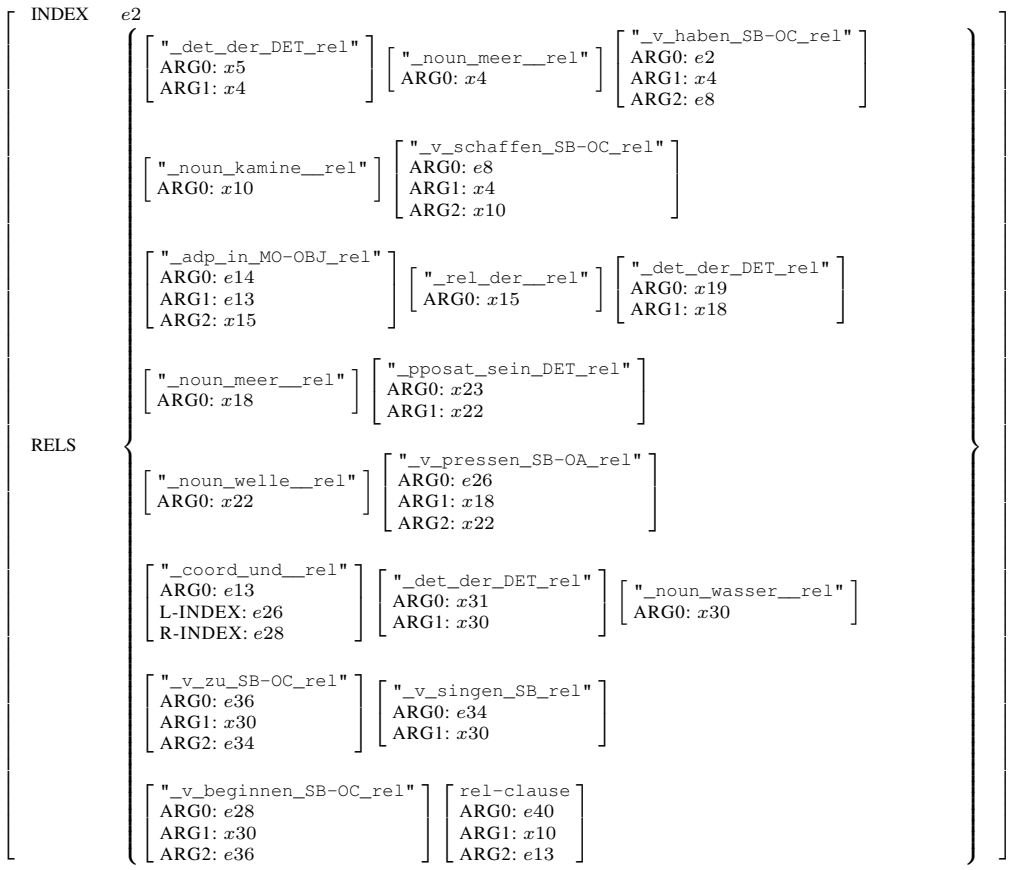


Figure 5.2: An MRS, as produced by the grammar, is depicted. The LTOP, HCONS and LBL features are omitted, as they are not used for determining the final dependencies.

Coordinations In normal coordinations, the coordinator is taken to be the head. In asyndetic coordinations, the leftmost conjunct is taken.

DL nodes In reported speech, a DL (discourse-level constituent) is split between a DH (discourse-level head) node and an RS node (reported speech). The former is assigned the head role in this case.

VZ phrases It seems more obvious to treat ‘zu’ (‘to’) as the head in these phrases, and use the verb as its dependent.

An example of how the dependency extraction works is given in figure 5.1. Normally, the number of dependencies is equal to the number of words (if the ROOT dependency is included in the count), but not in this case, because one extra dependency is generated due to the use of the secondary edge, where the PP that introduces the relative phrase is copied into the second VP conjunct. It should be noticed that the extraction of the dependency triples is not hampered by the non-projectivity of the tree, introduced by the extraposed, coordinated relative clause.

5.1.2 Role identification in the predicate

Now that the gold standard is in dependency triple format, we need to make sure that the output of the parser can follow the same format. The native format of the output of a DELPH-IN grammar is Minimal Recursion Semantics, which is an unordered set of elementary predicates, with a predicate and a set of arguments. However, the core grammar only makes use of a restricted subset of the functionality of MRS, as explained in section 3.3.4. An MRS structure that the grammar produces is given in figure 5.2, and these should be converted to the gold standard dependencies in figure 5.1.

First, the `ARGx` naming scheme has to be changed into syntactic dependencies. Normally, MRS predicate names are built according to a strict coding scheme, encoding the relation's part-of-speech, the lemma and the semantic sense. In the scheme in this grammar, the sense field is going to be an enumeration of the syntactic roles that the individual arguments play. As a consequence, the transitive verb 'pressen' ('press') has the following predicate: `_v_pressen_SB-OA_rel`, where SB stands for subject, and OA for accusative object. The `ARG0` argument is the relation's index, and will not be changed in this step. The other arguments can then be re-labelled using the `sense` field. In the previous example, `ARG1` will become `SB`, and `ARG2` `OA`. This step is illustrated in the following example[1]:

$$\begin{bmatrix} \texttt{"\_v\_pressen\_SB-OA\_rel"} \\ \text{ARG0: } e26 \\ \text{ARG1: } x18 \\ \text{ARG2: } x22 \end{bmatrix}$$

will become:

$$\begin{bmatrix} \texttt{"\_v\_pressen\_SB-OA\_rel"} \\ \text{ARG0: } e26 \\ \text{SB: } x18 \\ \text{OA: } x22 \end{bmatrix}$$

In some cases, the head does not select for the dependent, but the other way around: modifiers (MO, MNR, AG, PG) and determiners (DET, AG). Therefore, these have to reversed. In the example above, that means that the dependencies between 'in' ('in') and 'presst' ('presses') (and 'beginnt'; 'starts') are affected. This reversal process is also necessary for determiners, because the associated label is not always DET, but can be AG as well for some determiners (*e.g.* 'dessen'; 'whose'). Therefore, the determiner decides the dependency label, not the noun, and therefore the syntactic function that the determiner plays is encoded in the predicate name of the determiner.

The last step is to dissolve all non-lexical relations. For a few phenomena, it would be illogical to identify lexical roles in the relation. The core grammar

[1] In this step, also L-INDEX and R-INDEX, which are involved in the representation of coordinations, are transformed into CJS.

solves this by introducing non-lexical relations. In the example in figure 5.2, REL-CLAUSE is such a non-lexical relation. This is removed, and an RC link between 'Kamine' ('chimneys') and 'und' ('and') (the latter representing both relative clauses) is established. What follows is an illustration of this:

$$\left[\begin{array}{l}\texttt{"\_n\_kamine\_\_rel"}\\ \text{ARG0: } x10\end{array}\right]\left[\begin{array}{l}\texttt{rel-clause}\\ \text{ARG0: } e40\\ \text{ARG1: } x10\\ \text{ARG2: } e13\end{array}\right]\left[\begin{array}{l}\texttt{"\_coord\_und\_\_rel"}\\ \text{ARG0: } e13\\ \text{CJ: } e26\\ \text{CJ: } e28\end{array}\right]$$

will become:

$$\left[\begin{array}{l}\texttt{"\_n\_kamine\_\_rel"}\\ \text{ARG0: } x10\\ \text{RC: } e13\end{array}\right]\left[\begin{array}{l}\texttt{"\_coord\_und\_\_rel"}\\ \text{ARG0: } e13\\ \text{CJ: } e26\\ \text{CJ: } e28\end{array}\right]$$

The conversion process is realised as a structured script. The algorithm always succeeds, although the conversion might be incorrect for a number of corner cases.

5.2 Unit testing

During the development of a grammar, the grammar writer needs to keep track of the consequences of modifications that are made. The consequences are multidimensional, and the grammar writer needs to keep a good balance between these: (exact match) coverage, efficiency, overgeneration, etc. For instance, the introduction of a rule that covers a fairly infrequent phenomenon, but which also forms a source of severe local ambiguity might be revised or even rejected by the grammar writer. In the grammar development set-up in this thesis, doing such regression testing is even more important, because there are quite a few potential points of breakdown in the entire chain: not all lexical entries might be correctly derived from the treebank or the MRS-to-dependencies conversion can contain errors. Also, there can be sentences for which the core grammar lacks, for instance by missing or too constrained constructions, or due to an omission in the core lexicon.

Profiling a hand-written test suite in the `[incr tsdb()]` system (Oepen and Carroll 2000b) has been the most common way to do regression testing within the DELPH-IN framework, and this regression testing paradigm is widely used within the grammar engineering community. Test suites have a number of virtues. For instance, they are systematic and diagnostic: all phenomena that the grammar covers are included, and when a certain modification in the grammar causes one sentence in the test suite to fail, it is instantly clear which linguistic phenomenon is not covered anymore. Furthermore, test suites provide the possibility to test whether the grammar properly rejects negative items. Therefore, if the goal of the grammar writer is to write a grammar that models the language as precisely as possible, test suites are a good means to assist reaching that goal.

However, test suites have one particular shortcoming: the data is artificial. This means that the sentences are usually fairly short, and that the distribution of phenomena in real-life text is not reflected in test suites. In other words, the influence of infrequent phenomena is overestimated. If the aim of the grammar writer is to maximise the grammar's quality (as indicated by a quality measure of choice) on unseen text, test suites are less appropriate. The procedure I propose (unit testing) is meant as a tool to focus the grammar writer's attention towards those linguistic phenomena that have the largest impact. Also, regressions are discovered not only in the core grammar, but also in the DLA procedure and the MRS-to-dependency conversion.

The unit test works as follows. Instead of deriving the lexicon from a large set of sentences, the DLA procedure is applied to only one sentence. That same sentence is then parsed by the resulting grammar, and all resulting MRSs are then converted to dependencies in Tiger format. If one of the analyses contains the exact match, this sentence passes the test; otherwise, it fails. Notice that the quality of the disambiguation model has no influence on the results. The unit test only checks whether the pipeline of core grammar, DLA procedure and MRS-to-dependency conversion works well. When results from earlier development cycles are kept, regressions can be easily spotted. Apart from the fail/success measurements, statistics about the grammar's efficiency and ambiguity are also output.

During the development of the grammar, the first 500 sentences of the Tiger treebank have been used for quality assurance. Currently, 48% of this set pass the unit test. Most of the development has gone into optimising the toolchain for this set, and hence evaluating the errors on this set will give an erroneous picture of where the problems in the chain are located. Therefore, the first 55 sentences *outside* the first 500 sentences which do not pass the unit test are taken, after which the source of the error is tracked down manually. The results of the experiment are summarised in table 5.1. As the data clearly show, lacking constructions in the core grammar is the main cause for failing the unit testing procedure. Examples of such missing constructions are gapping, complicated coordinations, 'weder... noch...' ('neither... nor...') coordinations and parenthetical constructions. However, and this is where the long tail of grammatical constructions appears, there is not a single failure among these that occurs more than once. A solution might be easy to implement for some of the failures, but might have too severe ramifications in terms of efficiency: a PP coordination without coordinator (an asyndetic coordination) is only permitted when the heads of both PPs are equal. If this constraint would be omitted, this would be a large source of ambiguity; however, asyndetic PP coordinations with distinct heads do occur in the Tiger treebank, and cannot be parsed completely correctly. The other categories in table 5.1 show the same kind of flat distributions, which

Category	♯
Error in gold standard	3
Core grammar	31
Acquisition	7
MRS conversion	7
Timeout	1
Unclassified	6
Total	55

Table 5.1: A division of the reasons for failing to reproduce the correct dependencies during unit testing.

indicates that the low-hanging fruit has been harvested, and much time would have to be invested in order to raise the percentage of sentences passing the unit test.

5.3 Automatic creation of a dynamic treebank

No modern-day, wide-coverage grammar can do without a treebank. Its most well-known use is for training disambiguation models (Toutanova *et al.* 2005; Riezler *et al.* 2001; Malouf and Van Noord 2004). Treebanking by hand takes much effort, and therefore, this section investigates how this can be sped up, if a treebank in a different format or linguistic formalism is already available. The deep grammar extraction community has used forms of heuristic conversion processes, for instance by adding f-structures (Cahill *et al.* 2002) or feature structures (Miyao *et al.* 2004). However, as argued in section 2.7, there are several reasons to opt for the core grammar/DLA combination in this study, but a disadvantage of this workflow is that it does not automatically yield a treebank.

5.3.1 Methodology

In this study, an attempt is made to convert the Tiger treebank to HPSG derivation trees that are compatible with the grammar that has been constructed until now. Roughly, the methodology consists of a parsing stage, after which the disambiguation is done automatically using the information from the source treebank. This idea is not new. For instance, Riezler *et al.* (2001) used the Penn treebank annotations to train a model that distinguishes readings from the LFG parser that are compatible with the Penn treebank annotations, and readings that are not compatible. Because the authors restricted the training set to those sentences for which no more than 1000 readings were returned, the training set only consisted of 10k sentences. When the guiding gold standard data and the parser output are very anisomorphic, the procedure can become cumbersome. In the study by Zhang *et al.* (2009), the gold standard consists of dependencies, and the authors used simple heuristics on the HPSG parse trees to find the head of

a phrase, in order to recover the dependencies. They only took the parse with the highest unlabelled dependency accuracy score, but do not set a minimum on that score. Hence, each sentence that receives a parse is treebanked. In this study, the advantage of the source treebank being isomorphic is not there, and the methodology is more similar to the one by Zhang *et al.* (2009). The effort to make the grammar's output compatible with the dependency triples from the Tiger treebank (see section 5.1) renders a heuristic head-finding procedure redundant. Also, more stringent criteria are used: the score will be *labelled* f-score, and a minimum f-score is required for the sentence to be treebanked (in order to exclude poor readings).

Instead of opting for a stand-off annotation format, the idea of dynamic treebanks is used in this study, as explained in section 2.3.1 (Carter 1997; Oepen *et al.* 2004; Rosén *et al.* 2009). This has a number of consequences, some of them positive, some of them negative. A downside is the impossibility to treebank extragrammatical data. However, there are a number of advantages, too, as summarised by Oepen *et al.* (2004). The most important one for this grammar has to do with the grammar's future. It is well conceivable, as will be discussed in the Interlude, that this grammar is only an intermediate step towards a more advanced grammar. Therefore, there must be easy ways to have the treebank co-evolve with the grammar, and that is exactly what is accommodated by the update function of dynamic treebanks.

Creating a dynamic treebank consists of two stages:

1. Parsing
2. Disambiguating

During the parsing stage, the PET parser (introduced in section 2.6) (Callmeier 2000) is used. Common state-of-the-art techniques (subsumption-based packing, selective unpacking, hyper-active parsing) are turned on. As input, the text from the training set of the Tiger treebank (s1-s45000) is used. Together with the raw text, the gold part-of-speech tags are input (although mapped to broader categories[2], for practical reasons). These are used to restrict search space, using one of the methodologies of Dridan *et al.* (2008), in which only those lexical entries whose types are subtypes of the part-of-speech tag are put onto the chart. For each sentence, a maximum of 500 readings is recorded.

The disambiguation stage is fairly similar to the one described in section 5.2 on unit testing. From all readings the MRSs are extracted, and converted into Tiger-style dependencies, and the labelled f-score compared to the gold standard is computed. If the highest f-score is above a certain threshold β, that best

[2]The chosen tag set is: ADJ, ADP, ADV, CARD, DET, KON, NE, NN, OTHER, PART, PRELAT, PRON, TRUNC, VFIN, VIMP, VINF, VIZU, VPP.

Description	♯	% of all	% of parsed
Total	44993	100.0%	
Not parsed	13827	30.7%	
No analysis	11304	25.1%	
Timeouts	2510	5.6%	
Lexical errors	13	0.0%	
Parsed	31166	69.3%	100.0%
$0 < n \leq 100$	24177	53.7%	77.6%
$100 < n < 500$	3465	7.7%	11.1%
$n = 500$	3524	7.8%	11.3%
Annotated, $\beta = 1.0$	20379	45.3%	65.4%
Annotated, $\beta = 0.9$	25168	55.9%	80.8%
Annotated, $\beta = 0.8$	28656	63.7%	92.0%
Annotated, $\beta = 0.7$	29996	66.7%	96.3%

Table 5.2: This table lists the results for the treebanking experiment. n stands for the number of readings a sentence receives. β is the treebanking threshold.

reading is accepted; if none of the readings exceeds the threshold, all readings are rejected.

A maximum of 500 readings is recorded. Which 500 readings are recorded in case the grammar licenses more than 500 readings depends on the disambiguation model. However, no model is available initially. Therefore, only 6000 sentences are parsed and treebanked, from which 500 readings are extracted randomly (using a Maximum Entropy model with no features). From those sentences that are treebanked successfully, a disambiguation model is trained (Toutanova *et al.* 2005). Then, the parsing and disambiguation stage are repeated once, using the complete training set this time. This bootstrapping of a basic disambiguation model hopefully pulls as many correct readings as possible in the pool of 500 readings per sentence. The basics of the training of Maximum Entropy-based disambiguation models will be explained in section 5.4.

5.3.2 Results

A summary of the results is given in table 5.2. The first striking observation is that the grammar's ambiguity rates are relatively low, suggesting that the control the grammar writer has in this paradigm pays off: three quarters of the parsed sentences yield less than 100 readings. Only for 11.3% of the parsed sentences, the grammar licenses more than 500 readings. It is only for this part of the total set that the disambiguation model could be relevant. For around 25% of the sentences, the grammar did not recognise the structure of the sentence. Timeouts[3]

[3]Parsing stops after 60 seconds. No memory limit was enforced.

Rank	♯	%
$r = 1$	16767	66.62%
$1 < r \leq 10$	6121	24.32%
$10 < r \leq 100$	1869	7.43%
$100 < r \leq 500$	411	1.63%

Table 5.3: This table describes how the ranks of the annotated readings are distributed. For this table, $\beta = 0.9$.

were observed in 5.6% of the sentences[4]. Lexical errors indicates that a certain word was not recognised, and neither was its part-of-speech, in order to spawn one of the generic lexical types.

Looking at the treebanking results, the parser can reproduce the exact set of gold standard triples for around 45% of the input sentences. This number rises to 56% when the treebanking threshold β is lowered to 0.9, and rises to 64% when β is lowered even further to 0.8.

The quality of the disambiguation model (trained on 6000 sentences) can be assessed by looking at table 5.3. For all sentences that are treebanked (with $\beta = 0.9$), around two third of them already have rank 1, and more than 90% have a rank of 10 or smaller. This is a first indication that the disambiguation model works well, and will be elaborated on in section 5.4.

An important factor determining the chance of success for the treebanking procedure is the length of the sentence. This is illustrated in figure 5.3. The upper graph shows the distribution of sentence lengths in the Tiger treebank. The middle graph gives an indication of how sentence length influences ambiguity. The graph clearly shows that the number of readings rises with longer sentences. The bottom graph shows that the success of the treebanking procedure is highly dependent on sentence length. Each bin is divided in four categories (ranked from light to dark gray in figure 5.3):

- Treebanked successfully ($\beta = 0.9$)
- At least one analysis but not treebanked
- No error, but no analysis
- Error

Longer sentences show larger proportions of timeouts and unanalysed sentences, while the proportion of annotated sentences tends to become lower. The proportion of sentences where at least one analysis is presented, but none of them having an f-score exceeding β, is fairly constant in the entire range of sentence lengths.

[4]The so-called 'quickcheck' technique (Kiefer *et al.* 1999) was not used for this experiment. Later experiments will give lower percentages of the number of timeouts.

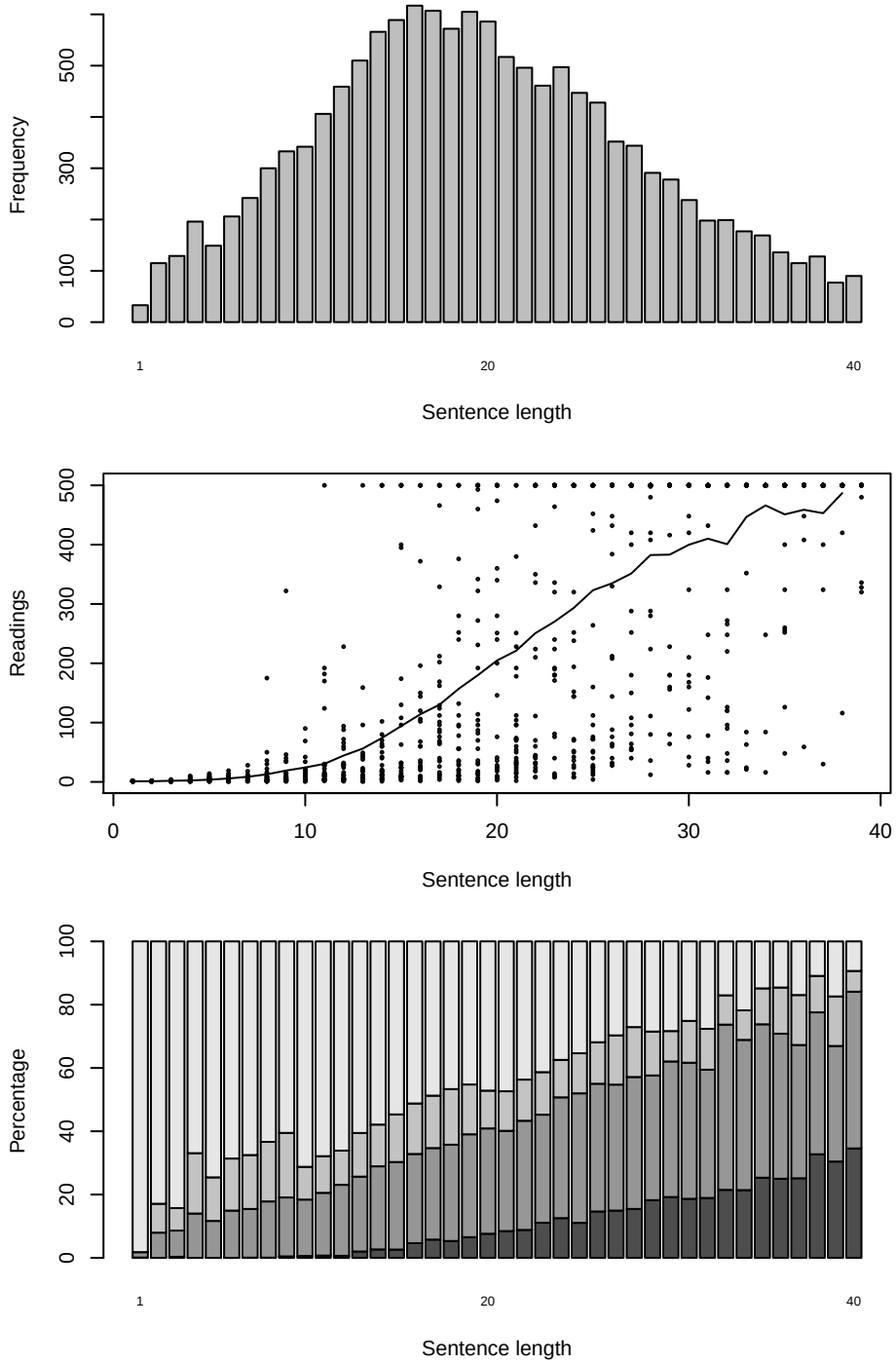


Figure 5.3: These three graphs indicate how the system performs with respect to sentence length. The first graph describes the distribution of the corpus' sentence length. The second shows for a subset of 1000 sentences how the number of readings correlate with sentence length. The bottom graph indicates how many sentences result in (from bottom to top): error; no error, but no readings; at least one reading but not annotated; annotated, with $\beta = 0.9$.

Depending of the reader's perspective, the annotation effort can be perceived as either a failure or a success. The results are nowhere near the percentages (approaching 100%) reported in other deep grammar extraction studies. However, an informative HPSG analysis is given for those sentences that did receive an annotated reading (45% or 56%). Before, the largest HPSG treebank for German contained around 3500 sentences, with fairly simple linguistic data (7.2 input tokens per sentence, on average), after a large amount of manual disambiguation (Wahlster 2000).

It is hard to estimate the development time of the treebank, because the time to create the link between the parser output and the Tiger treebank was the largest part of the work, and could be considered part of the process of grammar writing as opposed to creating the treebank. It seems reasonable to assume that this procedure of constructing a discriminant-based treebank is not slower than a manual approach.

5.4 Parsing unseen text

The results from the parsing stage from the previous section are positively biased, in a number of respects. First, the algorithm parses text on which the DLA algorithm has been run, and hence, there are few unknown words. Also, the results are based on the best score among all readings that are returned by the parser, which is equal to the oracle score of the disambiguation model. In a realistic scenario, only the highest-ranked solution is evaluated. In this section, the performance of the grammar on parsing unseen text will be evaluated. First, the development set is used to set a number of parameters related to the training procedure of the disambiguation model. After all parameters are set to their optimal values, one run will be done on the test set. The results from that experiment will function as a reference to which the experiments in the following chapters should be compared.

In order to improve the reproducibility of the results, the exact conditions will be given here. The PET parser is used, revision 766 from the Subversion repository[5], compiled from source, but with one difference: after the timeout (60 seconds) is reached, the parser will *not* fail, but the unpacking routine will be called, in order to find the solutions that were found until then, if any. This should give a small increase in coverage in comparison to the stock version of PET. The input to the parser is the raw text, annotated with gold part-of-speech tags, which will prevent the parser to spawn lexical types that are not compliant to the part-of-speech, as explained by Dridan (2009).[6]

[5] `https://pet.opendfki.de/repos/pet/main`

[6] The exact command-line to invoke the PET parser is: `cheap -cm -yy -mrs -default-les=traditional -packing=15 -sm=sm.mem -nsolutions=1 -timeout=60`

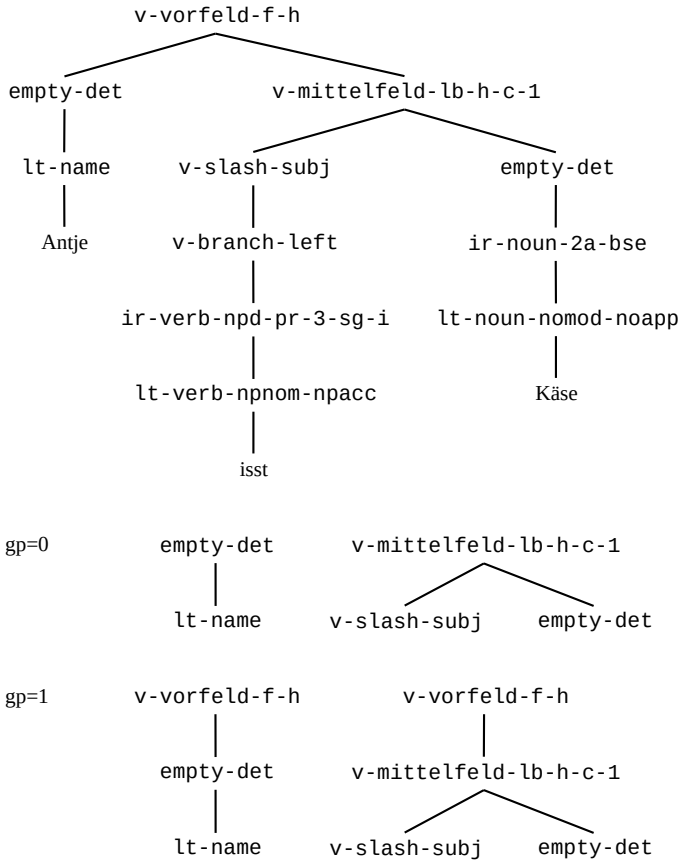


Figure 5.4: Shown is a parse tree of a simple sentence, as output by the grammar, accompanied by a number of features extracted from it, with different grandparenting levels. The translation for the sentence is 'Antje eats cheese'.

5.4.1 Optimising the disambiguation model

In this section, only one factor is being varied: the statistical disambiguation model. The PET parser currently supports one type of disambiguation model: the standard Maximum Entropy model. Adopting the same naming conventions as Toutanova *et al.* (2002), the probability of a specific reading of a sentence is:

$$P(t_i|s) = \frac{\exp \sum_{j=1,\ldots,m} f_j(t_i)\lambda_j}{\sum_{i'=1,\ldots,n} \exp \sum_{j=1,\ldots,m} f_j(t_{i'})\lambda_j}$$

Here i is the reading identifier and j is the feature identifier. $f_j(t_i)$ is the feature extraction function, which denotes the number of times feature j occurs in reading t_i. Informally, the learning procedure will try to maximise the conditional likelihood of all sentences in the training set by adjusting the weights λ_j, assuming that this will give the best results when the model is applied to held-out linguistic data. One reason for the attractiveness of this type of modelling is

`german.grm`, where `sm.mem` is different for all experiments in this section.

Setting		Result	
sent	β	Exact match	Precision
RANDOM		15.0%	74.8%
45k	1.0	24.6%	83.8%
45k	0.9	24.6%	84.1%
45k	0.8	24.5%	84.1%
1k	0.9	20.2%	80.8%
2k	0.9	20.6%	81.5%
3k	0.9	21.6%	82.0%
4k	0.9	21.8%	82.4%
5k	0.9	22.2%	82.5%
10k	0.9	23.0%	83.0%
20k	0.9	23.6%	83.5%
30k	0.9	24.2%	83.8%
40k	0.9	24.6%	84.0%
ORACLE		29.4%	88.8%

Table 5.4: Results indicating the influence of changing the training procedure of the disambiguation model. These results are based on the development set. 'sent' indicates on how many sentences the model is trained (these sentences might be annotated or not). β is the parameter for the annotation threshold.

that it scales well with very large numbers of features (in the order of millions). Although learning is computationally complex, decoding the probability of a new sentence given a certain model is straightforward, if the feature extraction function is easy to compute. In the experiments in this section, only those features that are recognised by the stock version of the PET parser are used. These are local parse trees, appended with a number of (grand-)parents. In figure 5.4, a parse tree with some of its associated features is depicted.

The quality of the disambiguation models will be evaluated using a number of statistics. On the sentence level, there is the 'exact match' metric, indicating for how many sentences the perfect solution has been found. Second, I will report on the precision of the parser's predictions on the level of syntactic dependencies. Both evaluation metrics are not particularly informative without context, and therefore a lower and an upper bound will be provided. As a lower bound, a RANDOM selection strategy is chosen[7]. Because precision-oriented parsers try to keep the number of analyses for a given sentence low, this baseline can perform surprisingly well. The second piece of context is the ORACLE score, which indicates the maximum performance of the model, if the best reading would always be picked. Between these two scores, there is a spectrum, and a model's performance can be assessed by looking at the location on this spectrum.

The results are summarised in table 5.4. Two variables can be altered. The first is the size of the training set (45k is the complete training set). The number of actual training instances depends on the second variable, β, the f-score

[7]In practice, this is done by using a disambiguation model with no features. It gives the same score to all readings, and therefore the decision to rank one reading over another reflects a chance distribution.

threshold that is used to decide whether a certain sentence should be annotated or not (see section 5.3). The disambiguation procedure works reasonably well. Compared to the RANDOM baseline, there is a 66% reduction of errors on the dependency level (relative to the ORACLE score). It must be taken into account that these scores are calculated over the *covered* sentences, which on average are slightly shorter than the entire treebank (12.9 vs 15.9 words per sentence, excluding punctuation tokens).

A second observation is that the training parameters have surprisingly little influence on the results. Changing the β parameter has little effect. Lowering β increases the training set (as more sentences will be accepted for annotation), but possibly with a lower quality. Hence, the training procedure might learn wrong generalisations. On the other hand, setting β will shrink the training set, and using less training instances usually leads to a degradation of results when machine learning methods are applied. However, the precision of the disambiguation model is fairly robust against these changes, without any noticeable difference between the results.

This result suggests that there is enough training data in the first place, or, in other words, that the training algorithm converges already after a smaller amount of data. We checked this by using smaller parts of the treebank for training the disambiguation model. The outcome was that, after a training set of 5k (of which around 2750 are annotated), the model's precision is already approaching the precision of the model trained on the full training set, a somewhat surprising result. One might attribute this result to two factors: the relatively small amount of ambiguity, and the high level of abstraction of the features.

5.4.2 Evaluation on the test set

The optimal parameters for training the disambiguation model have been found, it is time to test the performance on truly independent linguistic data: the test set. The results are summarised in table 5.5, and will function as a reference point for the results of the experiments that will be carried out in consequent chapters. The average time to parse the sentence is just over 4 seconds, which is the same order of magnitude as other DELPH-IN parser. 60.4% of the sentences receive at least one analysis, which is about 10% lower than in the treebanking experiments. For 21.7% of the sentences, the parser yielded the exactly correct parse. Compared to the development set, it appears that the parser was able to sustain its high precision on the dependency level (83.7%). The f-score on the test set is 54.9%. Generally, it is reassuring that the parameter settings that were chosen in the previous section generalise well to the test set.

The results in table 5.5 are best compared to the ones presented by Forst

Sentence-level	
Avg parse time	3.92s
Coverage	60.4%
Exact match	21.7%
Dependency-level	
Recall	40.8%
Precision	83.7%
F-score	54.9%

Table 5.5: This table shows the result of the parser on the test set, using the best parameters from the development set to train the disambiguation model ($\beta = 0.9$).

(2007), who applied a hand-written grammar in the LFG framework to another subset of 1600 sentences from the Tiger treebank. The author reports a dependency-only f-score of 77.2%. Their coverage is reported to be 86.4% on that data set (Forst 2007). Another source of comparison is the performance of two treebank-induced LFG grammars, as reported by Rehbein and Van Genabith (2009), where dependency-only scores are presented of 72.7% (for their own work) and 78.6% (for work done by Cahill *et al.* (2005)). Although the results are not immediately comparable, because they use a different subset and tag set (Forst *et al.* 2004), this does give a clear indication that the grammar, as it stands now, is not state-of-the-art, mostly caused by the low coverage score.

The results in table 5.6 give more insight in the performance of the parser on a number of aspects. Let's first observe the aggregate results at the bottom. The average recall, precision and f-score are repeated from table 5.5. Extra columns are added for the f-score when a random reading was selected (the baseline), and when the best possible reading was selected (oracle upper bound). The relative improvement between the lower and upper bound is given in the rightmost column. The disambiguation model achieves a 70.2% relative improvement over the entire test set.

Table 5.6 also shows how the parser performs on different tags. A prominent observation is that there is a large variation between the observed scores on these tags. When a certain tag is not recognised correctly in many sentences, this can be due to two sources: the grammar *per se* or the disambiguation model. The ORACLE f-score indicates how the grammar performs, irrespective of the disambiguation model. When the oracle f-score for a tag is lower than the micro-average[8] oracle f-score (57.7%), this indicates that the formal model does not suffice to model this tag correctly. The relative improvement shows how apt the disambiguation model is to pick the correct reading from the readings that are licensed by the grammar.

Relatively good scores are achieved for the ROOT, DET, SB, EP (expletive

[8]The micro-average averages over all instances in all classes (tags in this case). A macro-average is the average over all classes, thus giving more weight to instances of infrequent classes.

Label	♯	Recall	Precision		F-score		Rel.
				RANDOM	Observed	ORACLE	impr.
MO	9530	39.6%	79.0%	43.8%	52.8%	57.2%	66.8%
DET	5517	49.6%	98.0%	65.5%	65.9%	66.1%	63.8%
OBJ	5159	46.1%	94.0%	59.9%	61.9%	62.3%	81.7%
SB	3618	45.0%	89.9%	53.2%	60.0%	62.7%	71.5%
CJ	2769	30.2%	81.3%	40.5%	44.0%	46.6%	57.6%
ROOT	2500	54.0%	89.5%	61.9%	67.4%	68.0%	89.9%
OC	1934	43.6%	84.8%	49.7%	57.6%	57.8%	97.5%
OA	1446	42.5%	74.9%	39.1%	54.2%	58.1%	79.7%
MNR	1356	28.5%	64.9%	30.5%	39.6%	53.9%	39.0%
AG	1098	43.9%	91.6%	50.3%	59.4%	60.8%	85.9%
PNC	857	10.4%	98.9%	17.7%	18.8%	18.8%	100.0%
APP	677	35.2%	30.6%	19.5%	32.6%	40.9%	61.3%
PD	514	29.2%	85.2%	29.3%	43.5%	48.4%	74.5%
RC	405	34.6%	76.1%	31.8%	47.6%	57.3%	62.0%
OP	314	37.9%	66.9%	28.9%	48.4%	60.9%	60.9%
REFL	280	34.6%	95.1%	32.5%	50.7%	52.0%	93.5%
NG	276	33.7%	63.3%	35.5%	44.0%	58.7%	36.6%
SVP	271	35.1%	99.0%	51.8%	51.8%	51.8%	na
DA	255	29.4%	64.1%	26.1%	40.3%	42.4%	87.2%
PM	235	38.7%	98.9%	55.6%	55.6%	55.6%	na
PAR	214	0.0%	0.0%	0.0%	0.0%	0.0%	na
RE	138	8.7%	80.0%	11.9%	15.7%	17.0%	75.0%
NMC	135	10.4%	100.0%	9.9%	18.8%	18.8%	100.0%
CC	116	21.6%	64.1%	20.1%	32.3%	38.7%	65.4%
EP	87	48.3%	76.4%	37.2%	59.2%	64.3%	81.2%
SBP	82	18.3%	78.9%	14.5%	29.7%	34.3%	76.8%
AD	55	18.2%	83.3%	26.9%	29.9%	31.9%	59.5%
OG	12	16.7%	33.3%	8.7%	22.2%	22.2%	100.0%
other	192	0.0%	0.0%	0.0%	0.0%	0.0%	na
total	40042	40.8%	83.7%	48.3%	54.9%	57.7%	70.2%

Table 5.6: Listed in this table are the scores for the individual dependencies. The column under the symbol ♯ indicates how often the given dependency tag is observed in the gold standard. The rightmost column gives the error reduction that the disambiguation model achieves. The dependencies under 'other' are: AMS, CD, NK, AC, CP, PH, JU, DM, VO, OA2, RS, PG, –, AVC, HD, ADC, CM, SP, UC.

subject) and OBJ (complements of adpositions) tags. The f-scores are significantly higher than the micro-averaged f-score. In the case of the DET and OBJ tags, this is mostly due to the tight modelling of the formal grammar, as there is not much difference between the oracle and random scores for these tags. On the other hand, the disambiguation model properly models the ROOT, SB and EP tags, with high degrees of relative improvement between the upper and lower bounds.

It is widely accepted that coordinations are difficult to model. Formal models for these constructions are tedious to create, and the rules usually introduce much ambiguity. The low score for CJ (44.0%) is therefore not a surprise. The source of the errors lies in both the grammar and the disambiguation model: both the oracle f-score (46.6%) and the relative improvement (57.6%) are below the micro-averages by a large margin.

The parser's observed f-score is just below average on the most common tag: MO (adverbial modification; PP modification of the verb). Another modifier tag, MNR (PP modification of the noun), is recognised correctly in fewer cases. PP attachment is a common source of errors for both grammar-based and statistical parsers, because it is hard to constrain the over-generation and because there are no obvious features that can distinguish plausible analyses from less plausible analyses. The fact that the relative improvement score is below average is thus not surprising, but that the oracle f-score is below par is an unexpected result.

PNC (multi-word proper nouns) and NMC (multi-word numbers) suffer from a similar problem: they consist of more than one word (the tags are annotated on the relation between the individual parts between the lexical tokens). The low recall and high precision figures indicate that the lexical item is used appropriately during parsing, in those cases that the proper multi-word lexical item is in the lexicon. However, the DLA procedure has been unable to capture enough of these multi-word units. Better preprocessing facilities might help to overcome this issue. A similar conclusion can be drawn for the REFL tag, which shows low recall and high precision as well. Reflexives are fairly productive in German, and it might be more suitable to model reflexives as a verbal modifier rather than as a complement of the verb.

Modelling appositions (APP) turns out to be troublesome for the parser as well. The grammar allows any name to have appositions, as well as nouns which appear as the head of a head-apposition construction in one training sentence. It appears that the discovery of lexical items from the second category is insufficient, given the low oracle f-score. Because the head-apposition construction in the core grammar hardly constrains the head and the apposition, this must be due to the lexical acquisition.

The dependency tag PAR indicates parenthetical modifiers. As can be seen, this tag has not been implemented in the core grammar, nor is it a tag that can be inferred during the DLA stage. This is a typical case of a dependency that is not hard to model: it combines an unsaturated verbal phrase with a saturated phrase (regardless of its head). However, this definition of parenthetical modifiers would incur a heavy efficiency penalty, due to the massive overgeneration it would cause. Hence, this tag is not covered by the grammar.

5.5 Summary

The experiments in this chapter were meant to make maximal profit of the existence of a large, detailed gold standard treebank. In order to do so, the Tiger treebank was converted to dependency triples, and the output of the parser was shaped to facilitate the comparison between that output and the gold standard.

This entailed a divergence from the MRS formalism, which is the *de facto* standard within the DELPH-IN community. The resulting output format is more similar to the predicate-argument structures that the Enju parser produces and the predicates-only output within the LFG community.

The combination of different grammar engineering paradigms entails that different points of breakage are brought into existence. A new method for identifying and quantifying the sources of errors (quality assurance) in a deep grammar extraction has been presented (unit testing), taking advantage of the direct comparability of the deep grammar's output and the dependency treebank that was achieved before. A prominent result of experiments was that insufficient constructions within the core grammar was the main reason for not being able to fully reproduce the original Tiger annotations.

The same methodology was used to automatically convert the Tiger treebank into a full-fledged dynamic HPSG treebank. At the time of writing, this treebank is the largest HPSG treebank for German, containing the most realistic text. It was argued that a conversion by parsing (as opposed to using a heuristic procedure) was the only reliable strategy to reach a maintainable and transparent grammar-treebank symbiosis. For 45% of the sentences in the original treebank, an HPSG analysis compatible to the grammar from chapters 3 and 4 was found that can reproduce the entire set of dependency triples in the gold standard. When a small margin of error is tolerated, this percentage goes up to 56%.

Furthermore, experiments exploring the aptitude of the parser to analyse unseen linguistic data were presented. One interesting observation is that the results did not change much for a fairly wide range of settings for the disambiguation model training procedure. Also, a small treebank was already sufficient to reach the learning optimum. The results for the independent test set were promising: the parser's coverage exceeded 60%, of which more than one-third were given the exactly correct parse. On the dependency level, recall and precision turned out to be 40.8% and 83.7%, respectively, yielding an f-score of 54.9%. Results on the individual tags were mostly in line with the expectations, showing lower scores for coordination and PP attachment, and higher scores for tags indicating complementation.

Interlude

At this point in the thesis, an entire grammar, including a statistical disambiguation model, has been built up from scratch. I will refer to the grammar as 'Cheetah' (as in: a family member of Tiger). Now concrete results are available on Cheetah's achievement on unseen text, it is time to reflect on what has been achieved so far. Although a direct comparison is very hard to make across linguistic paradigms, languages and data and evaluation sets, I will try to make a fair and balanced comparison with other approaches within the entire parsing community, which I will divide in four categories: statistical dependency parsers (Nivre 2007; McDonald *et al.* 2005), deep grammar extraction (DGE) parsers (Clark and Curran 2004a; Cahill *et al.* 2005; Miyao *et al.* 2004), parsers based on hand-written grammars (Van Noord 2007; Riezler *et al.* 2001), and one particular group of instances of hand-written grammars: DELPH-IN grammars (Flickinger 2000; Crysmann 2005; Siegel and Bender 2002). Because each parsing paradigm has its advantages and disadvantages, the comparison will be multi-dimensional. A summary of the comparison is given in table 5.7.

Linguistic relevance

One of the reasons to use hand-written, precision-oriented grammars is that they grant us the possibility to test linguistic hypotheses on a large scale. Two types of such testing can be distinguished. The first is to encode certain linguistic predictions in the grammar. When trying to parse text (either unseen text or a test suite), one can learn from coverage and ambiguity rates whether the predictions about the language were accurate or not (Bender 2008b). Due to the scale of these experiments, it also becomes possible to test whether there are interactions between hypotheses. The second way to use deep parsing technology for hypothesis testing is by creating large treebanks, which can be queried for certain phenomena. An example is the study by Bouma and Spenader (2009), who investigated how Dutch reflexives behave in different contexts. Interesting about this work is that a treebank need not be available, as the output of the parser is considered to be accurate enough.

Cheetah offers a systematic, integrated formal account of the most prominent aspect of the German language. However, no publications offering new insights into formal models of German have appeared (Crysmann (2005) offers a good example how this has been done for German before). But there is no reason to assume that it could not. A grammarian can alter the lexical types, remove constraints or add constructions. Possible effects of such changes can now be tested on a large scale. In comparison to the HPSG grammar for German

	DELPH-IN grammars	Hand-written grammars	Deep grammar extraction	Statistical parsers
Linguistic relevance	-	-	+	+
Level of abstraction	-	=	=	+
Ability to generate	-	=	+	+
Efficiency	±	±	-	?
Coverage	±	±	-	-
Development time	+	+	-	-
Clean evaluation	+	=	=	=

Table 5.7: This table describes how Cheetah compares to existing parsing paradigms on a number of dimensions. '+' means that Cheetah compares favourably against the other categories; '-' means the opposite; '=' indicates equal performance on this particular domain. '?' indicates that no clear answer can be found in the literature, and '±' means that within the category, there are some to which Cheetah compares favourably, and some to which they do not.

(Crysmann 2005), the grammar writer can profit from a compatible gold standard and a higher coverage (these relative advantages do not apply to the more mature LFG grammar for German, for which extensive documentation is available (Dipper 2003)). Such changes are easier to make with Cheetah than with a completely hand-written grammar, because the lexicon derivation algorithm can be changed: if a refinement of a lexical type is made into two others, the lexicon derivation algorithm might be able to make this distinction as well, hence rendering a manual pass over the lexicon unnecessary. Also, because there is a large treebank in stand-off format available, the grammar writer can test whether a change increases the annotation rate (see section 2.1.5). In a DELPH-IN setting, the grammar writer typically needs to check by hand whether the newly covered sentences indeed contained the correct solution.

The space to evaluate linguistic hypotheses using purely treebank-based parsers seems very slim[9]: more linguistic information can only be added in the procedure that converts the source treebank to the target formalism. Cheetah's workflow therefore offers a more principled way to add linguistic knowledge into the grammar, which is showcased by a more elegant, constructionalist solution of permutation of adjuncts and complements.

[9]Not really linguistic, but certain cognitively inspired models of languages, such as adaptor grammars (Johnson *et al.* 2007), offer insights into language, although their view on human language capacities is predominantly probabilistic.

Level of abstraction

As was discussed in chapter 2, syntactic dependencies are nowadays the most widely used modality for parser evaluation, both within the deep and shallow parsing communities. Cheetah followed this trend, by using the syntactic dependencies directly derived from the Tiger treebank. The term 'syntactic dependencies' suggests, however, a sort of unified view on how this annotation is done, but in fact, there is no consensus on this topic. The available schemes differ widely. To assess the complexity of the task, one could suggest that there is a correlation between the number of different tags that are used in the annotation scheme and the complexity of the task for the parser, because the parser needs to be able to make finer-grained distinctions. However, if a certain label were partitioned into several labels following a deterministic procedure, the new dependency set would be as complex as the old set, even though there are more labels in the new set. Also, there are disagreements about whether dependency graphs can be non-projective and whether secondary links should be allowed. On top of that, even if there is agreement on the fact that scheme A is more complex than scheme B, this does not necessarily entail that scheme A is also more *informative* than scheme B for a certain upstream application.

These complications are there for syntactic dependencies alone. Some research groups prefer to evaluate their parsers using a superset of the information that syntactic dependencies offer. For instance, the XLE parser and the grammars within DELPH-IN also output variables, for instance to denote number, gender and even animateness. The DELPH-IN community takes this format one step further in the abstractness hierarchy, by using a form of (flat) semantics as their output (MRS: Copestake *et al.* 2005). The MRS formalism allows the grammar writer to specify (or underspecify) semantic scope, and aims to abstract away more from the actual surface form than dependencies do, meaning that MRSs across languages are more similar to each other than less abstract syntactic/semantic representations.

To the extent that it can be assessed properly, I conclude that the complexity of the task Cheetah is evaluated on is roughly as high as the problem DGE parsers and statistical dependency parsers are trying to solve. This is also true for the hand-written Alpino grammar (Van Noord 2006). In principle, Cheetah is also able to output variables, although they have not been evaluated for the studies in this thesis. However, there is still a gap to close with respect to the DELPH-IN grammars, as semantic scope can not be represented. Other differences, for instance the assignment of void semantic representations to a number of categories (particles, auxiliary verbs, complementisers, infinitival 'zu'), can be realised in the core grammar and lexicon with relatively little effort.

Ability to generate

Most attention in this thesis is devoted to parsing. However, the bi-directional nature of DELPH-IN grammars is a distinctive feature (Velldal and Oepen 2005), and is used for (among other things) transfer-based machine translation (Bond *et al.* 2005). One of the prerequisites for doing generation is a grammar that does not suffer too much from over-generation. Technically, it is possible to generate from Cheetah, but no real experiments have been carried out. The main barrier is the exhaustiveness of the generator in the DELPH-IN toolchain, which leads to too large a computational burden even for slightly less constrained grammars.

Generation from treebank-derived grammars is theoretically possible, though. It has been successfully done both with the Penn Treebank-based grammars for LFG (Cahill and Van Genabith 2006) and HPSG (Nakanishi *et al.* 2005). This leads me to argue that such experiments should be within reach for Cheetah as well, if the generation algorithm is adapted accordingly.

Efficiency

As we have seen in section 5.4, Cheetah needs around 4 seconds per sentence on unseen newspaper text. This is already after a certain amount of pruning on the basis of the part-of-speech tags. Cheetah's efficiency is therefore comparable to the ERG. The XLE grammar for English is a little faster, with about 2 seconds per sentence (without pruning; taken from table 2 by Cahill *et al.* (2008b)), or around half a second, if only the core grammar is used (Kaplan *et al.* 2004) (this is the normal grammar, but without support for certain infrequent phenomena, such as topicalisation). Van Noord (2009) does not give average parse times when parsing exhaustively, but using pruning, at least 6 seconds per sentence are needed to get good results. On the other hand, some DGE parsers are fairly slow, because these parsers do not strongly restrict the search space using their constraints: without pruning, parse times for the Enju system (without pruning) are reported to be around 2 *minutes* per sentence (Ninomiya *et al.* 2005). No such numbers are reported for other DGE parsers in the CCG of LFG formalism.

Many DGE parsers have resorted to statistical methods for constraining the search space, both on the lexical (Clark and Curran 2004a; Matsuzaki *et al.* 2007) and phrasal (Ninomiya *et al.* 2005; Zhang *et al.* 2010a) levels, yielding impressive reductions of parse times. Such methods have also proved to be successful for hand-written parsers, even though the constrained nature of these grammars is meant to keep ambiguity rates under control (Cahill *et al.* 2008b; van Noord 2009). It is clear, on the other hand, that such restriction strategies give larger speedups for treebank-derived parsers than for precision-oriented

parsers, because the redundancy in the parse charts of the former type is larger, and hence pruning these unlikely items is easier. It seems that the PET parser is one of the few deep parsers that still use exhaustive parsing, and it is an obvious choice to adopt pruning strategies for the PET parser as well. We will look into this in chapter 6.

Coverage

When writing a grammar by hand, there is a constant trade-off between efficiency/precision and coverage, and the results from section 5.4 are the result of a series of choices between these two. However, it is clear that the set of applications that this parser can be used for, is severely restricted if only results are returned for around 60% of the sentences.

The coverage number for Cheetah does not compare favourably to current state-of-the-art parsers, both treebank-derived or hand-written. The former usually report coverage numbers approaching 100%, the latter around 80%, depending on the type of text. The coverage number reported in chapter 5 (just above 60%) is lower than both the ERG and the XLE parser (for English), and for the German LFG parser (Rohrer and Forst 2006) (although the out-of-the-box coverage of an existing DELPH-IN grammar for German was lower than Cheetah's (Cramer and Zhang 2009)). It seems plausible to attribute this difference to the shorter development time measured in person months, and more syntactic phenomena deserve attention in order to resolve this.

There is also another way to perceive coverage, and that has to do with the way the parser acts when the grammar does not license the utterance. Most current algorithms that are used for parsing with large hand-written grammars employ a form of fallback strategy, for instance by outputting fragments. This lifts the coverage to 100%, even though not all sentences are licensed by the grammar. In this respect, the PET parser is an outlier, because only preliminary work has been done on this type of robustness measures. Chapter 7 will look into a number of strategies to handle such situations.

Development time

One of the main motivations for the workflow behind Cheetah was practical in nature: it takes an extra-ordinary amount of time and expertise to create a wide-coverage grammar manually. This is true for the development of the formal model (constructions, lexical types, lexicon) and for the annotation of a realistically sized treebank. The methods that have been proposed shorten the development time of each of these components. Compared to state-of-the-art

hand-written grammars, gaps in the inventory of lexical types and constructions are spotted with more ease due to the unit testing procedure. Also, the lexicon is derived automatically, and no labour-intensive re-iterations over the lexicon are needed when the inventory of lexical types is modified. Last, a decent portion of the source treebank can be automatically converted to a rich, dynamic treebank in the target formalism using an automatic disambiguation procedure. In total, I would estimate the total time to create the resources to be 2 person years.

The workflow of treebank-derived deep parsers guarantees a shorter development cycle than for Cheetah. For these parsers, the development process is more straightforward, because the shape of the core grammar is considerably simpler[10]. On the other hand, more is required from the heuristic annotation/conversion procedure, for which scalability issues can be expected when more depth is required.

Treebank-derived and hand-written deep parsers are not incomparable categories of parsers. Rather, they should be regarded as extremes on a dimension that indicates the amount of manual effort that is invested into the grammar. Cheetah is located somewhere in the middle of this line, trying to combine the best of both worlds. Compared to the treebank-derived parsers, there is only little more development time, but it has large advantages in terms of the ability to generate and linguistic relevance. On the other hand, compared to hand-written grammars, there is a large speedup in terms of development time, with only a relatively small penalty in terms of depth of the output.

I should make some general remarks here about the cost of the grammar in relation to the treebank. Of course, it also takes labour and expertise to create the source treebank, and one could argue that, if one wishes to compare the development time of different parsing paradigms, the time to create that treebank should be counted in as well. The counterargument is that the goals that treebanks serve are manifold, and that some treebanks have been used in thousands of studies. Therefore, the development efforts of creating the treebank should be divided by the number of beneficiaries of the treebank, which is a large number for the Tiger treebank.

Clean evaluation

The standard type of evaluation within the machine learning community is to split the available data in a training set and a test set (and possibly a development set). The experiments in this thesis clearly follow this paradigm, and learnt parsers (either shallow or deep) generally use this form of evaluation as well. It must be said, though, that this paradigm is not entirely flawless. For instance,

[10]This is called the *specification* step in the development of the Enju parser (Miyao *et al.* 2004).

when a community has converged to a certain test set, evaluations are repeated many times on the test set, with the possible danger that the algorithms that are developed are incidentally geared towards that specific set. Similar comments can be made about the domain of such sets, to which learnt parsers are particularly sensitive, more so than hand-crafted parsers, which are meant to be universal. This criticism does for instance apply to the repeated evaluation on section 23 of the Penn Treebank (Marcus *et al.* 1994).

A distinction between training and testing sets is not applicable to hand-written grammars. The evaluation of hand-written grammars outside the DELPH-IN framework is often done with respect to a stand-off format (*e.g.* Van Noord 2006). When no stand-off annotation is available, which has been the case for DELPH-IN grammars, one might be tempted to report statistics on the coverage or annotation rate on a certain development set, in which the grammar and the treebank are co-developed (Oepen *et al.* 2004). However, such scores are biased, because special attention has been paid to that particular set of linguistic data. Later, results from out-of-the-box experiments, in which the grammar writer has not seen the data beforehand, have been reported for the ERG (Ytrestøl *et al.* 2009; Flickinger *et al.* 2010). Also, work has gone into designing evaluation paradigms that give a finer-grained picture of the quality of a returned parse (Elementary Dependency Match: Dridan 2009), instead of a binary one. What is still lacking, however, is a robust evaluation of sentences that are not licensed by the parser, which, according to Flickinger *et al.* (2010), still forms approximately 15% of realistic data. Because gold standard MRSs within DELPH-IN have only been created when the sentence was covered, this poses a problem for a more granular evaluation paradigm. Cheetah is not affected by this issue, because the gold standard dependencies are available in a stand-off format.

Outlook

The larger question that this thesis is trying to answer is the one of *feasibility*. Most of the work in chapters 3, 4 and 5 was devoted to reducing the workload, and to automate as much as possible in the process. This was achieved, but there were also some downsides. Compared to DELPH-IN grammars, there is a loss of abstraction, as true MRS structures can not be created. The fact that the lexicon was learnt automatically also caused more over-generation and ambiguity than observed in carefully handcrafted grammars. Last, grammars that are entirely learnt by an algorithm have the advantage that (almost) full coverage can be achieved.

From here, two directions of research can be chosen: trying to coerce Cheetah into a DELPH-IN-style grammar or trying to improve on other factors that neg-

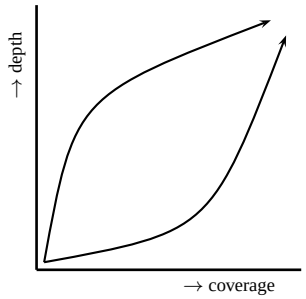


Figure 5.5: A schematic representation of two strategies of creating DELPH-IN grammars.

atively influence the adoption of deep parsing (and HPSG parsing in particular), most notably efficiency and coverage.

Making the grammar a 'true' DELPH-IN grammar requires the output of the grammar to be enhanced. The output of Cheetah is currently geared towards Tiger-style dependencies (although the agent-patient distinction is properly made), meaning that the level of abstraction should be brought to a higher level, in order to reach a view on semantics that is common within DELPH-IN. A number of modifications have to be implemented. For instance, all lexemes that can act as a modifier must specify whether the modification is intersective or scopal, either by changing or specialising lexical types. When a lexical type is specialised into a number of novel types and the information for this specialisation can not be found in the source treebank, a manual pass over the lexicon is necessary to decide with which lexical type that lexeme should be associated. Constructions dealing with modification need to be adapted accordingly as well. Furthermore, the predicates of some lexical entries need to be changed. For example, all personal pronouns need to have the same, abstract predicate name, having only different morpho-syntactic variables. There is a list of syntactically divergent words already (in the core lexicon), and they are most likely the ones that need their semantics changed.

One question that comes to mind when discussing these issues is: what is the most efficient strategy to create precision-oriented deep grammars? Two scenarios can be distinguished (see figure 5.5 for a visualisation of the scenarios). In the first scenario, the developers start out with a high level of abstraction (MRS) from the start, and write the entire grammar manually. However, this method has proved to be very labour-intensive. The alternative method is to follow Cheetah's workflow, and when the grammar is in the state that Cheetah is at this point in the thesis, alter the grammar to make it entirely DELPH-IN style. Cheetah's level of development after two person years of work makes me believe that the second scenario is more efficient.

As mentioned above, there are two other large obstacles that prevent the adoption of large-scale HPSG precision parsing using the PET parser. The first is the relative lack of efficiency. Computing the feature structures is expensive, and so far, the algorithm of the PET parser will search through the complete search space. Some studies have indicated that large speed gains can be expected, without loss in accuracy, and I will investigate in chapter 6 whether those speed-ups can be achieved with the PET parser as well.

The second obstacle is the lack of a good fallback strategy in PET when the parsing stage fails. Rule-based systems in the field of artificial intelligence have always been criticised for their brittleness: if there is input that does not conform to the rules' expectations, it is not obvious how the system should behave. In the case of PET, when the sentence is not covered by the grammar, the parser will fail, and output nothing. However, if precision-oriented HPSG parsing aims to be wide-coverage, better strategies have to be invented. In chapter 7, a number of strategies will be investigated, among them the use of robustness rules in combination with the search space restriction strategies from chapter 6.

6 Agenda-based task pruning

Under the assumption that ambiguity can be tamed satisfactorily by defining constraints, deep parsers have often relied on an exhaustive search through the search space in order to find the best analysis. In this setting, only after constructing the complete parse forest, the analyses are extracted from the forest, and ordered with respect to a certain statistical disambiguation model. This requires the grammar to be very compact, and even then, true ambiguity can be pervasive enough to reach the parser's computational boundaries. Many deep grammar extraction parsers have already resorted to search space restriction (Ninomiya *et al.* 2006; Clark and Curran 2004b), which is a logical choice: these grammars are fairly permissive in the structures that they allow to combine, and therefore, an exhaustive search strategy would cause a large computational burden. Lately, more and more work has gone into restricting the search space for precision-oriented grammars as well (Cahill *et al.* 2008b; Dridan 2009; van Noord 2009), with promising results.

Apart from the pursuit to make the parser faster, there is also a case to make for making parse times more predictable. For instance, imagine that parsers A and B have equal average parse times: 2 seconds. Parser A always takes 2 seconds to finish the parse, whereas parser B usually needs 1 second, but sometimes 10 seconds. In most cases, parser A is preferable because the end user of a system that depends on either parser will not end up being frustrated by unexpected long lags. Hence, predictability is a valuable asset when we evaluate the efficiency of a parser.

In this chapter, a novel type of search space restriction, *agenda-based task pruning*, is investigated. The chapter starts with a comparison of existing methods to see which kind of model is most suitable for integration into the PET parser. Then, the generative model is explained, on which a model for prioritising parser tasks is based. This is followed by the introduction of a number of pruning strategies, along with the results that these models achieve on the development set. The chapter is closed by a discussion of the results on the test set, as well as some directions for further research.[1]

6.1 Introduction

As has been extensively discussed in section 2.5.2, a number of studies have successfully applied search space restriction strategies in order to ameliorate the efficiency of various parsers. Methods to reduce the computational demands of parsers can act on two different levels: lexical and phrasal. Lexical restriction

[1] Parts of this research have been published previously (Cramer and Zhang 2010).

has been proved to be useful for a number of parsers (Clark and Curran 2004b; Van Noord 2006; Ninomiya *et al.* 2006), including the PET parser (Dridan *et al.* 2008; Dridan 2009). Actually, a form of such a restriction is already used in the reference experiments in chapter 5: only those lexical types that are subtypes of the gold standard parts-of-speech were put on the chart. In the experiments described by Dridan (2009), this cut parse times for the ERG (Flickinger 2000) in three, when the input was annotated with automatically assigned parts-of-speech from a tagger. However, this was at the cost of a small, but significant loss of correct readings.

Good results have been obtained with phrasal restriction as well. Cahill *et al.* (2008b) introduced a beam search for each chart cell in the XLE parser, based on a generative model conditioned on LFG's c-structures. They report a 60-70% speedup, with a slight increase of coverage (which is due to less sentences going into SKIMMING mode). Ninomiya *et al.* (2005) show how a beam can be introduced in the Enju parser, although not only a beam on chart cells is defined, but also on the sets of cells with the same span size. Also, each chart cell is assigned a limit of the number of chart items it can contain. The probabilities are based on a discriminative model (without non-local features). Under its best settings, speedups of more than 99% are reported, although it must be noted that an average parse time in the order of minutes was observed when no restriction was applied. Another approach is presented by van Noord (2009), who presents a pruning algorithm, in which statistics on *splines* (a goal category with a number of parser steps to reach that goal) are used to predict which parser steps to filter. The resulting average parse times (with equal accuracy scores) are decreased by a factor between 3 and 4. The innovation of this technique lies in the fact that no annotated training data is needed, because the output of the parser on unannotated text is used as training data[2]. Another interesting point is the possibility to prevent certain tasks to be carried out instead of filtering the results afterwards, as is done by the chart pruning algorithms presented by both Cahill *et al.* (2008b) and Ninomiya *et al.* (2005).

Lexical and phrasal restriction both have their own specialities in terms of what they can prune successfully: lexical methods are more useful to determine whether certain sequences of tags are probable, and phrasal methods have other types of information to their disposal, which can determine whether certain combinations of rules are more likely to be part of the final solution than other candidates. It appears reasonable to assume that a combination of both types of restriction can be complementary, as is done by Zhang *et al.* (2010a) for the CCG formalism.

It is important to realise that the expectations that one should have when

[2]It must be said that the algorithm assumes that the disambiguation model is good enough, which, in turn, is based on annotated data.

applying restriction strategies are directly dependent on the degree of local ambiguity that the grammar allows. For instance, the more care is put into the quality and compactness of the lexicon, the less lexical restriction is going to help. The same holds on the phrasal level: if the grammatical constructions are tightly constrained, there is probably less value in making an attempt to prune chart items. In other words, the less restricted the grammar is, the more densely populated the parse chart will be, and the more effective pruning strategies can be.

Lexical restriction with the PET parser has been investigated before (Dridan 2009), but applying limitations on the number of phrasal items has not. Given the good results for the Alpino, XLE and Enju parsers, it would be interesting to see whether such methods could be applied to the PET parser (Callmeier 2000) as well. However, because PET has a flexible agenda of parser tasks (Erbach 1991) (see also section 2.6.2), existing restriction algorithms are not immediately transferable to PET. For instance, van Noord's (2009) model cannot be used, because the parser does not define splines, over which the guiding statistic is calculated. Furthermore, the methods as proposed by Cahill *et al.* (2008b) and Ninomiya *et al.* (2005) are not suitable either. Their restriction algorithms assume that, at one given point in time, it is certain that all items in a particular chart cell have been found. Restriction on that cell is then applied. In the PET parser, there is no such point in time[3], making it impossible to use these restriction algorithms. The same argument holds for the study by Ninomiya *et al.* (2005), in which also a beam on the set of cell with the same span length level is applied (global thresholding). For this technique, there must be a specific point in time at which all chart cells with a certain span length are known, which, again, is not the case for the PET parser. Also, their approximation of the outside probabilities (Goodman and Shieber 1998) (which is necessary to ensure comparability of chart items with different yields) is impossible to port to the PET parser.

The method that will be developed for the PET parser bears most similarity to the one described by Cahill *et al.* (2008b), because the same statistical model is used: a generative model (to be explained in the next section). However, there is one principal difference between their generative model and the one for PET: in LFG, the c-structures (LFG's context-free backbone) can be used, whereas HPSG does not define such a structure. Instead, the generative model will be conditioned on the HPSG rule application tree (or: derivation tree). It is not the first time that this type of modelling is used in conjunction with DELPH-IN grammars. Toutanova *et al.* (2005) have experimented with such models for parse disambiguation. The main result of this study was that disambiguation

[3]It would be straightforward to create a priority model that triggers such behaviour, however. I will expand on this possibility in section 6.5.

with discriminative modelling is more accurate than with generative models, but that the performance difference between the two is not very large, at least if grandparenting is used in the generative model[4]. The idea of using a context-free backbone of HPSG rule applications is therefore not new, but it has never been used for search space restriction.

The second difference to Cahill's study (and to most other studies) is that the pruning algorithm works on agenda tasks, rather than on chart items. The algorithm will make an attempt to favour tasks that are more promising, in the sense that they have a larger probability to be included in the correct parse. Less promising tasks are either deferred, or discarded altogether if the limit for the number of tasks has been reached. This has the consequence (as already noted by van Noord (2009)) that the execution of unifications is prevented, rather than their results being pruned, with an extra speed-up as a possible advantage.

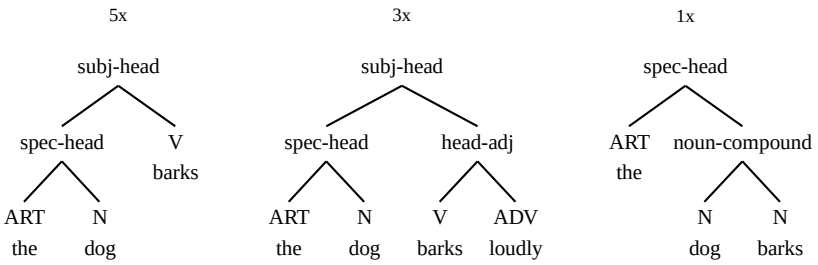
6.2 Task-based search space restriction

6.2.1 Prioritising parser tasks

The implementation of the PET parser is based on an agenda of tasks (see section 2.6.2 for more background information). Each task consists of exactly one unification. Tasks are ordered in a priority queue, and therefore each task is assigned a certain priority. The basic parsing loop starts when lexical items are put on the chart. Each time an item (either lexical or phrasal) is put on the chart, the parser will spawn new tasks trying to combine the new chart item with existing neighbouring chart items. These tasks are executed in order of priority, and whenever a tasks succeeds, a new chart item is created, which in turn causes more tasks to be put on the agenda. This interaction between agenda and chart continues until the agenda is empty, after which the solutions are harvested from the chart.

Currently, the priorities of the tasks are based on a heuristic, aiming to reach the top of the chart as soon as possible. That will be changed in this chapter: the priorities of tasks will be based on whether the task is likely to be part of the final solution or not. As mentioned before, the guiding heuristic will be a generative model of HPSG rule applications. To better understand how the priority model works, let's consider the example in figure 6.1. The treebank contains a total of 9 trees, and a corresponding PCFG (without smoothing) is given in the same figure. A generative model predicts the probability of a given tree (including its leaves), from the top. So, the generative probability of a tree is the product of

[4]One remark is in place here. The used grammar has received considerable attention in the meantime, and the treebanks are now larger and contain more complex text. Hence, it must be seen whether the results from that study extend to the contemporary situation.



ROOT	→	subj-head	$\frac{8}{9}$
ROOT	→	spec-head	$\frac{1}{9}$
subj-head	→	spec-head V	$\frac{5}{8}$
subj-head	→	spec-head head-adj	$\frac{3}{8}$
head-adj	→	V ADV	1
spec-head	→	ART N	$\frac{8}{9}$
spec-head	→	ART noun-compound	$\frac{1}{9}$
noun-compound	→	N N	1

Figure 6.1: The upper part of this figure denotes a treebank of derivation trees (including the counts of the trees, so in total the treebank contains 9 trees). The lower half is a description of the PCFG of rule applications that is derived from the treebank.

the probabilities of all rule applications. Therefore, given this PCFG model, the generative probability of the leftmost tree is $\frac{8}{9} \cdot \frac{5}{8} \cdot \frac{8}{9} \approx 0.493$ (if the generative model is not conditioned on the words). The probability of the rightmost tree is $\frac{1}{9} \cdot \frac{1}{9} \cdot 1 \approx 0.111$. Therefore, if the parser would be presented with the sentence 'The dog barks', the disambiguation component would choose for the first reading, because the generative probability of the first reading is higher (which corresponds to our intuition). In this study, the generative model is not used for disambiguation, but for parser task prioritisation, which means that nodes that are likely to lead to the final solution are carried out before tasks that are less probable. Some less probable items will even be discarded altogether.

There are two options for choosing the symbol that is used for leaf nodes in the derivation tree: the name of the lexical entry and the name of the lexical type. The difference is that two different words with the same properties share their lexical type, but not the name of the lexical entry. Therefore, there are much fewer lexical types than lexical entries, and in order to reduce data sparsity, lexical types are used.

Although the example computation above is top-down, the PET parser works bottom-up, meaning that the actual computation has to follow this order as well. The lexical chart items receive a probability of 1 (we will experiment with this setting in section 6.3.4). Each time a new phrasal item E is created from chart item(s) E_i, using the production rule $\alpha \rightarrow \beta$, where β is either one or two

symbols, the following score is calculated[5]:

$$p(E) = p(\beta|\alpha) \cdot \prod_i p(E_i) \tag{6.1}$$

At first sight, equating the probability of this imaginary chart item with the task's priority seems like an attractive strategy. This can be done, because the generative score of the *possibly resulting* chart item can be computed before the actual unification takes place. However, there is another aspect to this. Consider the following two rule productions, together with their respective frequencies in the training set (where $c()$ is to be read as a count function):

Rule production	$c(\alpha \rightarrow \beta)$	$c(\alpha)$	$p(\beta\|\alpha)$
$\alpha_1 \rightarrow \beta$	10	100	0.10
$\alpha_2 \rightarrow \beta$	2	20	0.10

If there are two adjacent chart items that unify with β, we certainly want the application of α_1 to have higher priority than the application of α_2. However, both rule applications have the same relative frequency. Therefore, the probability of applying a rule α, irrespective of the daughters, must also be included when calculating the priority of a task:

$$Pr = p(\alpha) \cdot p(E) \tag{6.2}$$

where $p(\alpha)$ is the probability of applying rule α, and $p(E)$ is the generative probability of the possibly resulting chart item.

It might be that the generative score of an item needs to be estimated, without having seen that particular combination of rule and daughters in the training set. To do this, a common technique called *smoothing* is used. Smoothing takes away a little bit of probability mass from observed events, and grants it to unseen events. I will use one specific type of smoothing here: Lidstone smoothing (Lidstone 1920), which adds a certain parameter δ to all counts. The unobserved event also gets a count of δ. The result is that if the left-hand side of a context-free rule α is observed $c(\alpha)$ times, and rule application $\alpha \rightarrow \beta$ has been seen $c(\alpha \rightarrow \beta)$ times, the conditional probability is not $\frac{c(\alpha \rightarrow \beta)}{c(\alpha)}$ (which it would be without smoothing), but $\frac{c(\alpha \rightarrow \beta)+\delta}{c(\alpha)+N\delta}$, where N is the number of production rules with the same left-hand side. Unknown productions have $c(\alpha \rightarrow \beta) = 0$, and hence their probability is estimated to be $\frac{\delta}{c(\alpha)+N\delta}$. Typically, $\delta \ll 1$, and has been set to 0.01 in the experiments in this chapter.

An unsolved problem in the priority model is to find a sound solution for the packing mechanism. Let's assume we have a situation, in which the agenda

[5]Naturally, log probabilities were used in practice, to prevent underflow.

contains task $T_{\alpha \to AX}$, that connects chart items A and X using rule α. The priority of this task is denoted as $Pr(T_{\alpha \to AX})$. Later on in the process, chart item B is created, but packed under A. Hence, no new tasks will be spawned. However, it is possible that, were it calculated, $Pr(T_{\alpha \to BX}) > Pr(T_{\alpha \to AX})$. Ideally, the priority of $Pr(T_{\alpha \to AX})$ would be updated to reflect this change. The described problem might not be so large, given that the most probable items are tried first anyway, and combining two items with fairly large probabilities is likely to give more probable items than a combination of two items with lower probabilities. Also, the packed item (B in the example) can still be found by the selective unpacking procedure. There might be solutions to solve this issue, but the priorities will be left unchanged in the experiments in this chapter, in order to keep the algorithm simple.

6.2.2 Task pruning strategies

In this section, I define three *pruning strategies* (LOCAL, STRIPED and GLOBAL[6]), and three *counting strategies* (ALL, SUCCESS and PASSIVE). These dimensions are orthogonal, meaning that 9 different combinations can be made.

The pruning strategy defines how rigidly the tasks are tied to specific chart cells, and the three variants are represented schematically in figure 6.2. The LOCAL strategy assigns a constant number of tasks to one particular cell; STRIPED assigns a fixed number to all tasks with the same span length; GLOBAL defines one number of tasks for the complete sentence. The main parameter is t, which determines how many task slots are allocated for one specific chart cell. The total number of slots depends on both t and the length of the sentence (n), and are divided in such a way that for equal t and n, the total number of task slots is equal for all strategies. For instance, let's consider a sentence of length $n = 3$ (as in figure 6.2), and set the cell size to $t = 1000$. Using the LOCAL strategy, there are $\frac{1}{2}n(n+1) = 6$ chart cells, and hence the total number of tasks slots is $t\frac{1}{2}n(n+1) = 6000$. If the STRIPED strategy is chosen, the number of cells for each span length is set proportional to the number of cells with that size: $t(n-i)$, where i is the span length. There are 3 chart cells with span length 1, 2 cells with span length 2, and 1 cell with span length 3. The total maximum number of tasks is then $\sum_{0<i\leq n} t(n-i)$, which equals $3000 + 2000 + 1000 = 6000$. The total number of slots for the GLOBAL strategy is set by the agenda to $t\frac{1}{2}n(n+1)$, which is 6000 as well in this example.

All strategies try to cut out parts of the search space that are unlikely to contribute to the final result. However, the evidence on the basis of which the

[6]Note that what is called STRIPED in this chapter, is acting on the same level of the chart as the GLOBAL pruning strategy as defined by Ninomiya *et al.* (2005).

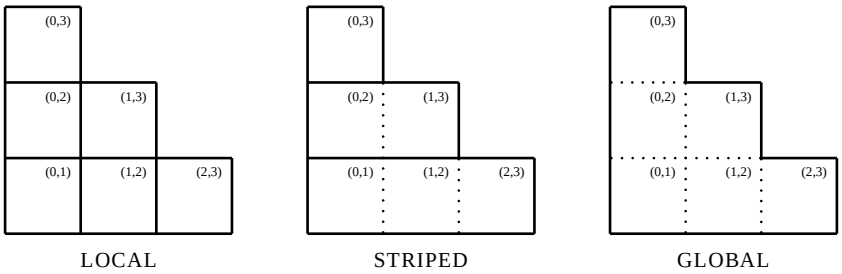


Figure 6.2: A schematic representation of the three different restriction strategies. In LOCAL, each cell is closed. All cells with the same span length are in one counting space in the STRIPED strategy. In GLOBAL, all cells are in the same counting space.

pruning strategies base their decisions is different. The LOCAL strategy uses the availability of many chart items in one cell as a pointer that some of them will not be necessary. However, it can not discriminate between good and bad spans, and hence, the bad spans will receive an equal amount of attention as the good ones. A tightly constrained grammar can only partially prevent this. On the other hand, the strategies with a wider scope (STRIPED, GLOBAL) do allow 'leaking' of attention between the different chart cells, but these strategies bear the risk that one chart cell containing many tasks with high priorities will use up the complete reservoir of available tasks in one counting space, leaving no slots for other potentially useful chart cells. As we will see later in this chapter, the comparability of priorities between different chart cells is a major concern, and will prevent the successful deployment of STRIPED and GLOBAL.

Orthogonal to the pruning strategies, there are also three different counting strategies: ALL, SUCCESS and PASSIVE. In the first case, all executed tasks are counted. The second strategy only counts successful tasks, *i.e.* ones whose unification succeeds. The third strategy will only count those tasks that have led to a passive item on the chart for that particular span. The packing mechanism, although it is used in the experiments, does not affect the counting: if a task succeeds, but the resulting chart item is packed, it is still counted. Irrespective of counting strategy, only phrasal tasks are counted, including ones having a span length of 1, because preliminary experiments showed that exhaustive parsing for inflectional and lexical rules was necessary to get good results.

One thing that was left unanswered so far is how the span of a certain task can be determined. For chart items, as done in other pruning studies, this was obvious, because their algorithms acted on the chart item level. However, the definition of the span of a task is less straightforward. For unary RULE+PASSIVE and binary ACTIVE+PASSIVE tasks (see figure 2.13 for an illustration of these types of tasks), we already know all daughters of the resulting chart item, and hence we can just project the span of this chart item to be the span of the task.

This is not applicable to binary RULE+PASSIVE tasks, because the span of only one of the two daughters is available to the algorithm. As an approximation, the span of the only known daughter will be considered the span for this task.

6.3 Experiments

In this section, we will see how the different strategies work in a number of settings. As we are trying to find the optimal parameters to restrict the search space, the first experiments are carried out using the development set. When the best settings are found, these will be used for the test, so that the results can be compared to the reference results from chapter 5.

In the result figures that come, five different metrics are used to measure the success of the pruning strategies. On the sentence level, the proportion of sentences that are covered and that give the exact parse are recorded. On the dependency level, recall, precision and f-score are recorded. The x-axis is measured in average parse time that the parser needed to complete the parse. Each dot in a graph represents one particular trade-off between time and quality, and is determined by the parameter t, the size of the cell: larger t should result in higher parse times, but also higher quality. Two points of reference are given. First, there is the situation where no task pruning is done. This is indicated by the single, solid point at the right side of the graphs, along with the horizontal dashed line. Second, a priority model is used where each task receives a random priority between 0 and 1, but pruning is still executed. Hence, this baseline is represented as a trade-off between time and quality as well (the dashed curve). For this baseline, the best of the three counting strategies is taken, to make the baseline competitive. For all pruning strategies, the PASSIVE counting method turned out to perform best.

6.3.1 Find the same solution faster

The first experiment in this chapter tests the price of the generative model. Instead of the original priority model, the generative model is used. The idea is that for those sentences that hit the timeout limit, the good solution might be found within the 60 seconds more often, because promising tasks are carried out earlier. No pruning takes place. The result of this experiment is promising (table 6.1). The overhead of the additional generative model is small. A small increase in coverage is observed, showing that more often a correct solution can be found for some of the sentences that reach the computational limit (60 seconds).

	Reference	+PCFG
Sentence-level		
Avg parse time	3.92s	4.16s
Coverage	60.4%	60.5%
Exact match	21.7%	21.7%
Dependency-level		
Recall	40.8%	40.8%
Precision	83.7%	83.8%
F-score	54.9%	54.9%

Table 6.1: This table shows the performance of the parser on the test set, the only difference being whether the heuristic priority model or the priority model based on the generative model is used.

6.3.2 Scope of pruning and counting strategies

A large set of results is depicted in figures 6.3, 6.4 and 6.5, for the three different restriction strategies LOCAL, STRIPED and GLOBAL, respectively. Glancing over all graphs, a number of global observations can be made. The first is that all curves follow the expected trend, with higher values for t yielding better results. The second is that the precision remains high and constant (between 80% and 84%), independent of the type of pruning. The random models perform reasonably well in terms of precision.

Of the three pruning strategies, the LOCAL strategy is performing best. For instance, when the cell size is set such that the average time is about half a second, for almost all sentences the optimal f-score has been reached already, showing a tenfold speedup relative to the situation where no pruning is applied. In this setting, the coverage increases slightly (from 63.5% to 65.4%), because sentences that were too long and/or complex to parse in the exhaustive setting, now become feasible to parse within the 60 second window. However, no additional exact matches are found. The recall number shows a larger increase (from 42.5% to 45.4%) over the unpruned setting compared to the increase of the coverage number, because the newly parsed sentences are longer on average. Precision is slightly below the one in the unpruned setting (84.1% versus 83.5%). Instead of looking at the point where the optimum is reached first, the intersection with the horizontal dashed line can also be observed. In other words, how long does it take to get an f-score equal to the unpruned setting? It turns out that this is achieved with an average parse time of 0.164 seconds, a speedup of 96.6% (roughly 30 times faster).

Significantly worse is the performance of the STRIPED strategy. Compared to the other two pruning strategies, there is a large divergence between the different counting strategies: PASSIVE is showing significantly better scores than ALL and SUCCESS. The curve of ALL is even below the RANDOM-PASSIVE baseline. However, even though the coverage is slowly converging towards the score for the non-pruned parser, the baseline score for exact match is reached

more quickly. The baseline f-score is reached after at an average cost per sentence of around 1 second (with the PASSIVE counting strategy), which is almost a five-fold speedup. Precision scores are slightly higher than for the LOCAL pruning strategy.

The GLOBAL strategy shows the least attractive time-quality trade-offs, with all curves scoring below the RANDOM-PASSIVE baseline. Only the curve for exact match follows the baseline. No large difference between the counting strategies is recorded.

One remarkable finding is that the RANDOM-LOCAL model scores surprisingly high, and when the graphs are compared, it performs better than RANDOM-STRIPED and RANDOM-GLOBAL, which suggests that a per-cell restriction is, *a priori*, the most sensible type of restriction, regardless of the counting model. In fact, the RANDOM-LOCAL restriction shows better time/quality trade-offs than the curves for STRIPED and GLOBAL that do use the priority model, which is a somewhat disappointing result. I will analyse the causes of this behaviour below.

6.3.3 Adjusting global priorities for span length

We have seen before that the GLOBAL strategy performs poorly. It also seems that this is due to lack of coverage, as the metrics that indicate the correctness of the results (precision, exact match) are fine. One of the possible causes is that the priorities of tasks with different span lengths are compared directly, which may be inappropriate. However, if one thinks about the effect of a rule application, the probability of a chart item will always be smaller (or equal) than the product of its daughters. This has a negative effect on the priorities of tasks aiming to create items high up in the chart. The consequence of this strategy is that practically exhaustive parsing is done at the bottom of the chart (items with small span lengths), and that when the top of the parse chart is reached, the limit on the number of tasks has already been hit.

To counter this problem, an extra experiment has been conducted. If one considers the reduction of the log probability caused by a rule application as a price, then one could imagine that the *average* price of a rule application within the complete subtree is a good heuristic for determining whether that subtree is a promising candidate or not. This is achieved by using an alternative formula for the priorities:

$$Pr = \frac{\log(p(R)p(X))}{n} \tag{6.3}$$

where n is the span length, which is an approximation of the number of rule applications, because unary rules are not counted in this metric. The consequence

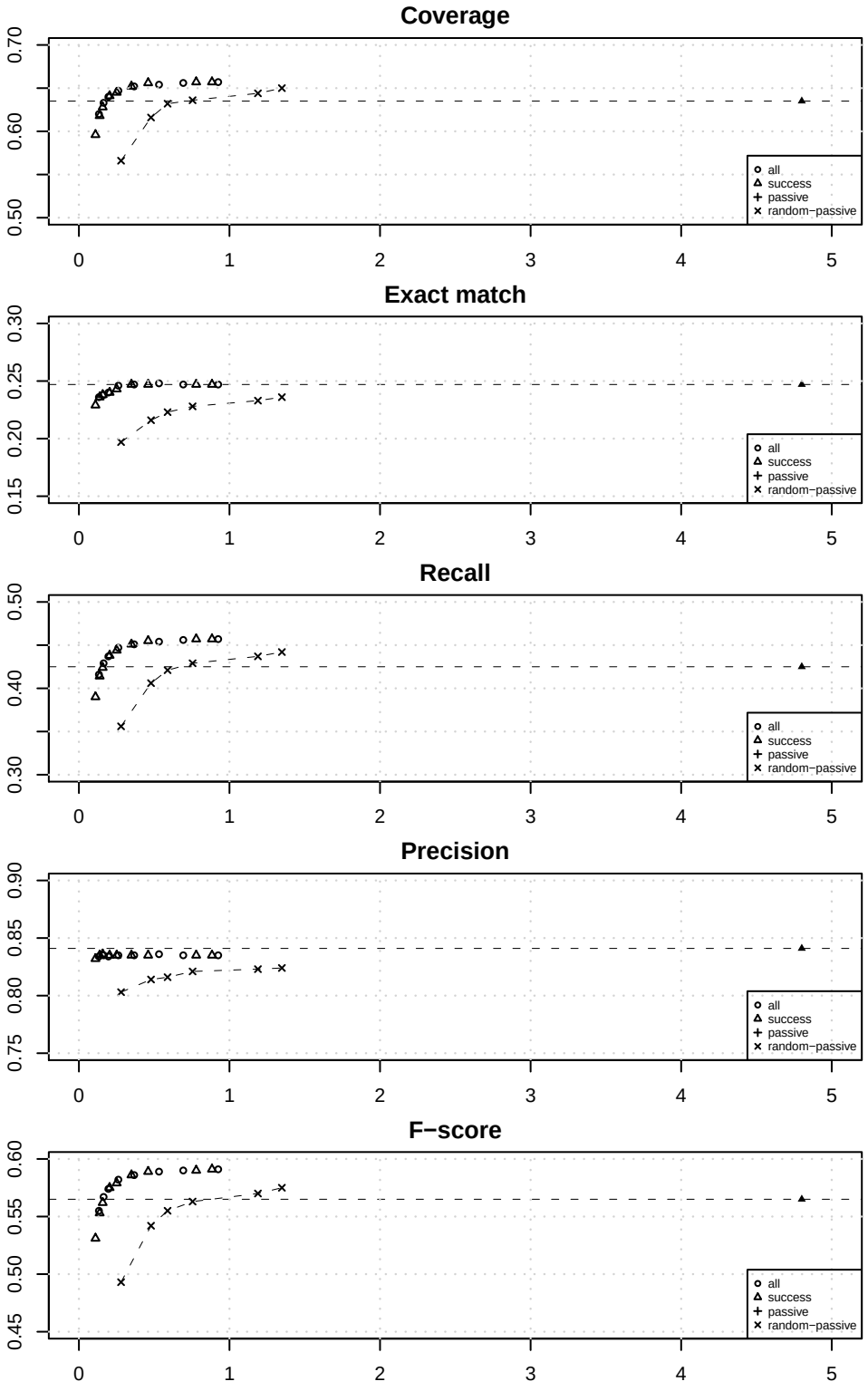


Figure 6.3: These graphs show the results for restriction using the LOCAL strategy.

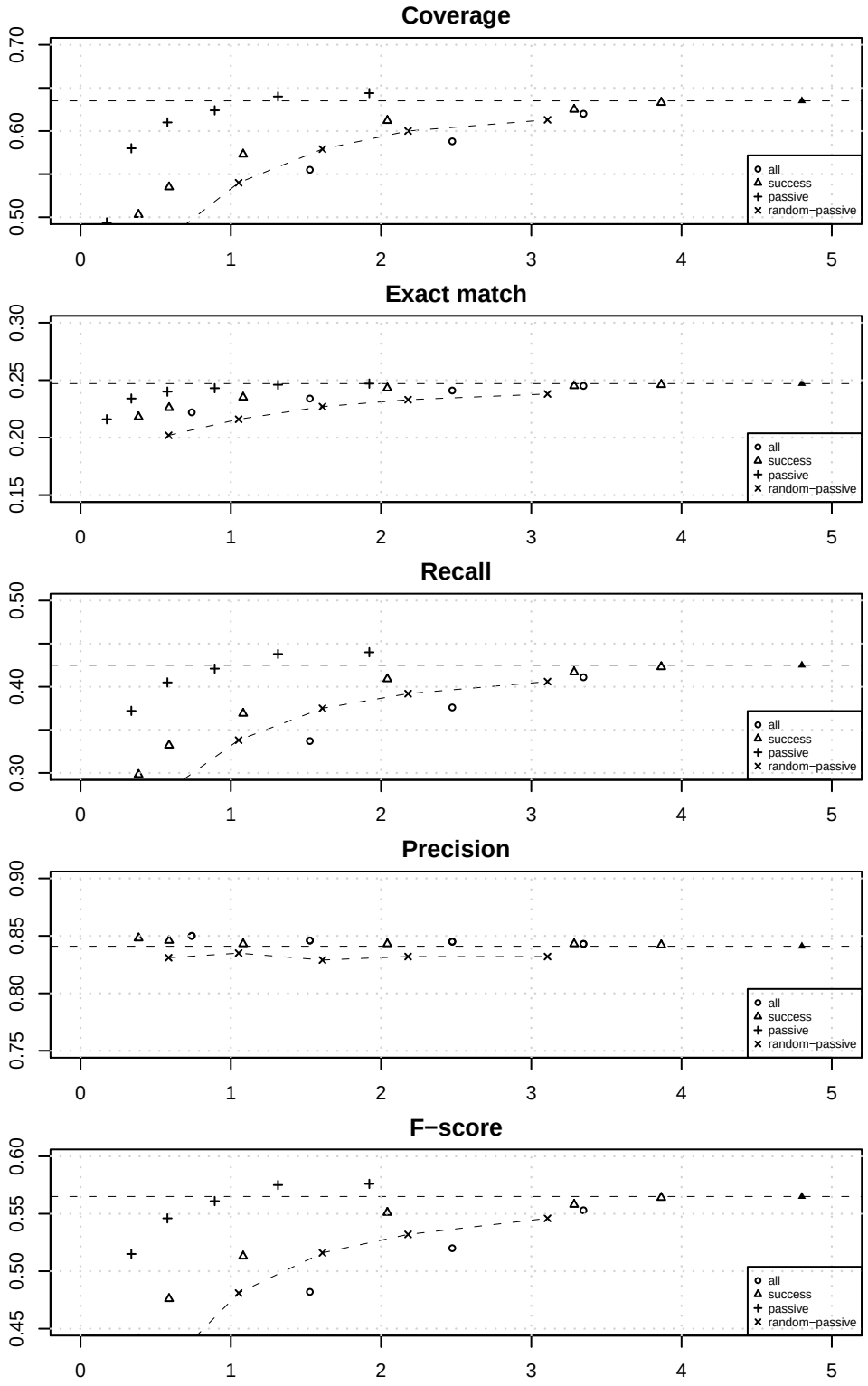


Figure 6.4: These graphs show the results for restriction using the STRIPED strategy.

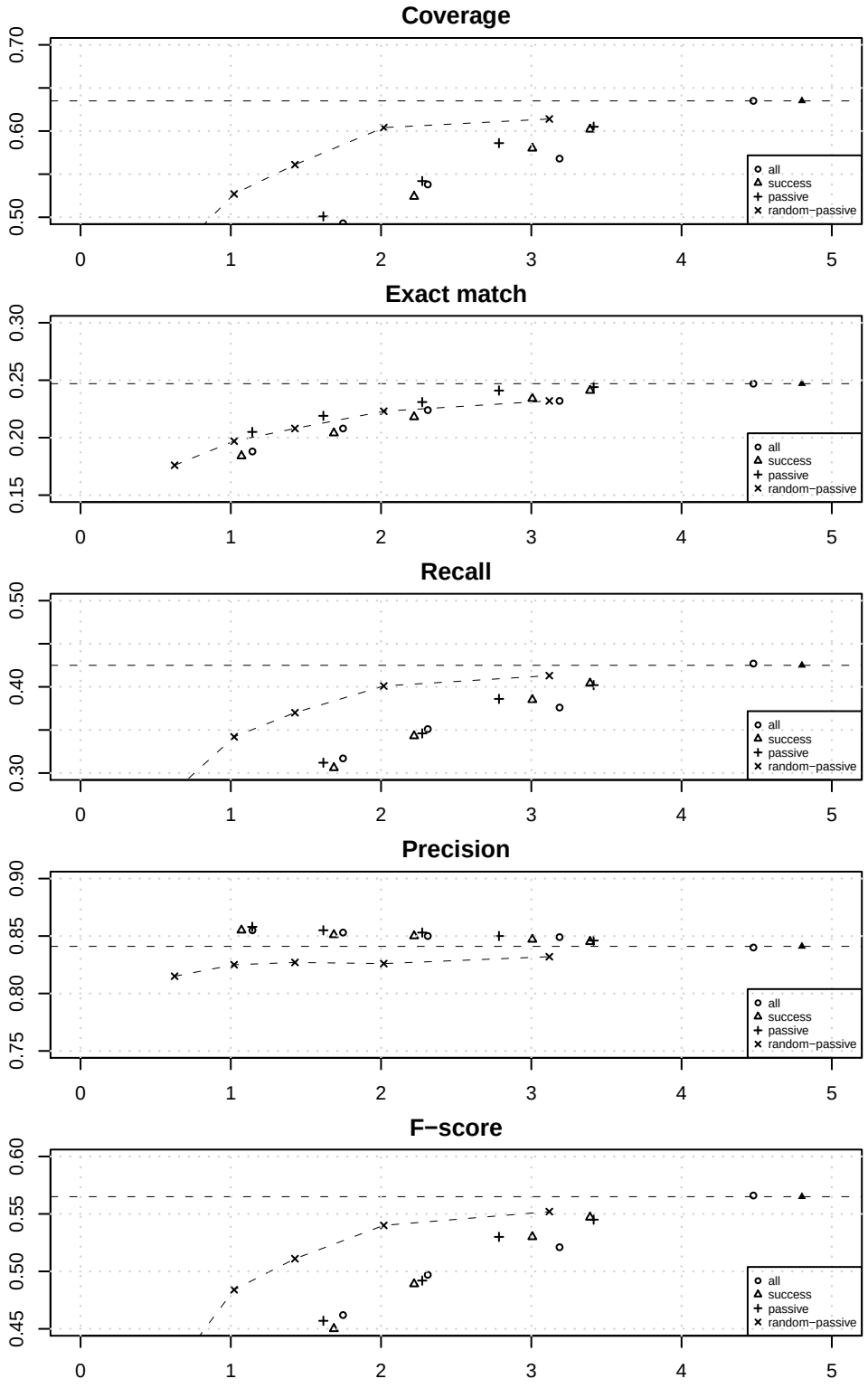


Figure 6.5: These graphs show the results for restriction using the GLOBAL strategy.

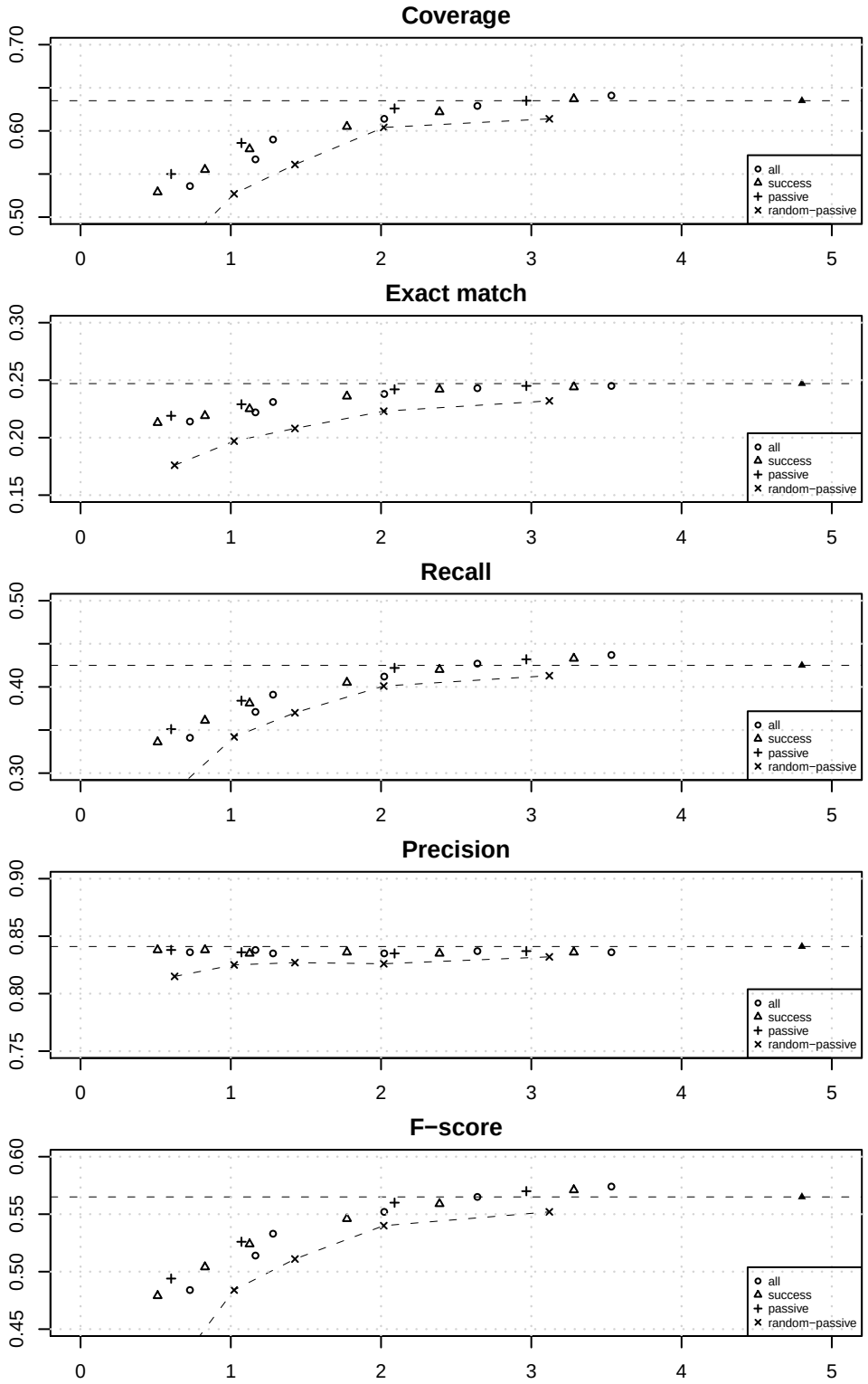


Figure 6.6: These graphs show the results for restriction using the GLOBAL strategy, with the span length correction.

is that the unary rule applications will be dispreferred slightly.

The results are shown in figure 6.6, and they show a remarkable improvement over the initial GLOBAL strategy: all curves are now above the RANDOM-PASSIVE line. Equal results to the exhaustive strategy are reached now at an average parse time of almost 2.6 seconds, which entails a 55% speedup. However, this result is still below the random baseline for the LOCAL pruning strategy.

This illustrates how the lack of *comparability* of the generative scores between chart items in different cells can form a major problem. The technique discussed here is an attempt to ameliorate this between chart items with different span lengths. The same problem crops up for chart items with the same span length, but different spans, which can affect the results for the STRIPED strategy. For instance, consider span $(a, a+m)$, which contains words that only spawn high-frequency lexical types. Another span $(b, b+m)$ contains words that spawn lexical types with lower probabilities. Tasks with span $(a, a+m)$ are more likely to get high priorities than tasks with span $(b, b+m)$, although there is no reason to assume that span $(b, b+m)$ needs less attention than span $(a, a+m)$. Preliminary experiments were conducted to counter this (for instance by penalising frequent leaves), but this had no effect.

6.3.4 Conditioning on tree leaves

So far, the stochastic models assumed that the words were not there. The models also assumed that all lexical items that a word spawned should receive equal probabilities. However, we might also be interested in the probability that a lexical type is spawned by a word. If the distribution of lexical types for a specific word is very different from the *a priori* distribution of lexical types, it might be advantageous to condition the lexical types on the word as well. In the example in figure 6.1, this could be imagined by adding an extra layer of words in the rule set, with rules such as 'ART $\rightarrow$ the'. The conditioning in this experiment is done on the basis of the word form, not on the lemma.

The results are shown in figure 6.7. Only results using the LOCAL restriction are given, using different counting strategies. The results are nearly identical to the curves shown in figure 6.3, with a slightly quicker convergence to the maximum score. An f-score higher than the reference score is already found with an average parse time of 0.140 seconds, which is more than 34 times faster: the best result so far.

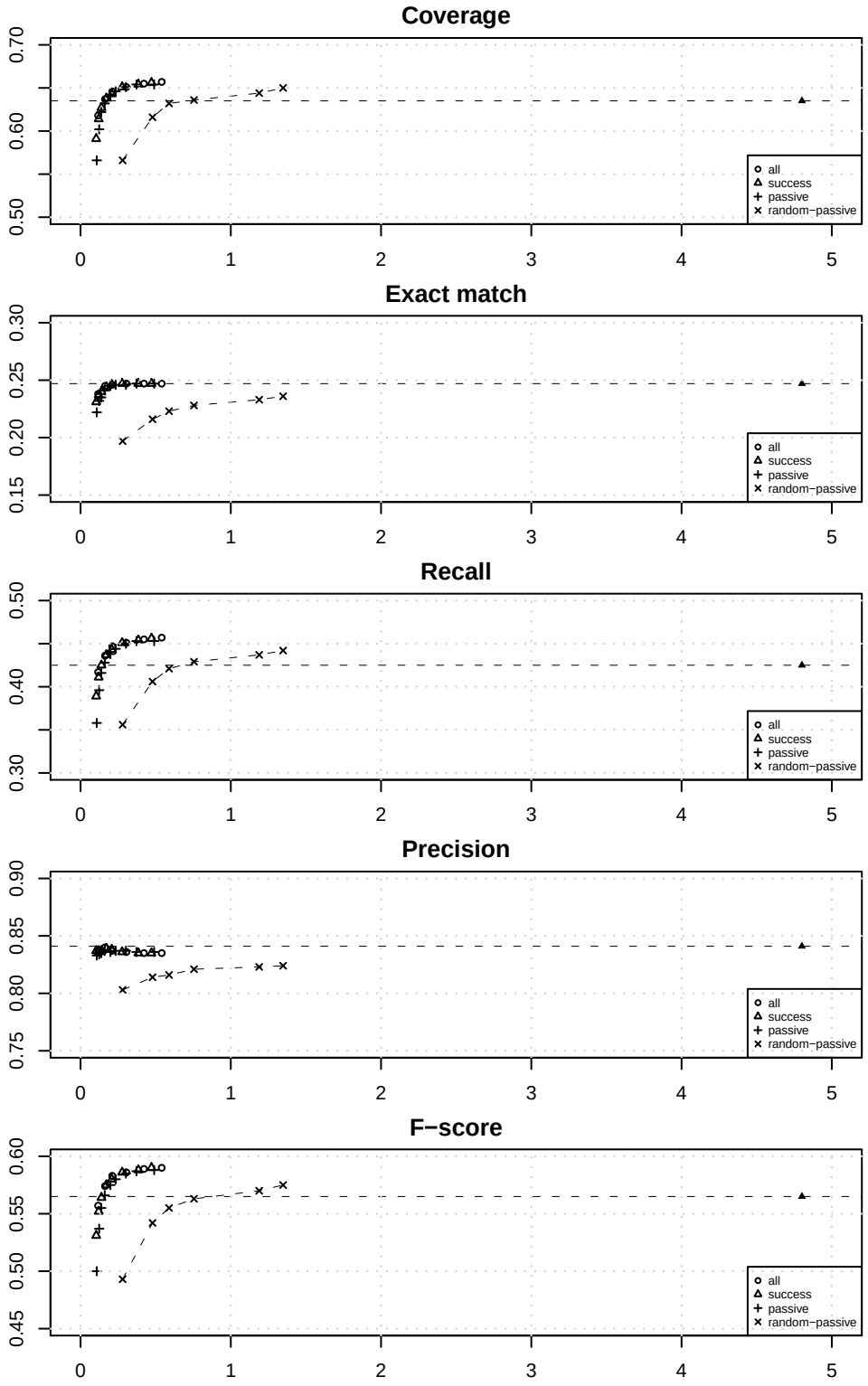


Figure 6.7: These graphs show the results for restriction using the LOCAL strategy, with lexicalisation.

	Unrestricted	$t = 250$	$t = 750$
Sentence-level			
Avg parse time	3.92s	0.19s	0.42s
Coverage	60.3%	60.4%	61.9%
Exact match	21.7%	21.4%	21.7%
Dependency-level			
Recall	40.7%	41.1%	42.5%
Precision	83.7%	83.7%	83.3%
F-score	54.7%	55.1%	56.3%

Table 6.2: Depicted are the results of applying the restriction technology to the test set. The pruning strategy used is LOCAL; counting strategy is ALL. Lexical conditioning is used. Two different cell sizes t are shown, and compared to the reference result from chapter 5.

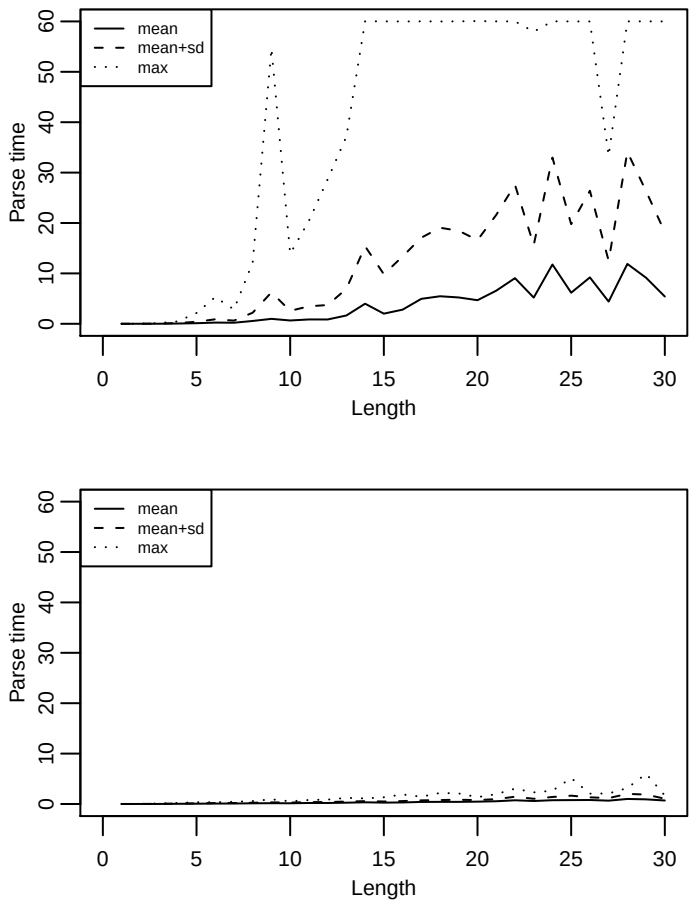


Figure 6.8: Depicted are two graphs that indicate how sentence length and parse time are related. The top graph depicts the variant in which no search space restriction is applied. The bottom graph depicts the restricted variant (LOCAL, ALL, $t = 750$). For each graph, the bottom line gives the mean parse time; the middle line the mean parse time plus the standard deviation; the top graph the maximum parse time.

	All		Near-lexical		Not near-lexical	
Setting	Exec	Prun	Exec	Prun	Exec	Prun
Exhaustive	1046	0	167	0	879	0
LOCAL, ALL, $t = 1500$	690	192	127	37	564	154
LOCAL, ALL, $t = 750$	513	209	99	63	414	147
LOCAL, ALL, $t = 250$	280	210	60	97	220	113

Table 6.3: This table shows how many tasks are executed and discarded, for different task pruning settings, for 50 sentences of length 20 from the Tiger test set. For clarity, all numbers have been divided by 1000. The meaning of 'near-lexical' is explained in the running text. The abbreviations 'exec' and 'prun' stand for 'executed' and 'pruned', respectively.

6.4 Evaluation

The results from the previous section have been reached after much experimentation, and the question is whether the results extend to the test set as well. As can be seen in table 6.2, this is largely the case. The reduction of average parse time (without loss in f-score) turns out to be 95%, although the figure for exact match decreases slightly. Also, when somewhat larger margins are allowed, the speedup decreased to around 90%, but the proportion of covered sentences increased with 1.6% percent point, and the f-score increased with 1.9 percent point. Compared to the other search space restriction study with PET (Dridan 2009), the reduction in parse time seems to be significantly higher (with the reservation that different grammars have been used), and with almost no decrease in accuracy.

Another question is the one of predictability. If we know the length of a sentence, can we be sure that the reaction of the parser is within certain bounds[7]? Figure 6.8 is meant to give an impression of how parse times are divided. The lowest line indicates the mean parse time necessary to complete the parse forest. The second line shows the mean parse time plus one standard deviation. The maximum recorded parse time is given by the dotted line. It can be clearly seen that for sentences up to length 30, the restricted variant is not only much faster, but also much more predictable: all sentences can be parsed within 6 seconds.

6.4.1 What is pruned?

One aspect of the search space restriction deserves extra attention: *what* is actually being pruned? To test this, 50 sentences from the test set that consist of 20 words are selected and parsed. This is slightly longer than the average sentence length in the Tiger treebank, but this can give a better insight in the nature of the pruned tasks than for sentences that hardly benefit from search space restric-

[7]Of course, this is only one way to define predictability. Predictability can also be assessed with a regression model, for instance.

tion. All tasks that the parser either executes or discards are recorded, including the type of task (RULE+PASSIVE or ACTIVE+PASSIVE), the span of the task and the name of the rule. A task is called *near-lexical* if the span length is 1 (for RULE+PASSIVE tasks) or 2 (for ACTIVE+PASSIVE tasks). This means that near-lexical tasks only combine lexical chart items.

The outcome of this experiment is given in table 6.3. As can be expected, the total number of tasks that is executed is lower when stricter pruning settings are used. The number of pruned near-lexical tasks is higher with lower t. It is interesting, however, to note that the number of pruned tasks that are not near-lexical is *lower* with more aggressive pruning of tasks. This observation is not as illogical as it seems at first sight, though: pruning of a task might result in one chart item less to be put on the chart, which in turn might prevent a large quantity of new tasks to be spawn. Therefore, the lower in the parse chart the pruning is applied, the larger the possible accumulated effect is. Given table 6.3, I conclude that the effect of the task pruning strategy, although originally set up as a phrasal restriction mechanism, is mostly due to pruning at the near-lexical level.

6.5 Directions for future research

As argued in section 6.1, existing methods for search space restriction were not immediately portable to the PET parser, mostly because the flexible agenda of PET does not define a single point in time at which it is certain that a chart cell will not receive new chart items. The experiments in this thesis have thus focused on pruning the search space on the basis of agenda tasks. However, the agenda can be changed in such a way that a fixed order of chart cells is enforced. The array of techniques as introduced by, for instance, Ninomiya *et al.* (2005) and Cahill *et al.* (2008b) will then be available, and it will be interesting to directly compare the effectiveness of methods that prune agenda tasks and methods that prune chart items.

Another point of attention is that the current statistical model underlying the priorities has its limitations. The main issue is that grammars that are used in conjunction with the PET parser fairly often make use of unary rules. On the lexical level, chains of length 3 or 4 are not uncommon. The use of unary rules (either lexical or phrasal) introduces two distinct sources of inaccurate modelling:

- It is always true that a unary node has a lower (or equal) probability than its only daughter, and the priority model has no good way to compensate for this inequality. Hence, the generative model is biased towards trees with less unary rule applications. Discriminative modelling does not suffer from

this problem, and therefore, basing the priorities on a discriminative model might be a sensible thing to try. A similar approach for a greedy best-first strategy was already tried by Zhang *et al.* (2007b), but was not used for pruning the search space.

- A unary rule application can hide valuable information from the generative model. A good example of this within the PET parser is that inflectional rules are between the lexical type and phrasal rules. It would make sense to condition phrasal rule applications on the lexical type rather than on the inflectional rule. For instance, if a lexical type reveals that it has no complements, an instance of the HEAD-COMPLEMENT rule is not very promising. However, the intermediate inflectional rule application blocks this inference. Incorporating more context, such as grandparenting information might help to overcome this issue, as has also been suggested by Toutanova *et al.* (2005).

6.6 Summary

In this chapter, a number of strategies to reduce the computational requirements for the agenda-based PET parser have been presented. The underlying model is a generative model, based on HPSG rule application trees, which has been trained on the treebank that was created in section 5.3. The leaves of the tree are the lexical types. The priorities of the tasks on the agenda (a priority queue) are based on the generative scores of the possibly resulting chart item, corrected with the probability of the applied rule, in order to promote these tasks within the agenda. The exact way of pruning was determined by three factors:

Chart cell size The limit that is set on the number of task slots per chart cell.

Pruning strategy Three different pruning strategies have been introduced: LOCAL, STRIPED and GLOBAL, differing in the degree to which the tasks were allowed to 'leak' to other chart cells.

Counting strategy What exactly is counted is defined by the counting strategy. Three counting strategies have been defined: ALL, SUCCESS and PASSIVE.

The results for the different pruning strategy were mixed. The strategies that offered the parser flexibility in terms of which chart cells it could give more attention (GLOBAL and STRIPED) performed significantly worse than the rather rigid allocation of attention (LOCAL). In fact, the random baseline of the LOCAL strategy performed surprisingly well. The differences between the different counting strategies were small, except when the STRIPED pruning strategy was used. A last experiment, in which the lexical items on the chart were also

conditioned on the word form gave a small extra improvement. Final results on the test set, using the LOCAL pruning strategy, indicated that a speedup of 95% can be achieved without negatively affecting f-score. With slightly less aggressive pruning, the average parse time can be reduced by 90%, resulting in a small increase of coverage, recall and f-score. It was also showed that the parse times were not only lower, but also more predictable, as no slow outliers were recorded for sentences up to length 30. Last, additional analysis revealed that the most effective pruning takes place at the near-lexical level, because the effect of pruning accumulates to higher levels.

7 Improving parser robustness

In the previous chapter, we have looked at one factor that bars wide adoption of deep parsing technology: lack of efficiency. Another factor, fragility, is addressed in this chapter. Fragility can be defined as the inability of the parser to give an output for an unexpected input. Unexpected inputs can be, for instance, very long and/or complex sentences exhausting computational resources, incorrect punctuation or sentences that the underlying grammar does not identify as part of the language (extra-grammaticality). In this chapter, the focus is on the latter type of fragility, and two methods (fragment parsing, robustness rules) to improve the PET parser on this aspect are applied and compared.

7.1 Fragment parsing

The most widely used strategy within the deep parsing community to cope with extra-grammatical input is *fragment parsing* (or: *partial parsing*). If there is no chart item that spans the complete sentence and fulfills the root condition, several existing chunks are combined in order to create an analysis that does span the entire sentence. There are a number of variations of this idea, depending on how it is decided which combination of chart items is the best one. For instance, a heuristic that selects the partial parses minimising the number of chunks is a sensible option (Riezler *et al.* 2001). A heuristic on the basis of a shortest-path algorithm is given by Kasper *et al.* (1999), in which the chart is depicted as a directed acyclic graph (a similar approach has also been used by Van Noord *et al.* (1999)). In this graph, each word w_i is a vertex, and an edge (w_i, w_j) represents a chart item spanning words w_i to w_j. Each edge is associated with a certain cost/length, and the shortest-path algorithm tries to find the path (or: combination of chart items) minimising that cost. In this case, a fragment parse is defined as that shortest path. The cost function that was used was fairly simple. Phrasal items receive a cost of 1, lexical items a cost of 2. Input items (items that did not pass lexical processing) were given a very large penalty, practically prohibiting including input items from being included in the partial parse.

This cost function was extended by Zhang *et al.* (2007a) to include the Maximum Entropy score of the item. Therefore, their cost function is an additive combination of the segmentation score and the Maximum Entropy scores of the combined passive items. Because there can be tens of thousands of passive items on the chart, and the segmentation score function is difficult to compute, finding the shortest path is computationally expensive, and therefore two different approximations are used. In *Model I*, only the sum of the Maximum Entropy scores of the individual spans is computed, leaving the segmentation aside. For

each chart cell, the best passive item can be found, and only that item has to be considered when finding the shortest path, making the search feasible. *Model II* first finds a restricted number of reasonable segmentations, after which the entire subgraph can be searched. Because the search space is not searched exhaustively, the algorithm can not guarantee that the optimal solution will be found. Although the experiments were carried out successfully, an evaluation against a gold-standard target data set was not possible. Instead, they used the output of the RASP parser (Briscoe and Carroll 2006), and computed the normalised distance between RASP's RMRS output (Copestake 2007) and the partial parse as output by the different models they evaluate. *Model II* clearly outperformed the shortest-path algorithm by Kasper *et al.* (1999) and *Model I* on this evaluation.

In contrast to the partial parsing algorithm by Zhang *et al.* (2007a), the algorithm that is introduced here is able to find the optimal solution (which minimises the cost function), even though the best partial parse is defined both in terms of properties of the segmentation and the quality of the individual items. The first step is to define a score $s(t_i)$ for a passive item t_i:

$$\begin{aligned} s(t_i) = & \cdot \text{MaxEnt}(t_i) \\ & - c \cdot \text{IsLexItem}(t_i) \\ & - c^2 \cdot \text{IsNotRoot}(t_i) \\ & - c^3 \end{aligned} \tag{7.1}$$

Four terms are distinguished in this score function. $\text{MaxEnt}(t_i)$ is the score that the disambiguation model returns for the tree that t_i represents (as retrieved by the selective unpacking mechanism (Zhang *et al.* 2007b)). $\text{IsLexItem}(t_i)$ is a function that returns 1 when t_i is a lexical item, and 0 otherwise. Similarly, $\text{IsNotRoot}(t_i)$ is a function that returns 1 when t_i is not compliant with any of the root conditions. Each item also receives a constant penalty of size c^3. The score of a partial parse T is then the sum of the scores of the individual subtrees[1]:

$$s(T) = \sum_{t_i \in T} s(t_i) \tag{7.2}$$

The behaviour that will emerge from this specific score function is reminiscent of Optimality Theory (Prince and Smolensky 2004), which defines a strict ordering of violable constraints[2]. For instance, both $\text{IsNotRoot}(t_i)$ and $\text{IsLexItem}(t_i)$ are constraints that can be violated, but the penalty for violating the $\text{IsNotRoot}(t_i)$ constraint in one of the subtrees is much larger than for

[1] One might use the score $s(T)$ as a confidence score, which can be useful information for an upstream application using the parser.

[2] Thanks to Emily Bender for this insightful comment.

	No FP	FP		
	$t = 750$	Unrestricted	$t = 250$	$t = 750$
Sentence-level				
Avg parse time	0.42s	3.94s	0.19s	0.42s
Coverage	61.9%	99.8%	99.8%	99.8%
Exact match	21.7%	22.0%	21.7%	22.0%
Dependency-level				
Recall	42.5%	58.2%	58.2%	59.3%
Precision	83.7%	79.1%	79.1%	79.0%
F-score	54.7%	67.0%	67.0%	67.7%

Table 7.1: This table shows the results of applying the fragment parsing algorithm to the test set, compared to a particular setting of t (the restriction parameter), taken from table 6.2.

violating the IsLexItem(t_i) constraint. The largest penalty is a constant (c^3), penalising the mere existence of a subtree. Effectively, this constant penalty has the consequence that the algorithm will always prefer combinations with fewer chunks. The OT behaviour of the constraints only emerges if c is set to a large enough value, meaning that a combination of violations of lower-ranked constraints can not override a single violation of a higher-ranked constraint. In the experiments in this chapter, c is set to 100.

The algorithm to find the fragment parse T that maximises $s(T)$ is straightforward. Because the segmentation preference is already intrinsic in each subtree, no real search has to be performed. First, the t_i with the highest score is retrieved for each chart cell (using selective unpacking). This is followed by a simple 1-best CYK parsing algorithm to find the best combination of subtrees, maximising the total score $s(T)$.

An indication of the performance of the partial parsing algorithm with this cost function is given in table 7.1. As expected, almost the entire test set receives an analysis, which is accompanied by a rise of dependency recall from 42.5% to 59.3% ($t = 750$). However, this is at the cost of lower precision, indicating that the additional set of covered sentences is of lower quality. This comes as no surprise, given that the grammar does not license these sentences in the first place. The partial parsing algorithm is a fall-back strategy, after all. The reader might wonder how the exact match rate can be higher if no dependency is added when two chunks are combined. This is due to the acceptance of one-word utterances, of which the only word is not recognised as a valid root normally. The fragment parsing algorithm will consider this item as a valid root, assigning it the ROOT dependency label.

7.2 Robustness rules

Fragment parsing has been the standard strategy within the deep parsing community to deal with extra-grammatical sentences. Here, a novel method to re-

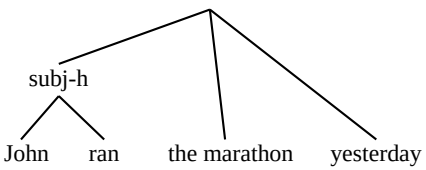
duce the fragility of the parser will be introduced: *robustness rules*. Two types of robustness rules can be distinguished, both of them extending the set of sentences that the grammar accepts, but with different aims in mind. The first type of robustness rules aims to counter specific combinations of chart items. This is similar to the approach that was taken to do grammar and style checking in the CHECKPOINT project (Crysmann *et al.* 2008), in which extra rules were added to give an analysis for extra-grammatical sentences. The use of a so-called *mal-rule* would then pin-point the location and nature of the mistake. For instance, a special rule was made to detect subject-verb disagreement. The second type of robustness rules consists of rules that aim to relax existing constraints on a wide scale. They are over-accepting, *i.e.* sentences that would otherwise be rejected by the grammar will now be given an analysis. It is this type of robustness rules that is experimented with in this chapter.

7.2.1 Motivation

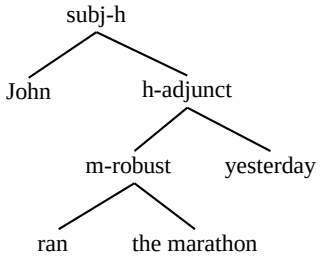
The robustness rules (henceforth: RRs) in this chapter are also meant to accept extra-grammatical sentences. In contrast to mal-rules, however, the RRs are not aiming to cover one particular combination of constituents, but are rather meant as a catch-all rule. To see in which cases robustness rules can be useful, let's consider the following example. Assume that the grammar only lists the verb 'run' as an intransitive verb.

(57) 'John ran the marathon yesterday'

Because the lexicon does not contain the transitive variant of the lexeme 'run', no regular parse will be returned. A fragment parsing approach would probably come up with the following solution:



In this case, 'John' will correctly be identified as the subject of 'ran'. However, no dependencies are established between 'the marathon' and 'ran', or 'yesterday' and 'ran'. The former is hard to establish, due to the missing lexical item. However, the latter should be doable: the grammar identifies 'yesterday' as an adverb that can modify verbs. If we could create a robustness rule that would absorb the object ('the marathon'), it would at least be able to identify the modifier dependency between 'ran' and 'yesterday'.



In other words, a fragment analysis solely combines items at the top level, whereas a robust parser would ideally be able to overcome barriers in both the lower and the higher regions of the chart, meaning that the damage of unrecognised material can be localised and thus minimised. The robustness rules that will be proposed in the following are intended to achieve that.

7.2.2 Restricting and dispreferring robustness rules

The nature of the robustness rules will be such that they will (massively) overgenerate. Therefore, the restriction algorithm from chapter 6 (task-based agenda pruning) will be used to restrict the search space that they will introduce. Robustness rules get a very large, constant penalty in the generative model that underlies the priority model. The consequence is that at first the parser will try to build the parse forest with the restricted set of rules, because tasks involving subtrees with only rules from the standard grammar will always have a higher priority than tasks using an item with a robustness rule application in its subtree. When this is finished, the robustness rules try to fill the gaps. Especially in the *success* and *passive* strategies, tasks with robustness rules are discarded if already enough chart items are found for a particular span, meaning that the parser automatically focuses on those parts of the chart that have not been filled before.

A similar strategy is chosen for the Maximum Entropy-based disambiguation model. When two chart items can be combined using a regular rule and a robustness rule, the disambiguation model prefers the regular rule. This will be accomplished by adding a very large penalty to the MaxEnt score, each time a robustness rule is applied. This penalty is set to be c^4 (see equation 7.1 for the meaning of c), so that when fragment parsing and robustness rules are combined, robust subtrees will be dispreferred more than lexical items, non-root items and items in general. This entails that when the fragment parsing model can choose between combining three fragments without robustness rule applications somewhere in the subtrees, or two fragments with any number of RR

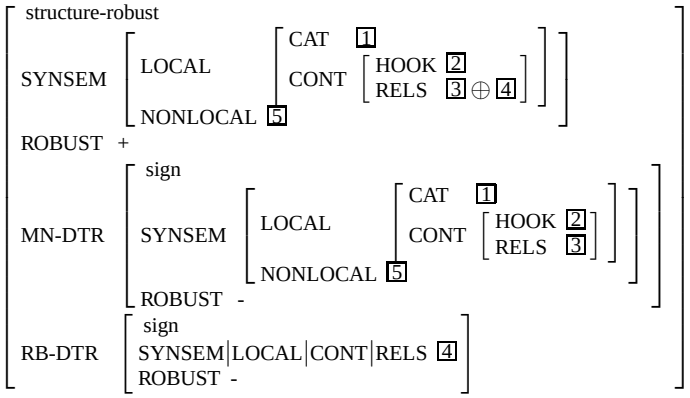


Figure 7.1: Depicted is the TFS that forms the basis for all robustness rules that are used in this chapter.

applications, the fragment parsing model will choose the former, even though more fragments need to be combined. In terms of Optimality Theory, this means that the model considers the avoidance of a RR as the least attractive constraint to violate.

7.2.3 Defining robustness rules

When defining RRs, one must realise that there can be an interaction between the use of RRs and the unification-based packing mechanism. When the chart is built up, items that are subsumed by an existing item are marked as 'frosted', and the latter (more general) item functions as the representative node in the remainder of the parsing process. When unpacking the best solution, the best derivation tree is extracted from the packed forest, which might include a frozen node. Because this frozen node has more constraints than its representative, this derivation tree is not guaranteed to be free of unification failures, and hence, before outputting, this is checked by replaying all the unifications in the derivation tree. This procedure is repeated until a sound, unifying derivation has been found.

What happens when an item in a chart cell is very general? Many nodes will be packed under that item, and hence the chart will remain compact. However, the unpacking process will become problematic, because many of the proposed derivation trees will be incorrect, leading to excessive computation times for the unpacking stage.

Therefore, we chose to define robustness rules such that the resulting chart items will be equally constrained as their daughters. They are all binary, and have one common ancestor in the type hierarchy (see figure 7.1). All rules have a MAIN daughter and a ROBUST daughter. The co-indexations are used for two

purposes. The first is to give the resulting chart item syntactic properties that are identical to its MAIN daughter. The second is to concatenate the relations from both daughters, so that the dependency triples from the ROBUST daughter also percolate up. As can be seen, no new dependency triple is added between the two items when they are combined using a robustness rule. The ROBUST feature is used to prevent the application of two robust rules consecutively. Additional constraints (not shown) make sure that morphological processing is finished, and that both parts are not involved in a coordination.

This definition of *structure-robust* is a blueprint for the robustness rules that are actually used in the experiments. The instantiated rules in the grammar are robustness rule pairs (henceforth: RRPs), which differ in the ordering of the MAIN daughter and the ROBUST daughter. The RRPs that are used in the experiments are given in table 7.2. An RRP is defined by three properties (POS, Sat and NonLoc) for both daughters. 'POS' indicates which parts-of-speech both daughters should have in order to unify. The 'Sat' variable determines whether the daughter is allowed to have non-empty VAL features, such as SUBJ and SUBCAT. A '+' means that the constraint is put in place; a '-' means that no additional constraint is enforced. The variable 'NonLoc' is defined similarly, indicating whether the daughter can have non-empty NON-LOCAL features, such as SLASH and REL.

The set of RRPs is divided in groups of 4 variants (a-d), which differ in the four different combinations of Sat and NonLoc for the ROBUST daughter. RRPs 1 to 4 are distinct in the type of parts-of-speech they accept for both daughters. RRPs 1a-d do not constrain the part-of-speech of the MAIN daughter, whereas the others only accept verbs. RRPs 2a-d do not place restrictions on the ROBUST daughter, whereas RRPs 3a-d and 4a-d only accept verbs and non-verbs, respectively.

7.2.4 Experiments

The first choice that needs to be made is which restriction strategy is chosen. Given the superior results in chapter 5, the lexicalised LOCAL model will be used. However, the counting strategy must still be determined. The differences in the performance between the counting strategies were negligible when the restriction was only used for parse time reduction, but it will be shown later that this is not the case when RRs are used. The SUCCESS counting strategy will be used, with t set to 150, which is a fairly restrictive setting, in order to avoid an unnecessary number of RR applications. The effect of varying t will be investigated later.

Table 7.2 lists coverage results of the grammar, augmented with different

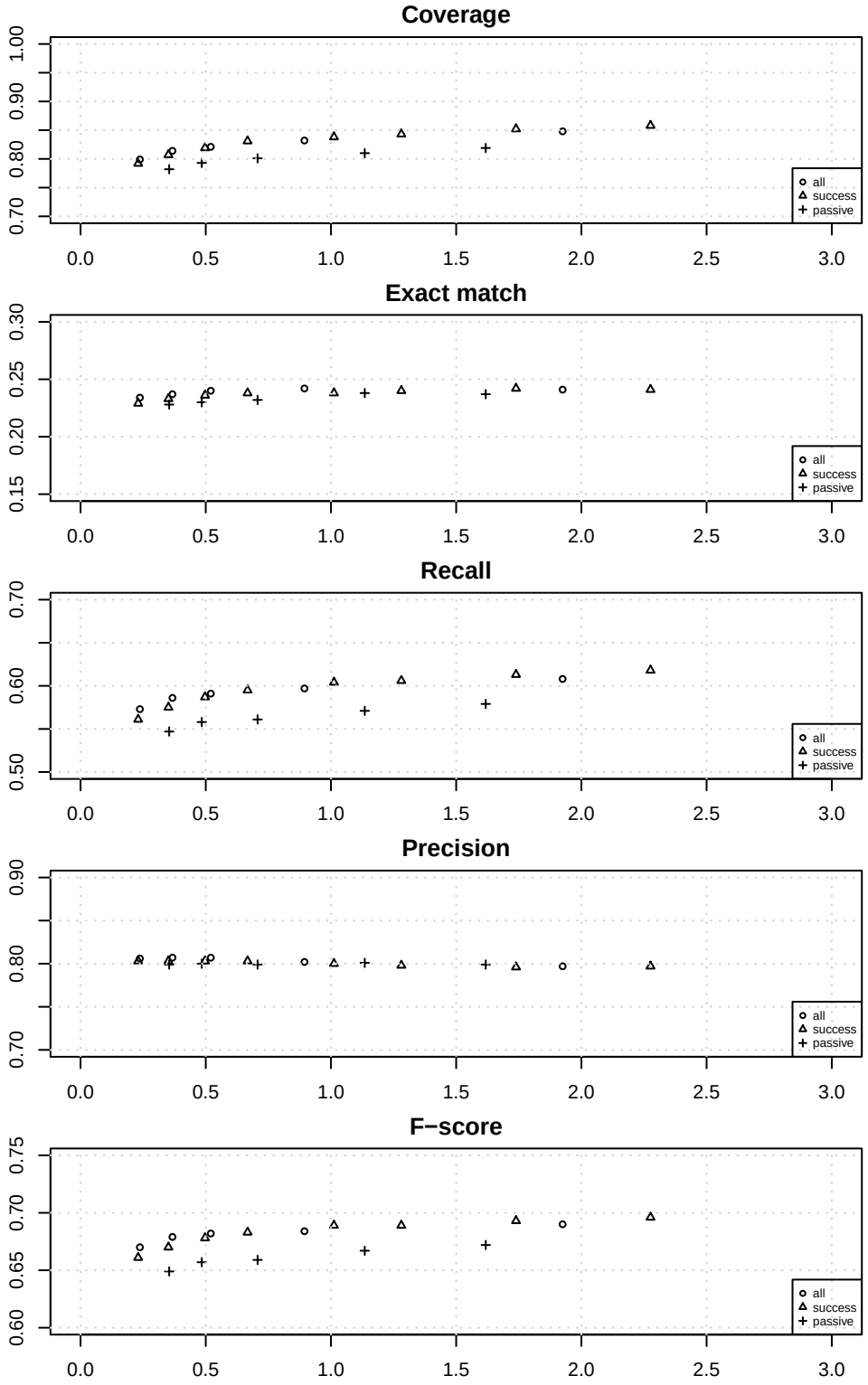


Figure 7.2: Depicted are the results of applying the three different counting strategies on the development set, when the grammar is augmented with RRP 2b.

RRP	MAIN daughter			ROBUST daughter			Coverage
	POS	Sat	NonLoc	POS	Sat	NonLoc	
Baseline	na						63.8%
1a	Any	-	-	Any	-	-	60.6%
1b	Any	-	-	Any	-	+	61.6%
1c	Any	-	-	Any	+	-	63.4%
1d	Any	-	-	Any	+	+	64.5%
2a	Verb	-	-	Any	-	-	83.3%
2b	Verb	-	-	Any	-	+	83.8%
2c	Verb	-	-	Any	+	-	78.5%
2d	Verb	-	-	Any	+	+	76.1%
3a	Verb	-	-	Verb	-	-	71.6%
3b	Verb	-	-	Verb	-	+	71.6%
3c	Verb	-	-	Verb	+	-	71.8%
3d	Verb	-	-	Verb	+	+	69.1%
4a	Verb	-	-	Non-verb	-	-	78.4%
4b	Verb	-	-	Non-verb	-	+	78.1%
4c	Verb	-	-	Non-verb	+	-	73.4%
4d	Verb	-	-	Non-verb	+	+	73.2%

Table 7.2: This table shows how the different robustness rule pairs are defined. Three properties of both the MAIN daughter and the ROBUST daughter are shown: 'POS' indicates which parts-of-speech are allowed in either of the two daughters; 'Sat' indicates whether a saturated VAL feature is enforced; 'NonLoc' indicates whether the daughter is required to have empty NON-LOCAL features. The rightmost column shows the proportion of the sentences in the development set that the augmented grammar covers. The LOCAL pruning strategy is used, with the SUCCESS counting strategy, using $t = 150$.

RRPs, on the development set. Models 1a-d, in which the part-of-speech of the MAIN daughter is not constrained, turn out to have too permissive constraints. In fact, the recorded scores are below the baseline (the grammar without RRPs)[3]. The other RRPs, which require the MAIN daughter to be headed by a verb, perform better. RRPs 2a-2d give the best scores, with, at best, an increase of 20 percent points in coverage. RRPs 3a-d and 4a-d show that most of the increase in coverage can be attributed to combinations of verbal phrases and non-verbal phrases (RRPs 4a-d), compared to combining verbal phrases to other verbal phrases (RRPs 3a-d). Constraining whether the ROBUST daughter is allowed to have non-empty VAL features influences the increase in coverage. However, placing restrictions on the NON-LOCAL features has virtually no effect, perhaps due to the fact that in most cases when a chart item with a non-empty NON-LOCAL feature is present, a similar chart item that does contain an empty NON-LOCAL feature is likely to exist as well.

In the end, RRP 2b is chosen, in which no constraints on the VAL feature are enforced, but in which restrictions are put on the NONLOCAL feature. Figure

[3]This seems to be a bug in the PET parser. It was checked that the ordering of tasks strictly followed the expected pattern, in which all regular tasks were carried out first, after which the RRPs come into play. In theory, this should only slow down the parser, but should not have a negative effect on the coverage. The bug turned out to be somewhere in the interaction between the packing mechanism and the RRPs, but the exact cause of the bug was not discovered. The bug also occurs with the other RRPs, but less frequently.

Robustness strategy	None	FP	RRP2b	RRP2b + FP
Counting strategy	ALL	ALL	SUCCESS	SUCCESS
t	750	750	150	150
Sentence-level				
Avg parse time	0.42s	0.42s	0.99s	1.02s
Coverage	61.9%	99.8%	81.3%	99.6%
Exact match	21.7%	22.0%	20.6%	20.9%
Dependency-level				
Recall	42.5%	59.3%	58.1%	69.1%
Precision	83.3%	79.0%	80.0%	77.6%
F-score	56.3%	67.7%	67.3%	73.1%

Table 7.3: This table shows how augmenting RRP 2b to the grammar affect the results on the test set, either in conjunction with fragment parsing (FP) or not. The definition of the RRP can be found in table 7.2. For ease of comparison, the results of the parser without the robustness rules (section 7.1) are also included.

7.2 shows the time/quality trade-off for this particular RRP, using the different counting strategies (the different curves) and the setting of t (the individual points on the curves). As already alluded to previously, the PASSIVE strategy scores significantly worse than the other two strategies (ALL and SUCCESS). None of the three strategies seems to converge to a maximum coverage score. With higher t, the dependency precision declines slightly.

Now that the influence of the counting strategy and the parameter t can be assessed properly, the SUCCESS counting strategy is chosen, t being set to 150 (equal to the experiments listed in table 7.2). These settings are used for parsing the test set, for which the results are listed in table 7.3. The results from table 7.1 are copied into this table to facilitate a comparison of the methods. Compared to the baseline (applying no robustness measure), the robustness rules cover almost 20 percent points more sentences, yielding an increase of 15.6 percent point on recall. However, this is at the cost of two factors: precision and speed. A drop of more than 3 percent points in dependency precision is recorded, and a small drop of exact match readings is also observed. The average time necessary for building the parse forest reaches approximately 1 second, due to the larger search space. Given the over-generating properties of the robustness rules, this is a remarkably low speed penalty, meaning that the restriction algorithm is able to properly constrain the application of the robustness rules. Interestingly, the f-score of using the robustness rules is almost as high as the f-score for fragment parsing (67.3% vs 67.7%), although almost 18.5% less sentences receive an analysis.

The result of applying both robustness rules and fragment parsing is shown in the rightmost column in table 7.3. Similar to applying fragment parsing alone, almost full coverage is achieved. However, on the dependency level, recall is almost 10 percent point higher than with just fragment parsing (at the cost of a penalty of 2.4 percent point on precision). Given that both fragment pars-

Head	Count
Adjective	4
Adverb	2
Coordinator	5
Determiner	7
Name	3
Noun/pronoun	13
Particle	8
Preposition	13
Verb	32
Total	87

Table 7.4: The distribution of the heads of the ROBUST daughters from a sample of 87 robustness rule applications.

ing and robustness rules do not introduce dependencies, it logically follows that the number of chunks that were connected in the setting with both fragment parsing and robustness rules is lower than in the setting using fragment parsing alone. Hence, the intuition that the robustness rules are able to overcome local obstacles (recall the examples in section 7.2.1) appears to be correct. This is an interesting finding, because fragment parsing has been the main type of robustness method so far in the literature.

7.2.5 What the model predicts

Manual inspection shows that the robustness rules do not solve one specific problem. Instead, they function as a catch-all mechanism. To illustrate this, a sample from the development set was taken, in which a robustness rule was applied at least once. For those sentences, a number of basic properties of the ROBUST daughter were recorded. In total, 87 robust rule applications were inspected.

One of the motivations for using robustness rules instead of fragment parsing was the possibility to combine chunks located low in the parse chart. This is indeed confirmed in the sample, as only 13 applications were at the root of the derivation tree. Another interesting trait of the methodology was that in the majority of cases (50), the ROBUST daughter spanned just one word. In the other cases, the ROBUST daughter was usually a small phrase. Another dimension is the head type of the ROBUST daughter. This distribution is listed in table 7.4. Although one category is certainly the largest (verbs), the distribution is fairly flat, showing the inclusive character of the robustness rules.

7.3 Summary

In this chapter, two methods to overcome extra-grammaticality of the input have been investigated. First, a variant of the well-known partial parsing method has been implemented. In contrast to the two models introduced by Zhang *et al.* (2007a), all characteristics of the segmentation are already contained in the score of each subtree, meaning that the optimal solution can be found within a reasonable time span. The shortest path (the combination of fragments incurring the lowest cost according to the cost function) could then be found using a simple variant of the CYK algorithm. On the test set, this resulted in almost full coverage, and an increase of f-score from 56.3% to 67.7%. A small decline of precision was recorded.

The second method to overcome extragrammatical sentences is to use overgenerating hand-crafted robustness rules. The main idea behind this concept is that when an unrecognised object is found, a robustness rule can take up this unrecognised object, after which the regular parsing routine can be resumed. The advantage of this method is that the dependency between items left and right of the unrecognised object can be recovered, which a fragment analysis is not able to do. The restriction algorithm from chapter 6 was used to make sure that the proliferation of the robustness rule applications was constrained and dispreferred properly. A number of experiments were carried out to discover which definitions of the robustness rules worked best, and to determine which parameters of the task pruning algorithm gave the most advantageous time/quality trade-offs. After these parameters were set, the grammar, augmented with the robustness rules, was used to parse the test set. The coverage on the test rose almost by 20 percent points, yielding an f-score of 67.3%. When the robust grammar was combined with fragment parsing, (almost) full coverage was reached again, but the recall rose by almost 10 percent points, compared to fragment parsing alone, indicating that the grammar with the robustness rules indeed manages to recognise dependencies that were impossible to recover when using fragment parsing alone. In this setting, an f-score of 73.1% is achieved. This score is a few percent points below state-of-the-art f-scores as reported by Rehbein and Van Genabith (2009), which are in the range between 75% and 79%.

7.3.1 Future work

Although this work has shown substantial improvements in terms of f-score, a few possible paths of research have not been taken. First, the statistical models (both for the tasks' priorities and the ordering of solutions for the disambigua-

tion model) are fairly unprincipled, as all applications of RRs are penalised equally. A more sophisticated framework for preferring one robustness rule over another might prove beneficial for the effectiveness of the method. This can be combined by writing more specialised robustness rules for specific uncovered phenomena. A very specialised robustness rule (for instance combining an unknown verb/particle combination) would then be preferred over a catch-all robustness rule, as they have been defined in this chapter. Also, one can conceive situations in which one application of a robustness rule is better than a really improbable combination of regular rules. The current model can not cover such situations, because the penalty given to the application of an RR is too high. Statistically more sound priority and disambiguation models could be trained from treebanks using the robustness rules as well. The treebanking methodology of section 5.3 certainly leaves space for this type of modelling.

So far, the robustness rules were agnostic about the dependency that they contributed. However, a fair guess can sometimes be made, based on basic properties (*e.g.* part-of-speech) of the MAIN and ROBUST daughters. For instance, if the MAIN daughter is a verb, and the ROBUST daughter is an accusative noun phrase, the dependency is likely to be OA. Simple, rule-based heuristics could provide such dependencies in the future.

8 Conclusion

The aim of the research that is described in this thesis was to find techniques that improve the feasibility of precision-oriented HPSG parsing. We have looked at several aspects that have negatively influenced a wide adoption of technology from the DELPH-IN framework, such as the time to develop a grammar and an accompanying treebank, modularity and maintainability of the grammar, efficiency and robustness. The strengths and weaknesses on these aspects of a number of grammar engineering paradigms have been outlined in chapter 2.

In chapter 3, a core grammar for German was developed, after an analysis of a subset of linguistic phenomena that occur within the German language and an overview of existing linguistic descriptions of these phenomena. Although no novel analyses have emerged from this exercise, the grammar is a valuable addition in the landscape of (German) deep parsers, being the computational synthesis of existing linguistic analyses available in the literature. Also, a flexible and powerful way to model topological fields by means of finite state automata was introduced, using only the restricted variant of the $\mathcal{TDL}$ formalism that is used in the DELPH-IN tools. A core lexicon was included to account for syntactically idiosyncratic lexemes.

Chapter 4 showed how the core grammar was extended with a deep lexicon, extracted from the Tiger treebank (Brants *et al.* 2002). Like previous studies, a hand-written procedure was written for this purpose. Specifically, attention was paid to deriving deep lexical features, such as complementation patterns, modification constraints and morphological features. The resulting lexicon is comparable in size to the lexicon of an existing HPSG grammar for German (Crysmann 2005), but the level of granularity of the tags was slightly lower.

Apart from extracting a lexicon from it, the Tiger treebank has been used as a gold standard in chapter 5. First, a conversion procedure was set up in order to convert the MRS-shaped output into a dependency-style format, in order to be able to compare to the Tiger treebank's syntactic dependencies. This opened up an array of possibilities. A new regression testing procedure (unit testing) was created, which positively influences the grammar writing scheme by pointing out and quantifying sources of error in the chain of core grammar, lexical acquisition and output conversion. Second, a treebank could be created by parsing the data with the grammar, and exploit the gold standard dependencies to automatically disambiguate between possible analyses. When a small margin of error was tolerated, 56% of the original Tiger treebank could be converted to a full-fledged dynamic HPSG treebank this way. Last, a test set was extracted from the treebank to evaluate the parser's performance on unseen text. Slightly over 60% of the sentence received at least one analysis, of which 21.7% received the

exactly correct parse. On the dependency level, recall and precision scores of 40.8% and 83.7% were recorded, respectively, yielding an f-score of 54.9%.

In the Interlude, the entire grammar engineering paradigm of creation of a core lexicon, deep lexical acquisition and automatic treebank construction was compared to other paradigms on a number of relevant dimensions: linguistic relevance, level of abstraction of the output, ability to generate, efficiency, coverage, development time and clean evaluation. One of the outcomes of the analysis was that the combination of the PET parser and the grammar from the first chapters scores below par on two dimensions: efficiency and coverage.

The lack of efficiency is addressed in chapter 5. In these experiments, the agenda of the PET parser (a priority queue of parser tasks) is manipulated, compared to the exhaustive strategy that is used normally. The priorities of the agenda tasks are based on a generative model of HPSG rule applications. Less likely combinations of rules and chart items receive lower priorities, and are either deferred or even deleted. A number of pruning strategies and counting strategies are tried. The experiments clearly indicated that the LOCAL pruning strategy produced superior results compared to the alternatives. Changing the counting strategy has virtually no effect when the LOCAL pruning strategy is used. In the most optimal setting (which is determined in experiments on the development set), the average time to produce the packed forest (on the test set from the Tiger treebank) is reduced from 3.9 seconds to 0.2 seconds, a speedup of around 95%, yielding equal f-scores. A small increase of f-score (+1.6%) is measured when the cell size is increased slightly, due to the lower number of timeouts.

In chapter 6, experiments aiming to reduce the fragility of the parser are carried out. As a baseline, a fragment parsing algorithm has been implemented, which has been the *de facto* fall-back standard in deep parsers facing similar issues. These experiments yielded coverage scores approximating 100% when applied on the test set, with an f-score of 67.7%. As an alternative, hand-written, massively over-generating *robustness rules* have been introduced. The basic intuition of the robustness rules is that unrecognised material can be taken up by the robustness rule, after which the regular parsing routine can be resumed, minimising the damage of the unrecognised obstacle. The over-generation of the robustness rules is constrained by the agenda pruning algorithm from the previous chapter. After exploring the effectiveness of a number of definitions for these robustness rules, one particular definition was chosen, and applied to the test set. This yielded a coverage score of 81.3%, but an f-score approaching the one of the fragment parsing approach (67.3%), despite the significantly lower number of sentences receiving any analysis at all. Combining both approaches yielded the best results, with almost full coverage and an f-score of 73.1%, which approximates state-of-the-art results for hand-written and DGE

parsers of German.

8.1 Directions for future work

The experiments have yielded insights in several aspects of precision-oriented HPSG parsing, but there are ample possibilities for future research, building on the research outlined in this thesis. A number of possible extensions have already been outlined in the summaries of the individual chapters, but I would like to highlight two other directions here.

Currently, the grammar's constructional coverage is too low to be considered wide-coverage. A number of uncovered phenomena are already listed in section 3.4, but it might also be insightful to study more closely the analyses listed by Dipper (2003). An extension of the core grammar, including a more principled approach to its morphology, is therefore a necessary step to improve the grammar. Apart from extending the grammar's coverage, one might opt to restore the semantic nature of the MRS structures as output of the grammar. This would entail that the direct link between the grammar and the source treebank is removed, but this link is not needed anymore for updating the treebank (given that it is a discriminant-based, dynamic treebank). However, an alternative form of evaluation would have to be shaped, because no straight evaluation against the Tiger treebank's syntactic dependencies is possible anymore.

Another interesting direction of research comes to mind as well, integrating the technology from each individual chapter. In the introduction in chapter 1, I already alluded to the tension between two schools of parser engineering: statistical parsing and grammar-based parsing. The grammar that has been created in chapters 3-5 is a blend between these schools, as it is the combination of a hand-written core grammar, a lexicon derived using symbolic learning and a disambiguation model based on statistical learning. However, no answer has been given to the question to which degree the 'ideal' parser is hand-written or learnt. In the paradigm that has been laid out in this thesis, it becomes possible to test this systematically. Aspects of the language that are now handled by the formal model can be left to the statistical model by removing or relaxing constraints from the core grammar. Doing this would have caused an unacceptable computational burden, but the task pruning mechanism assures that the computational cost remains within bounds.

A Test suite

What follows is a copy of the test suite. The marks on the left indicate whether the sentence is currently accepted by the grammar or not.

- x Es gibt Käse
- x Der Käse stinkt
- x Antje isst den Käse
- x Antje schenkt mir den Käse
- x Der Käse ist Käse
- x Der Käse ist herrlich
- x Antje freut sich auf den Käse
- x Antje freut sich darauf
- x Antje schlägt Käse vor
- x Antje denkt dass der Käse herrlich ist
- x Antje sagt der Käse stinkt
- x Antje weiß wo der Käse liegt
- x Antje weiß wer stinkt
- x Antje weiß was Peter isst
- x Antje weiß auf welchen Käse Peter sich freut
- x Antje weiß mit welcher Butter Peter das Brot isst
- x Bestimmt stinkt der Käse
- x Seit gestern stinkt der Käse
- x Weil es Käse gibt isst Antje den Käse
- x Antje schenkt mir den Käse ohne ihn zu essen
- x Den Käse schenkt Antje mir
- x Mir schenkt Antje den Käse
- x Antje versucht den Käse zu essen
  Den Käse versucht Antje zu essen
- x Bestimmt versucht Antje den Käse zu essen
- x Antje will den Käse essen
- x Den Käse will Antje essen
- x Antje sieht mich den Käse essen
- x Mich sieht Antje den Käse essen
- x Den Käse sieht Antje mich essen
- x Den Käse essen sieht Antje mich
- x Antje hat den Käse gegessen
- x Der Käse wird von Antje gegessen
- x Der Käse soll von Antje gegessen werden
- x Der Käse ist von Antje gegessen worden
- x Antje hat den Käse zu essen versucht

Antje hat den Käse versucht zu essen
x Antje hat versucht den Käse zu essen
x Antje hat den Käse essen wollen
Peter denkt dass Antje den Käse wird essen können
x Peter denkt dass Antje den Käse essen können wird
x Der Käse ist herrlicher Käse gewesen
x Isst Antje den Käse
x Wird Antje den Käse essen
x Wo isst Antje den Käse
x Welchen Käse isst Antje
x Wer isst den Käse
x Und der Käse stinkt
x Käse isst Antje nur mit Brot
x Dem Mann zufolge isst Antje den Käse
x Selbst der herrliche Käse von Antje stinkt
Der gegessene Käse stinkt
Der von Antje gegessene Käse stinkt
x Antje der Käse stinkt
x Der Käse des Mannes stinkt
x Frau Antje stinkt
x Dieses Mal isst Antje den Käse
x Antje isst dieses Mal den Käse
x Antje hat keine Ahnung wo der Käse liegt
x Antje hat keine Neigung den Käse zu essen
x Der Kampf um den herrlichen Käse
x Antje hat Sorgen dass der Käse stinkt
x Die 3 liegen
x Der Käse den Antje isst stinkt
x Der Käse auf den Antje sich gefreut hat stinkt
x Antje isst den Käse der stinkt
x Das Brot mit dem Antje den Käse isst stinkt
x Der Mann dessen Brot Antje gegessen hat stinkt
x Die Frau deren Brot Antje gegessen hat stinkt
x Die Männer deren Brot Antje gegessen hat stinken
x Der Mann auf dessen Brot Antje sich freut stinkt
Der Käse ist eine Woche alt
Das ist sich entwickelndener Käse
Das ist der mir zustehende Käse
x Der Käse ist bestimmt fast unglaublich herrlich

- x Der Käse stinkt und das Brot stinkt
- x Der Käse stinkt und Antje will den Käse essen
  Stinkt der Käse und will Antje den Käse essen
  Antje stinkt und will den Käse essen
- x Antje will mir den Käse schenken und das Brot essen
- x Bestimmt will Antje mir den Käse schenken und das Brot essen
- x Antje und Peter essen den Käse
- x Antje isst den Käse und das Brot
- x Der Käse und Käse stinkt
- x Der Käse liegt bei dem Brot und neben der Butter
- x Der Käse ist herrlich und kostbar
- x Der herrliche und kostbare Käse stinkt
- x Früher oder später wird Antje den Käse essen
- x Vier- bis fünfhundert Männer
- x Im Kindes- und Jugendalter
  Was denkt Antje dass sie gegessen hat
  Wer denkt Antje dass den Käse gegessen hat
  Wie denkt Antje dass Peter den Käse gegessen hat
- x Wer sagt Peter hat den Käse gegessen
- x Was sagt Peter hat Antje gegessen
- x Wie sagt Peter hat Antje den Käse gegessen
- x Wir sagt Peter haben den Käse gegessen
- x Den Käse sagt Peter haben wir gegessen
- x Bestimmt sagt Peter haben wir den Käse gegessen
- x Peter hat den Käse gegessen auf den Antje sich gefreut hat
- x Peter hat sich auf den Käse gefreut den Antje gegessen hat
- x Peter hat die Neigung den Käse zu essen auf den Antje sich freut
- x Antje denkt dass dein Käse herrlicher ist als mein Käse
- x Antje denkt dass Peter herrlicheren Käse isst als ich
- x Antje hat herrlicheren Käse als du gegessen
- x Antje hat herrlicheren Käse gegessen als du

Bibliography

ADOLPHS, P., S. OEPEN, U. CALLMEIER, B. CRYSMANN, D. FLICKINGER, and B. KIEFER. 2008. Some fine points of hybrid natural language parsing. In *Proceedings of LREC*, 1380–1387.

BALDWIN, T., E.M. BENDER, D. FLICKINGER, A. KIM, and S. OEPEN. 2004. Road-testing the English Resource Gammar over the British National Corpus. In *Proceedings of LREC*, 2047–2050.

BANGALORE, S., and A.K. JOSHI. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics* 25.237–265.

BENDER, E.M. 2008a. Evaluating a crosslinguistic grammar resource: a case study of Wambaya. In *Proceedings of ACL-HLT*, 977–985.

—— 2008b. Grammar engineering for linguistic hypothesis testing. In *Proceedings of the Texas Linguistics Society X Conference: Computational Linguistics for Less-Studied Languages*, 16–36.

——, and D. FLICKINGER. 2005. Rapid prototyping of scalable grammars: Towards modularity in extensions to a language-independent core. In *Proceedings of IJCNLP*, 203–208.

——, D. FLICKINGER, and S. OEPEN. 2002. The Grammar Matrix: An Open-Source Starter-Kit for the Rapid Development of Cross-Linguistically Consistent Broad-Coverage Precision Grammars. In *Proceedings of the Workshop on Grammar Engineering and Evaluation at CoNLL*, 8–14.

——, D. FLICKINGER, S. OEPEN, A. WALSH, and T. BALDWIN. 2004. Arboretum: Using a precision grammar for grammar checking in CALL. In *InSTIL/ICALL Symposium*, 83–86.

——, I.A. SAG, and T. WASOW. 2003. *Syntactic theory: A formal introduction*. CLSI Publications.

BOND, F., S. OEPEN, M. SIEGEL, A. COPESTAKE, and D. FLICKINGER. 2005. Open source machine translation with DELPH-IN. In *Open-Source Machine Translation: Workshop at MT Summit X*, 15–22.

BOUMA, G., and J. SPENADER. 2009. The distribution of weak and strong object reflexives in dutch. In *Proceedings of the seventh workshop on Treebanks and Linguistic Theory (TLT 7), Groningen*, 103–114.

——, G. VAN NOORD, and R. MALOUF. 2000. Alpino: wide-coverage computational analysis of dutch. In *Proceedings of Computational Linguistics in the Netherlands (CLIN)*, 45–59.

——, G. VAN NOORD, and R. MALOUF. 2001. Alpino: Wide-coverage computational analysis of Dutch. *Language and Computers* 37.45–59.

BRANCO, A.H., and F. COSTA. 2008. A Computational Grammar for Deep Linguistic Processing of Portuguese: LXGram, version A. 4.1. Technical report, Department of Informatics, University of Lisbon.

BRANTS, S., S. DIPPER, S. HANSEN, W. LEZIUS, and G. SMITH. 2002. The TIGER Treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, 24–41.

BRANTS, T. 2000. TnT: a statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, 224–231.

BRESNAN, J. 1982. *The mental representation of grammatical relations*. The MIT Press.

—— 2001. *Lexical-functional syntax*. Wiley-Blackwell.

BRISCOE, T. 2006. An introduction to tag sequence grammars and the RASP system parser. Technical report, University of Cambridge.

——, and J. CARROLL. 2002. Robust accurate statistical annotation of general text. In *Proceedings of LREC*, 1499–1504.

——, and ——. 2006. Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proceedings of COLING-ACL*, 41–48.

——, ——, and R. WATSON. 2006. The second release of the RASP system. In *Proceedings of COLING-ACL*, 77–80.

BURNARD, L. 2000. Users reference guide for the British National Corpus. Technical report, Technical report, Oxford University Computing Services.

BUTT, M., H. DYVIK, T.H. KING, H. MASUICHI, and C. ROHRER. 2002. The parallel grammar project. In *International Conference On Computational Linguistics*, 1–7.

CAHILL, A., M. BURKE, M. FORST, R. ODONOVAN, C. ROHRER, J. VAN GENABITH, and A. WAY. 2005. Treebank-based acquisition of multilingual unification grammar resources. *Research on Language & Computation* 3.247–279.

——, M. BURKE, R. O'DONOVAN, S. RIEZLER, J. VAN GENABITH, and A. WAY. 2008a. Wide-coverage deep statistical parsing using automatic dependency structure annotation. *Computational Linguistics* 34.81–124.

——, M. BURKE, R. ODONOVAN, J. VAN GENABITH, and A. WAY. 2004. Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of ACL*, 320–327.

——, T.H. KING, and J.T. MAXWELL III. 2007. Pruning the Search Space of a Hand-Crafted Parsing System with a Probabilistic Parser. In *Proceedings of the Workshop on Deep Linguistic Processing at ACL*, 65–72.

——, J.T. MAXWELL III, P. MEURER, C. ROHRER, and V. ROSÉN. 2008b. Speeding up LFG parsing using c-structure pruning. In *Proceedings of the Workshop on Grammar Engineering Across Frameworks at COLING*, 33–40.

——, M. MCCARTHY, J. VAN GENABITH, and A. WAY. 2002. Automatic annotation of the Penn Treebank with LFG F-structure information. In *Workshop on Linguistic Knowledge Acquisition and Representation-Bootstrapping Annotated Language Data at LREC*, 8–15.

——, and J. VAN GENABITH. 2006. Robust PCFG-based generation using automatically acquired LFG approximations. In *Proceedings of ICCL-ACL*, 1033–1040.

CALLMEIER, U. 2000. PET–a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering* 6.99–107.

CARPENTER, B. 1992. *The Logic of Typed Feature Structures: With Applications to Unification Grammars, Logic Programs, and Constraint Resolution*. Cambridge, UK: Cambridge University Press.

CARROLL, J., T. BRISCOE, and A. SANFILIPPO. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of LREC*, 447–454.

——, G. MINNEN, and T. BRISCOE. 1999. Corpus annotation for parser evaluation. In *Proceedings of the workshop on Linguistically Interpreted Corpora (LINC) at EACL*, 35–41.

CARTER, D. 1997. The TreeBanker. A tool for supervised training of parsed corpora. In *Proceedings of the Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, 9–15.

CHARNIAK, E. 1996. Tree-Bank Grammars. In *Proceedings of the National Conference on Artificial Intelligence*, 1031–1036.

—— 2000. A maximum-entropy-inspired parser. In *ACM International Conference Proceeding Series*, 132–139.

——, and M. JOHNSON. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of ACL*, 173–180.

CHIANG, D. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. In *Proceedings of ACL*, 456–463.

CLARK, S., and J.R. CURRAN. 2004a. Parsing the WSJ using CCG and log-linear models. In *Proceedings of ACL*, 104–111.

CLARK, STEPHEN, and JAMES CURRAN. 2004b. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING*, 282–288.

COLLINS, MICHAEL. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of ACL*, 16–23.

COPESTAKE, A. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Publications, Stanford, CA, USA.

—— 2007. Semantic composition with (robust) minimal recursion semantics. In *Proceedings of the Workshop on Deep Linguistic Processing at ACL*, 73–80.

——, D. FLICKINGER, C. POLLARD, and I. SAG. 2005. Minimal Recursion Semantics: An Introduction. *Research on Language & Computation* 3.281–332.

CRAMER, B., and Y. ZHANG. 2009. Construction of a German HPSG grammar from a detailed treebank. In *Proceedings of the GEAF workshop at ACL-IJCNLP*, 37–45.

——, and ——. 2010. Constraining robust constructions for broad-coverage parsing with precision grammars. In *Proceedings of COLING*, 223–231.

CRYSMANN, B. 2003. On the efficient implementation of German verb placement in HPSG. In *Proceedings of RANLP*, 112–116.

—— 2005. Relative Clause Extraposition in German: An Efficient and Portable Implementation. *Research on Language & Computation* 3.61–82.

——, N. BERTOMEU, P. ADOLPHS, D. FLICKINGER, and T. KLÜWER. 2008. Hybrid processing for grammar and style checking. In *Proceedings of COLING*, 153–160. Association for Computational Linguistics.

DIPPER, S. 2000. Grammar-based corpus annotation. In *Workshop on Linguistically Interpreted Corpora (LINC) at COLING*, 56–64.

—— 2003. Implementing and Documenting Large-scale Grammars–German LFG. *Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung (AIMS)* 9.

DRELLISHAK, S., 2009. *Widespread but Not Universal: Improving the Typological Coverage of the Grammar Matrix*. University of Washington dissertation.

DRIDAN, R., 2009. *Using lexical statistics to improve HPSG parsing*. Saarland University dissertation.

——, V. KORDONI, and J. NICHOLSON. 2008. Enhancing performance of lexicalised grammars. In *Proceedings of ACL-08: HLT*, 613–621. Association for Computational Linguistics.

ERBACH, G. 1991. *A flexible parser for a linguistic development environment*, 74–87. Springer.

FLICKINGER, D. 2000. On building a more effcient grammar by exploiting types. *Natural Language Engineering* 6.15–28.

——, S. OEPEN, and G. YTRESTØL. 2010. Wikiwoods: Syntacto-semantic annotation for english wikipedia. In *Proceedings of LREC*, 1665–1671.

FORST, M. 2007. Filling statistics with linguistics: property design for the disambiguation of german lfg parses. In *Proceedings of the Workshop on Deep Linguistic Processing at ACL*, 17–24. Association for Computational Linguistics.

——, N. BERTOMEU, B. CRYSMANN, F. FOUVRY, S. HANSEN-SCHIRRA, and V. KORDONI. 2004. Towards a dependency-based gold standard for German parsers–The TiGer Dependency Bank. In *Proceedings of LINC at COLING*.

FRANCIS, W.N., H. KUČERA, and A.W. MACKIE. 1982. *Frequency analysis of English usage: Lexicon and grammar*. Houghton Mifflin Harcourt.

GOODMAN, J.T., and S. SHIEBER, 1998. *Parsing inside-out*. Harvard University, Cambridge, MA dissertation.

HALL, J., J. NILSSON, J. NIVRE, G. ERYIGIT, B. MEGYESI, M. NILSSON, and M. SAERS. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, 933–939.

——, and J. NIVRE. 2008. A dependency-driven parser for German dependency and constituency representations. In *Proceedings of the Workshop on Parsing German at ACL*, 47–54.

HELLAN, L., and P. HAUGEREID. 2003. NorSource: An exercise in Matrix grammar-building design. In *Proc. the Workshop on Ideas and Strategies for Multilingual Grammar Development, ESSLLI 2003*, 41–48.

HINDLE, D. 1989. Acquiring disambiguation rules from text. In *Proceedings of ACL*, 118–125.

HINRICHS, E., and T. NAKAZAWA. 1994. *Linearizing AUXs in German verbal complexes*, volume 46, 11–37. Center for the Study of Language and Informatics.

HOCKENMAIER, J. 2006. Creating a CCGbank and a Wide-Coverage CCG Lexicon for German. In *Proceedings of ACL*, 505–512.

——, and M. STEEDMAN. 2002. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of LREC*, 1974–1981.

——, and ——. 2007. CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics* 33.355–396.

JOHNSON, M., T.L. GRIFFITHS, and S. GOLDWATER. 2007. Adaptor grammars: A framework for specifying compositional nonparametric Bayesian models. *Advances in Neural Information Processing Systems* 19.641.

JOSHI, A.K., and Y. SCHABES. 1997. Tree-adjoining grammars. *Handbook of formal languages* 3.69–124.

KAPLAN, R.M., S. RIEZLER, T.H. KING, J.T. MAXWELL III, A. VASSERMAN, and R. CROUCH. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of HLT-NAACL*.

KASAMI, T. 1965. An efficient recognition and analysis algorithm for context-free languages. Technical report, AF-CRL-65-758, Air Force Cambridge Research Laboratory, MA.

KASPER, W., B. KIEFER, H.U. KRIEGER, C.J. RUPP, and K.L. WORM. 1999. Charting the depths of robust speech parsing. In *Proceedings of ACL*, 405–412.

KATHOL, ANDREAS, 1995. *Linearization-Based German Syntax*. Ohio State University dissertation.

KAY, M. 1980. *Algorithm schemata and data structures in syntactic processing*, 35–70. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

KIEFER, B., H.U. KRIEGER, J. CARROLL, and R. MALOUF. 1999. A bag of useful techniques for efficient and robust parsing. In *Proceedings of ACL*, 473–480.

KING, T.H., R. CROUCH, S. RIEZLER, M. DALRYMPLE, and R. KAPLAN. 2003. The PARC 700 dependency bank. In *Proceedings of the International Workshop on Linguistically Interpreted Corpora (LINC) at EACL*, 1–8.

KORDONI, V., and J. NEU. 2005. Deep analysis of Modern Greek. In *Lecture Notes in Computer Science*, 674–683.

KRIEGER, H.U., and U. SCHÄFER. 1994. TDL: a type description language for constraint-based grammars. In *Proceedings of COLING*, 893–899.

KÜBLER, S., E.W. HINRICHS, and W. MAIER. 2006. Is it really that difficult to parse German? In *Proceedings of EMNLP*.

LIDSTONE, G.J. 1920. Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty of Actuaries* 8.80.

LIN, D. 1998. A dependency-based method for evaluating broad-coverage parsers. *Natural Language Engineering* 4.97–114.

MAGERMAN, D. 1995. Statistical decision-tree models for parsing. In *Proceedings of ACL*, 276–283.

MALOUF, R., and G. VAN NOORD. 2004. Stochastic attribute value grammars. In *IJCNLP-04 Workshop Beyond Shallow Analyses-Formalisms and statistical modeling for deep analyses*.

MARCUS, M.P., B. SANTORINI, and M.A. MARCINKIEWICZ. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19.313–330.

MARIMON, M., N. BEL, S. ESPEJA, and N. SEGHEZZI. 2007. The Spanish Resource Grammar: pre-processing strategy and lexical acquisi-tion. In *Proceedings of the workshop on Deep Linguistic Processing at ACL*, 105–111.

MATSUZAKI, T., Y. MIYAO, and J. TSUJII. 2007. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1671–1676.

MAXWELL, J.T., and R.M. KAPLAN. 1993. The interface between phrasal and functional constraints. *Computational Linguistics* 19.571–590.

MCDONALD, R., F. PEREIRA, K. RIBAROV, and J. HAJIC. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT/EMNLP*, 523–530.

MIYAO, Y. 1999. Packing of feature structures for efficient unification of disjunctive feature structures. In *Proceedings of ACL*, 579–584.

——, 2006. *From Linguistic Theory to Syntactic Analysis: Corpus-Oriented Grammar Development and Feature Forest Model*. University of Tokyo dissertation.

——, T. NINOMIYA, and J. TSUJII. 2004. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of IJCNLP*.

MÜLLER, S, 1997. *Spezifikation und Verarbeitung deutscher Syntax in Head-Driven Phrase Structure Grammar*. Saarland University dissertation.

MÜLLER, S. 2002. *Complex Predicates: Verbal Complexes, Resultative Constructions, and Particle Verbs in German*. Stanford, CA, USA: CSLI Publications.

MÜLLER, STEFAN, and WALTER KASPER. 2000. HPSG analysis of German. In *Verbmobil: Foundations of Speech-to-Speech Translation*, ed. by Wolfgang Wahlster, Artificial Intelligence, 238–253. Springer.

NAKANISHI, H., Y. MIYAO, and J. TSUJII. 2005. Probabilistic models for disambiguation of an HPSG-based chart generator. In *Proceedings of the International Workshop on Parsing Technology*, 93–102.

NERBONNE, J.A., K. NETTER, and C.J. POLLARD. 1994. *German in Head-driven Phrase Structure Grammar*. Center for the Study of Language and Inf.

NINOMIYA, T., T. MATSUZAKI, Y. TSURUOKA, Y. MIYAO, and J. TSUJII. 2006. Extremely lexicalized models for accurate and fast HPSG parsing. In *Proceedings of EMNLP*, 155–163.

——, Y. TSURUOKA, Y. MIYAO, and J. TSUJII. 2005. Efficacy of beam thresholding, unification filtering and hybrid parsing in probabilistic HPSG parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, 103–114.

NIVRE, J. 2007. Inductive dependency parsing. *Computational Linguistics* 33.

OEPEN, S., and J. CARROLL. 2000a. Ambiguity packing in constraint-based parsing: practical results. In *Proceedings of NAACL*, 162–169.

——, and ——. 2000b. Parser engineering and performance profiling. *Natural Language Engineering* 6.81–97.

——, D. FLICKINGER, K. TOUTANOVA, and C.D. MANNING. 2004. LinGO Redwoods. *Research on Language & Computation* 2.575–596.

PEREIRA, F.C.N., and D.H.D. WARREN. 1980. Definite clause grammars for language analysis–a survey of the formalism and a comparison with augmented transition networks. *Artificial intelligence* 13.231–278.

POLLARD, C.J., and I.A. SAG. 1994. *Head-Driven Phrase Structure Grammar*. Chicago, IL, USA: University Of Chicago Press.

PRINCE, A., and P. SMOLENSKY. 2004. Optimality Theory: Constraint interaction in generative grammar. In *The 12th West Coast Conference on Formal Linguistics*.

REHBEIN, I., 2009. *Treebank-based grammar acquisition for German*. Dublin City University dissertation.

——, and J. VAN GENABITH. 2009. Automatic acquisition of LFG resources for German-as good as it gets. In *Proceedings of LFG-09*. CSLI Publications.

RIEZLER, S., T.H. KING, R.M. KAPLAN, R. CROUCH, J.T. MAXWELL III, and M. JOHNSON. 2001. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of ACL*, 271–278.

ROHRER, C., and M. FORST. 2006. Improving coverage and parsing quality of a large-scale lfg for german. In *Proceedings of LREC*.

ROSÉN, V., P. MEURER, and K. DE SMEDT. 2009. LFG Parsebanker: A toolkit for building and searching a treebank as a parsed corpus. In *Proceedings of the Seventh International Workshop on Treebanks and Linguistic Theories (TLT7)*, 127–133.

SIEGEL, M., and E.M. BENDER. 2002. Efficient deep processing of Japanese. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization*.

SKUT, W., B. KRENN, T. BRANTS, and H. USZKOREIT. 1997. An annotation scheme for free word order languages. In *Proceedings of the fifth conference on Applied natural language processing*, 88–95.

STEEDMAN, M. 2000. *The Syntactic Process*. Cambridge, MA, USA: MIT Press.

TANAKA, T., F. BOND, S. OEPEN, and S. FUJITA. 2005. High Precision Treebanking-Blazing Useful Trees Using POS Information. In *Proceedings of ACL*, 994–997.

TOUTANOVA, K., C.D. MANNING, D. FLICKINGER, and S. OEPEN. 2005. Stochastic HPSG parse disambiguation using the Redwoods corpus. *Research on Language & Computation* 3.83–105.

——, C.D. MANNING, S. SHIEBER, D. FLICKINGER, and S. OEPEN. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of the First Workshop on Treebanks and Linguistic Theories*, 253–263.

VAN DER BEEK, L., G. BOUMA, R. MALOUF, and G. VAN NOORD. 2002. The Alpino dependency treebank. In *Proceedings of Computational Linguistics in the Netherlands (CLIN) 2001*, 8–22.

VAN NOORD, G. 1997. An efficient implementation of the head-corner parser. *Computational Linguistics* 23.425–456.

—— 2006. At last parsing is now operational. In *Verbum ex machina: actes de la 13e conférence sur le traitement automatique des langues naturelles (TALN 2006): Leuven, 10-13 avril 2006*, 20–42.

—— 2007. Using self-trained bilexical preferences to improve disambiguation accuracy. In *Proceedings of the 10th International Conference on Parsing Technologies*, 1–10.

VAN NOORD, G. 2009. Learning efficient parsing. In *Proceedings of EACL*, 817–825. Association for Computational Linguistics.

——, G. BOUMA, R. KOELING, and M.J. NEDERHOF. 1999. Robust grammatical analysis for spoken dialogue systems. *Natural language engineering* 5.45–93.

VELLDAL, E., and S. OEPEN. 2005. Maximum entropy models for realization ranking. In *Proceedings of the 10th MT-Summit (X)*.

WAHLSTER, W. 2000. *Verbmobil: foundations of speech-to-speech translation*. Springer verlag.

XIA, F., M. PALMER, and A. JOSHI. 2000. A uniform method of grammar extraction and its applications. In *Proceedings of EMNLP*, 53–62.

YOUNGER, D.H. 1967. Recognition and parsing of context-free languages in time n3*. *Information and control* 10.189–208.

YTRESTØL, G., D. FLICKINGER, and S. OEPEN. 2009. Extracting and Annotating Wikipedia Sub-Domains. In *Proceedings of the Seventh International Workshop on Treebanks and Linguistic Theories*, 185–197.

ZHANG, Y., B-G. AHN, S. CLARK, C. VAN WYK, CURRAN J., and L. RIMELL. 2010a. Chart Pruning for Fast Lexicalised-Grammar Parsing. In *Proceedings of COLING*.

——, and V. KORDONI. 2008. Robust parsing with a large HPSG grammar. In *Proceedings of LREC*.

——, ——, and E. FITZGERALD. 2007a. Partial parse selection for robust deep processing. In *Proceedings of the Workshop on Deep Linguistic Processing at ACL*, 128–135.

——, and T. MATSUZAKI. 2009. Junichi Tsujii. 2009. HPSG supertagging: A sequence labeling view. In *Proceedings of IWPT*, 210–213.

——, ——, and J. TSUJII. 2010b. A simple approach for HPSG supertagging using dependency information. In *Proceedings of NAACL-HLT*, 645–648.

——, S. OEPEN, and J. CARROLL. 2007b. Efficiency in Unification-Based N-Best Parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, 48–59.

——, R. WANG, and S. OEPEN. 2009. Hybrid Multilingual Parsing with HPSG for SRL. In *Proceedings of CoNLL*.

ZINSMEISTER, H., J. KUHN, and S. DIPPER. 2002. TIGER TRANSFER-Utilizing LFG Parses for Treebank Annotation. In *Proceedings of the LFG02 Conference*.